**AMAZON ONLINE SHOPPING**

**PROJECT TITLE: AMAZON (RETAIL)**

**COURE NUMBER AND SECTION: 6360.001**

**TEAM NUMBER: AMAZON-1**

**TEAM MEMBERS: NAMITHA MYSORE SHESHADRI(NMS190006)**

**NIKESH MANJUNATH (NXM190054)**

**SAHANA SUBRAMANYAM(SXS200210)**

**DATA REQUIREMENTS:**

* Amazon is an online e-commerce website that deals with several kind of products which can be supplied to customers as per their requirement
* A Customer should have sufficient details likes email id, phone no, Name, Sex, age and default address and payment mode
* A Customer can have multiple addresses and multiple payment modes.
* Products contains productID, Name, Price, Description, Category, Product Information, Quantity, Seller Id, Customer Question and Answer, Customer Reviews. Each Product has a single price
* A Customer can add items to cart before placing the order. Each cart will have productID, number of items per item, cost of the cart. Customers can add or delete items from the cart
* Customer can place multiple orders and each order contains a unique orderID, order status, order cost, Payment Info, order date, invoice.
* Customer can view order history
* Shipping Company handles shipment of the orders. Each Shipping Company has companyID and company name
* Delivery of each Order can be identified by orderID and Shipment CompanyID. Alternatively, Shipment of the order can also be tracked by unique ShipmentID.  Shipment details are shipment status, delivery date, dispatch date, shipping address
* Customer can post his/her review on products and provide ratings. One customer can review multiple products. However, customer cannot have multiple reviews per product
* Sellers can add items from the listings.  Seller details like sellerID, seller name, description, address and rating are stored in the database
* Products are stored and shipped from warehouses which are present at different locations and have product inventory details (Quantity for each product)
* The same product can be sold by multiple sellers and bought by multiple buyers
* Each customer can create one wishlist which contains product details that they wish to buy
* Products are added to the wishlist

**ER DIAGRAM:**

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties. By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases.

Assumptions :

* A payment mode can be the account number for net banking, card number for payment by card and 9999 for cash on delivery. By default, this option will be set to cash on delivery
* One Customer can have one CartID only
* A customer may or may not give a review of a product
* A single customer can give one review per product
* ShippingID is unique per OrderID and it can also be used to track the shipment
* A customer can have only one Wishlist. Can add multiple items to the wishlist.

**ER DIAGRAM**

**Diagram

Description automatically generated**

**MAPPING THE ER DIAGRAM TO RELATIONAL SCHEMA:**

**Diagram

Description automatically generated with low confidence**

**DATABASE NORMALIZATION RULES:**

* Normalization - is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies
* Normal Form - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
* The mapped relational schema violates two instances of the 1st Normal Form

1st Normal Form Violations:

* The 1st Normal Form does not allow composite attributes, multivalued attributes and nested relations
* 'Address' attribute from the CUSTOMER relation is a multivalued attribute.
* To normalize this, a new relation, CUSTOMER\_ADDRESS is created for the multi-valued attribute ('Address') along with CustomerID (primary key of the CUSTOMER relation)
* The new relation has a composite primary key of CustomerID and Address .
* 'Payment\_Mode' attribute from the CUSTOMER table is a multivalued attribute.
* To normalize this, a new relation, CUSTOMER\_PAYMENT is created for the multi-valued attribute ('Payment\_Mode') along with CustomerID (primary key of the CUSTOMER relation)
* This brings all the relations to the first Normal Form as no further violations are present

2nd Normal Form Violations:

* The 2nd Normal Form does not allow a non-key attribute to be functionally determined by a part of the primary key
* Upon observation, no relations having multiple prime attributes exist where the non-prime attribute only depend on a part of the primary key
* This means all the relations are in 2nd Normal Form

3rd Normal Form Violations:

* The 3rd Normal Form does not allow a non-key attribute to be functionally determined by another non-key attribute (there should not be a transitive dependency)
* In none of the relations, a non-key attribute can be determined by another non-key attribute

**RELATIONAL DIAGRAM AFTER NORMALIZATION:Graphical user interface

Description automatically generated with low confidence**

**CREATING THE TABLES:**

CREATE TABLE WISHLIST (

wishlistid char(9),

wishlistname varchar(25),

customerid char(9) not null,

primary key (wishlistid)

);

CREATE TABLE WISHLIST\_PRODUCTS(

wishlistid char(9),

productid char(9),

primary key (wishlistid,productid)

);

CREATE TABLE SELLER(

sellerid char(9),

sellername varchar(25) not null,

selleraddr varchar(250),

descrption varchar(250),

rating char(1),

primary key (sellerid)

);

CREATE TABLE PROD\_SALE(

sellerid char(9),

productid char(9),

primary key (sellerid,productid)

);

CREATE TABLE ORDERS(

orderid char(9),

order\_status varchar(25),

order\_cost double,

invoice varchar(250),

payment\_info char(250),

customerid char(9) not null,

cartid char(9),

primary key (orderid)

);

CREATE TABLE DELIVERY(

orderid char(9),

companyid char(9),

shippingid char(9) not null,

shipping\_addr varchar(250),

shipping\_date date,

dispatch\_date date,

primary key (orderid, companyid),

unique(shippingid)

);

CREATE TABLE CART (

cartid char(9),

cost double,

customerid char(9) not null,

primary key (cartid),

unique (customerid)

);

CREATE TABLE SHIPPING\_COMPANY (

companyid char(9),

company\_name varchar(25) not null,

primary key (companyid)

);

CREATE TABLE WAREHOUSE (

warehouseid char(9),

warehouse\_name varchar(25) not null,

warehouse\_location varchar(250),

primary key (warehouseid)

);

CREATE TABLE PROD\_CART (

cartid char(9),

productid char(9),

per\_item\_quantity integer,

primary key (cartid,productid)

);

CREATE TABLE PROD\_WAREHOUSE (

warehouseid char(9),

productid char(9),

number\_of\_products integer,

primary key (warehouseid,productid)

);

CREATE TABLE CUSTOMER (

customerid char(9),

customer\_name varchar(50) not null,

phone\_number numeric(10),

email varchar(50),

sex varchar(20),

age integer,

primary key (customerid)

);

CREATE TABLE CUSTOMER\_ADDRESS (

customerid char(9),

address varchar(50),

primary key (customerid,address)

);

CREATE TABLE CUSTOMER\_PAYMENT (

customerid char(9),

payment\_mode varchar(50) default 9999,

primary key (customerid,payment\_mode)

);

CREATE TABLE PRODUCT (

productid char(9),

product\_name varchar(50) not null,

price double not null,

customer\_q\_a varchar(200),

category varchar(50),

product\_description varchar(100),

quantity integer,

primary key (productid)

);

CREATE TABLE REVIEW (

reviewid char(9),

rating float,

productid char(9) not null,

customerid char(9) not null,

primary key (reviewid)

);

**TRIGGERED ACTIONS ON FOREIGN KEYS:**

ALTER TABLE prod\_warehouse ADD CONSTRAINT pwwid FOREIGN KEY(warehouseid) REFERENCES warehouse(warehouseid) ON DELETE CASCADE;

ALTER TABLE prod\_cart ADD CONSTRAINT pcid FOREIGN KEY(cartid) REFERENCES cart(cartid) ON DELETE CASCADE;

ALTER TABLE prod\_cart ADD CONSTRAINT pcpid FOREIGN KEY(productid) REFERENCES product(productid) ON DELETE CASCADE;

ALTER TABLE prod\_warehouse ADD CONSTRAINT pwpid FOREIGN KEY(productid) REFERENCES product(productid) ON DELETE CASCADE;

ALTER TABLE cart ADD CONSTRAINT ccustid FOREIGN KEY(customerid) REFERENCES customer(customerid) ON DELETE CASCADE;

ALTER TABLE orders ADD CONSTRAINT ocustid FOREIGN KEY(customerid) REFERENCES customer(customerid) ON DELETE CASCADE;

ALTER TABLE orders ADD CONSTRAINT ocid FOREIGN KEY(cartid) REFERENCES cart(cartid) ON DELETE CASCADE;

ALTER TABLE delivery ADD CONSTRAINT doid FOREIGN KEY(orderid) REFERENCES orders(orderid) ON DELETE CASCADE;

ALTER TABLE delivery ADD CONSTRAINT dcompid FOREIGN KEY(companyid) REFERENCES shipping\_company(companyid) ON DELETE CASCADE;

ALTER TABLE customer\_address ADD CONSTRAINT cacustid FOREIGN KEY(customerid) REFERENCES customer(customerid) ON DELETE CASCADE;

ALTER TABLE customer\_payment ADD CONSTRAINT cpcustid FOREIGN KEY(customerid) REFERENCES customer(customerid) ON DELETE CASCADE;

ALTER TABLE wishlist ADD CONSTRAINT wcustid FOREIGN KEY(customerid) REFERENCES customer(customerid) ON DELETE CASCADE;

ALTER TABLE wishlist\_products ADD CONSTRAINT wpwid FOREIGN KEY(wishlistid) REFERENCES wishlist(wishlistid) ON DELETE CASCADE;

ALTER TABLE wishlist\_products ADD CONSTRAINT wppid FOREIGN KEY(productid) REFERENCES product(productid) ON DELETE CASCADE;

ALTER TABLE review ADD CONSTRAINT rpid FOREIGN KEY(productid) REFERENCES product(productid) ON DELETE CASCADE;

ALTER TABLE review ADD CONSTRAINT rcustid FOREIGN KEY(customerid) REFERENCES customer(customerid) ON DELETE CASCADE;

ALTER TABLE prod\_sale ADD CONSTRAINT pspid FOREIGN KEY(productid) REFERENCES product(productid) ON DELETE CASCADE;

ALTER TABLE prod\_sale ADD CONSTRAINT pssid FOREIGN KEY(sellerid) REFERENCES seller(sellerid) ON DELETE CASCADE;

**TRIGGERS AND STORED PROCEDURES:**

**TRIGGER #1:**

Update Cost on CART table after UPDATE (of Per\_Item\_Quanity) on PROD\_CART table.

CREATE TRIGGER cart\_cost

AFTER UPDATE OF per\_item\_quanity, productid ON prod\_cart

FOR EACH ROW

DECLARE

qty\_diff INT;

product\_price INT;

BEGIN

/\* assume that Per\_Item\_Quanity and productid are non-null fields \*/

qty\_diff := :NEW.per\_item\_quanity - :OLD.per\_item\_quanity;

IF ( UPDATING

AND :old.productid = :new.productid

AND :old.per\_item\_quanity != :new.per\_item\_quanity ) THEN

SELECT price

INTO product\_price

FROM product

WHERE productid = NEW.productid;

UPDATE CART

SET cost = cost + ( qty\_diff \* product\_price )

WHERE cartid = :new.cartid;

END IF;

END;

**TRIGGER #2:**

Trigger to notify when wishlist item is in stock i.e., the item quantity is updated to a value greater than zero

CREATE OR REPLACE TRIGGER wishlist\_item\_available

AFTER UPDATE OF quantity

ON product FOR EACH ROW

DECLARE wishlistid\_value NUMBER;

wishlistname VARCHAR(25);

BEGIN

IF (:Old.quantity = 0

AND

:NEW.quantity > 0)

THEN SELECT w.wishlistid

INTO wishlistid\_value

FROM wishlist w,

wishlist\_products wp,

product p

WHERE wp.productid = p.productid

AND wp.wishlistid = w.wishlistid;

dbms\_output.put\_line('Wishlist : '

|| :wishlistId\_value

|| ' is in Stock');

END IF;

END;

**STORED PROCEDURE #1:**

Procedure to change the selling price of the product.

CREATE OR REPLACE PROCEDURE Update\_price (productid IN PRODUCT.productid%TYPE, newprice INT) AS

thisproduct PRODUCT%ROWTYPE;

CURSOR prod\_price IS SELECT \* FROM PRODUCT;

BEGIN

OPEN prod\_price;

LOOP

FETCH prod\_price INTO thisproduct;

EXIT WHEN (prod\_price%NOTFOUND);

UPDATE PRODUCT SET price=newprice WHERE productid=productid; dbms\_output.Put\_line (' Price has been changed');

END LOOP;

CLOSE prod\_price;

END;

**STORED PROCEDURE #2:**

Procedure to view the order history of a particular customer

CREATE OR REPLACE PROCEDURE Order\_history(cid IN ORDERS.customerid%TYPE) AS

min\_order float;

max\_order float;

thisOrder ORDERS%ROWTYPE;

CURSOR order\_details IS

SELECT \* FROM ORDERS WHERE cid=customerid;

BEGIN

select min(Order\_cost), max(Order\_cost) into min\_order, max\_order from ORDERS;

OPEN order\_details;

dbms\_output.put\_line( 'Here are the details of the orders the customer ' || cid || ' has ordered : ');

LOOP

FETCH order\_details INTO thisOrder;

EXIT WHEN (order\_details%NOTFOUND);

dbms\_output.put\_line( ' OrderID : ' || thisOrder.OrderID);

dbms\_output.put\_line( ' Total Cost : ' || thisOrder.Order\_cost);

dbms\_output.put\_line( ' Order Date : ' || thisOrder.Order\_date);

dbms\_output.put\_line( ' Invoice : ' || thisOrder.Invoice);

dbms\_output.put\_line( ' Payment Details : ' || thisOrder.Payment\_info);

END LOOP;

CLOSE order\_details;

dbms\_output.put\_line( ' Most expensive order' || max\_order);

dbms\_output.put\_line( ' Most inexpensive order' || min\_order);

END;