DBMS PROJECT REPORT

PES UNIVERSITY

DATABASE MANAGEMENT SYSTEMS

UE18CS252

SUBMITTED BY

PES2201800065 Rishab Kashyap

The following DBMS project - INVENTORY MANAGEMENT SYSTEM - is a simple representation of the inventory maintained in a grocery store. When people buy things from the store, the inventory loses items, and must hence be replenished by the storages or warehouses to the respective stores. If the storage units do not contain elements then they will place an order for the respective item and the shipment is delivered to the storage units for distribution. A certain group of people, or the Admins, handle majority of these transactions. They can create queries to find who is responsible for which storage and in what location, the orders placed by other admins, and many more. The database handles situations of invalid dates using triggers too, by checking if it crosses the acceptable limit such as the order date being after delivery date. This DBMS is capable of providing stores with a basic ordered format of representing the data, such as items they use, and to make transactions a lot more efficient and easy.

TABLE OF CONTENTS

Introduction

The following DBMS is the basic representation of an Inventory Management System that is present in a local goods and grocery store. It's basic functionality is to keep track of the items it sells, it's inventory of items, the storage units to resupply it and ordering new items whenever necessary, thus providing smooth functioning to the store sales and management tracking.

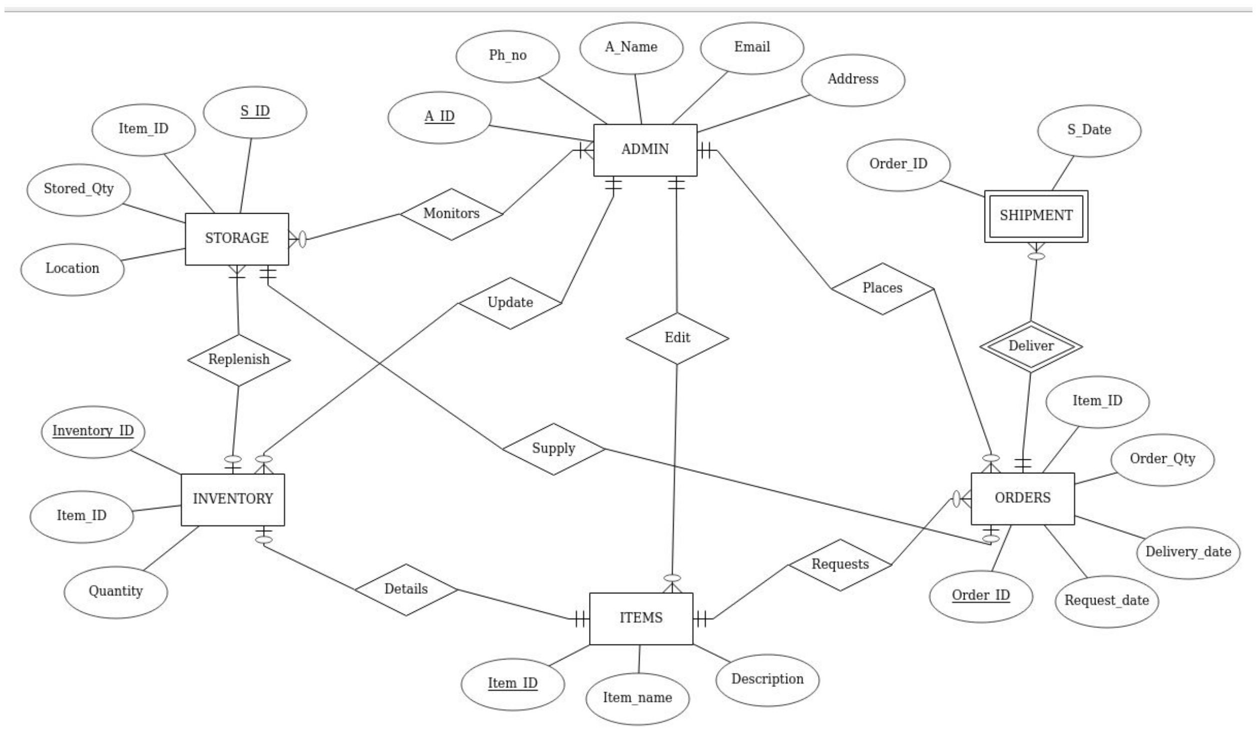
Mini-World Description

In this system, we have a total of 6 major entities that play a crucial role in the functioning of the DBMS. They are given as follows:

1. **ADMIN:** The administrators are the people who take care of all the transactions and functionalities of the DBMS. They are responsible for scrutinizing every detail, and keeping a track of the inventory system as a whole. Every time the products from the store are bought, i.e, reduced in number, the admins take care of changing the values respectively and requesting for orders or replenishments whenever necessary.
2. **INVENTORY:** The heart of the store, it keeps track of the quantity of each item that is present in the store, and it lets the admins know when the products are to be replenished.
3. **ITEMS:** This consists of a list of the list of items that the inventory or storage can hold, i.e, the list of goods that are sold by the store. New items can be added and old items deleted whenever necessary by the admins depending on the sales of the store.
4. **STORAGE:** This entity represents the warehouses where the items are stored for future use. Depending upon what the store sells, this replenished the respective goods to the inventory whenever it runs short of items.
5. **ORDERS:** This entity represents the list of items that require to be replenished in the storage itself. Whenever there is a shortage of a certain item, the admin places an order for that particular item and waits for the shipment to arrive.
6. **SHIPMENT:** The final entity, it represents the list of goods that are presently being shipped, i.e, which have been requested by the admins but have not yet been delivered. Once the goods are delivered, they are sent to the storages and the entire cycle repeats again.

DATA MODEL

ER-Diagram



ER diagram for Inventory Management

The above diagram shows the basic model of all the entities present in the DBMS along with their respective attributes. Each one of them is explained below:

- **<u>ADMIN:</u>** As described, the admin takes care of the overall functionality and is defined by five attributes :
    - Admin Identity (A_ID), which is the primary key
    - Admin name (A_Name)
    - Phone number (Ph_no)

    - Email and their address.
- **STORAGE:** It is defined by four main entities:
    - Storage ID (S_ID) which is the primary key to the entity
    - The quantity it stores (Stored_Qty)
    - The Item identification to identify the specific items (Item_ID) which is a foreign key, and
    - The Location of each storage unit.
- **INVENTORY:** It is defined by three attributes :
    - Invetory_ID which is the primary key to the entity
    - The item _ID as defined for the storage, and
    - The Quantity stored by it.
- **ITEMS:** It consists of
    - The Item_ID, which is the primary key, and a very important one as we shall see in the Schema
    - The item name (item_name) and
    - The Description of each item.
- **ORDERS:** This entity is used to represent the list of placed orders and consists of
    - The Order_ID, which is the primary key
    - The Request_date for the item
    - The Ordered_Qty to show how much to order
    - The delivery_date of the item, and
    - The item_ID as the foreign key.
- **SHIPMENTS:** This is a special entity that does not have a primary key associated with it. It is hence a weak entity which consists of:
    - The Order_ID as the foreign key and
    - The shipment date S_Date for when the shipment has been dispatched.

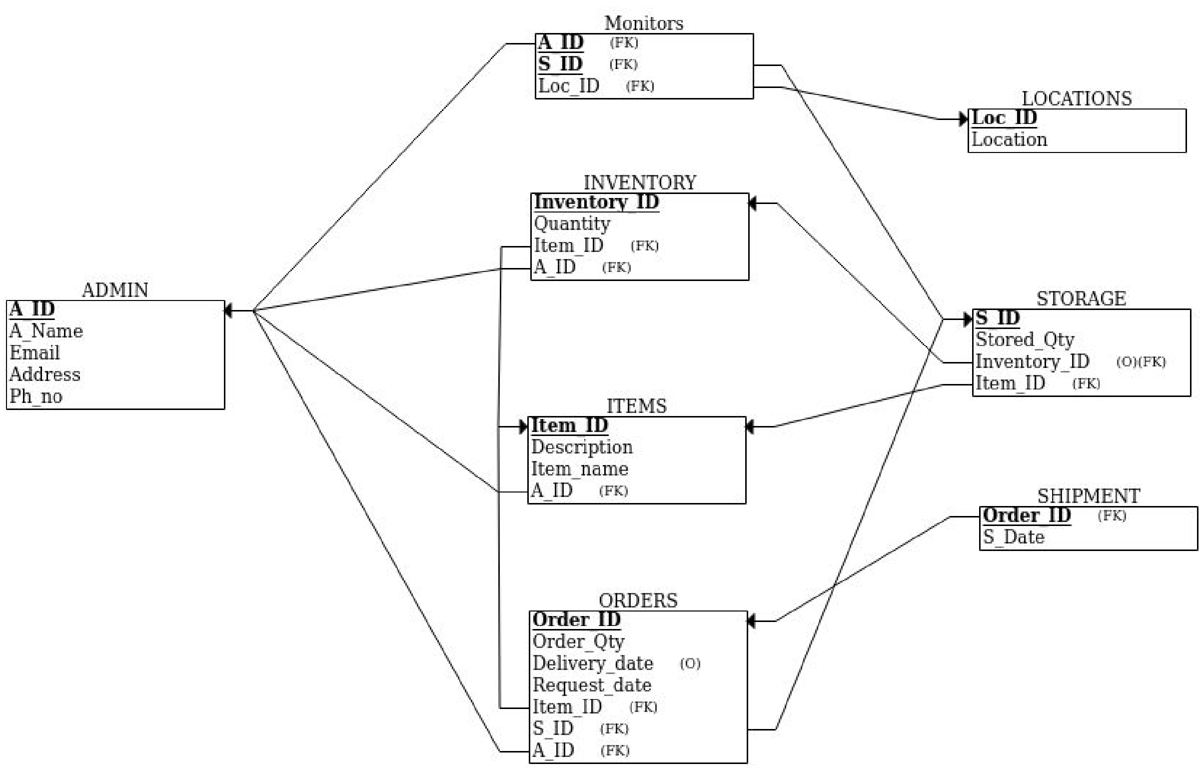
The entities thus share certain relations amongst them and can be defined as follows:

*Admin -> **MONITORS ->** the Storage with an M:N cardinality ratio*
*Admin -> **UPDATES ->** the Inventory with a 1:N ratio*
*Admin -> **EDITS** -> the Items list with a 1:N ratio*
*Admin -> **PLACES** -> Orders with a 1:N ratio*
*Storage -> **REPLENISHES** -> the Inventory with an N:1 ratio*
*Items provides -> **DETAILS** -> about each inventory item with a 1:1 ratio*
*Items send -> **REQUESTS** -> to Orders for new items with a 1:N ratio*
*Orders -> **SUPPLY** -> the Storage with items with a 1:1 ratio*

*Shipments -> **DELIVER ->** based on Orders with an N:1 ratio*

The last relation "Delivers" is defined to be an ***Identifying Relation*** as it connects the Orders strong entity to a weak entity Shipment.

Relational Schema Diagram



- The relational schema as shown above describes the ER diagram in a more suitable way to represent them in the form of tables based on their attributes.

- As we have seen in the previous section, the primary keys and foreign keys are mapped together in this schema in a way such that it is easy to distinguish between the use of the attributes and remove ambiguities that arise due to lack of definition from an ER diagram.
- We notice that there are certain differences in the schema compared to the diagram, such as the split of the storage entity into three parts in the schema.
- This is done so as to aid in the normalization process as we shall see in the next section.
- Now, the different types of keys represented here are :
  - PK - Primary Key - Highlighted in bold in the Schema given
  - FK - Foreign Key - Marked in brackets beside an attribute
  - (O) - Optional - these attributes can hold null values in them.
- The foreign keys help in establishing the relations that we have defined for the ER diagrams between all the entities in the schema. It makes mapping of values between interrelated entities a lot easier.
- For example, it is easier to identify the Inventory number in the storage entity so as to allow the storage to place the correct item in that specific ID slot with respect to the Item_ID defined as the foreign key reference in both the entities.
- The data types used to define the above attributes include:
  - INT - Integer type value, cannot represent anything but integer numbers
  - NUMERIC - Similar to integer type but has a defined length to it
  - CHAR - Only letters are allowed under this data type
  - VARCHAR - special type of char which includes special symbols and numbers in string format.
  - DATE - Represents the date on which any delivery or such was made.
- The tables Monitor and LOCATIONS are the child tables derived from storage. These will be explained during the normalization process in the next section.

FD AND NORMALIZATION

Functional Dependencies

A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1 [X] = t2 [X], they must also have t1 [Y] = t2 [Y].
This means that the value of a Y component is determined by the X component of a Tuple uniquely or functionally. This means that Y is Functionally Dependent on the value of X. Here, X is called the LHS attribute, and Y is called the RHS attribute.

The functional dependencies in the inventory schema can be shown as follows:

- **ADMIN:** *A_ID -> A_Name, Email, Address, Ph_no*
- **Monitors:** *{A_ID, S_ID} -> Loc_ID*
- **LOCATIONS:** *L_ID -> Location*
- **INVENTORY:** *Inventory_ID -> Quantity*
- **ITEMS:** *Item_ID -> Item_name, Description*
- **ORDERS:** *Order_ID -> Delivery_date, Request_date ;*
  {Order_ID, Item_ID} -> Order_Qty
- **STORAGE:** *S_ID -> Stored_Qty;*
- **SHIPMENT:** *Order_ID -> S_Date*

NORMAL FORMS

The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The nonadditive join or lossless join property, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
- The dependency preservation property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

The nonadditive join property is extremely critical and must be achieved at any cost, whereas the dependency preservation property, although desirable, is sometimes sacrificed.

FIRST NORMAL FORM

It is defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

In the Inventory Schema, we can see that prior to its creation, “location” was defined as a single attribute in the ER diagram. Now this could have been a multi-valued attribute, considering that the same item could be available in two or more locations too. To prevent this from happening, we separated all of them and wrote them individually, even if it looked almost like repetition of values, which is thus taken care of by splitting location into another table called “Monitors” which holds the location separatelyWith a defined Loc_ID. The other entities are all in 1NF.

SECOND NORMAL FORM

Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency $X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore; that is, for any attribute $A \varepsilon X$, $(X - \{A\})$ does not functionally determine Y. A functional dependency $X \rightarrow Y$ is a partial dependency if some attribute $A \varepsilon X$ can be removed from X and the dependency still holds; that is, for some $A \varepsilon X$,

$(X - \{A\}) \rightarrow Y$. Hence we say that A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

None of the entities in the Inventory schema violate the second normal form.

THIRD NORMAL FORM

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. Hence, according to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

In the Inventory schema, we had earlier split the Location from the Storage table and placed it with Loc_ID in the Monitors table. Now, we see an obvious transitive dependency, since Loc_ID is functionally dependent on both A_ID and S_ID, while location is dependent only upon Loc_ID. Hence, we remove the Location and place it in a separate Locations table with the Loc_ID as well as the location name and thus prevent violation of 3NF form in the schema.

The Inventory schema satisfies the BCNF also, a special case of 3NF.

DATA DEFINITION LANGUAGE

```
CREATE TABLE ADMIN
(
  A_ID NUMERIC(10) NOT NULL,
  A_Name CHAR(20) NOT NULL,
  Email VARCHAR(50) NOT NULL,
  CONSTRAINT chk_email CHECK (Email LIKE '%_@__%.__%'),
  Address VARCHAR(100) NOT NULL,
  Ph_no NUMERIC(10) NOT NULL,
  PRIMARY KEY (A_ID)
);

CREATE TABLE ITEMS
(
  Item_ID NUMERIC(3) NOT NULL,
  Item_name CHAR(20) NOT NULL,
  Description VARCHAR(50) NOT NULL,
  A_ID NUMERIC(10) NOT NULL,
  PRIMARY KEY (Item_ID),
  CONSTRAINT fk_A_ID
      FOREIGN KEY (A_ID) REFERENCES ADMIN(A_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE INVENTORY
(
  Inventory_ID NUMERIC(4) NOT NULL,
  Item_ID NUMERIC(3) NOT NULL,
  Quantity INT NOT NULL CHECK(Quantity >= 0),
  A_ID NUMERIC(10) NOT NULL,
  PRIMARY KEY (Inventory_ID),
  FOREIGN KEY (Item_ID) REFERENCES ITEMS(Item_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (A_ID) REFERENCES ADMIN(A_ID)
```

```
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE STORAGE
(
  S_ID NUMERIC(6) NOT NULL,
  Item_ID NUMERIC(3) NOT NULL,
  Stored_Qty INT NOT NULL CHECK (Stored_Qty >= 0),
  Inventory_ID NUMERIC(4) NOT NULL,
  PRIMARY KEY (S_ID),
  FOREIGN KEY (Inventory_ID) REFERENCES INVENTORY(Inventory_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Item_ID) REFERENCES ITEMS(Item_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE ORDERS
(
  Order_ID NUMERIC(5) NOT NULL,
  Item_ID NUMERIC(3) NOT NULL,
  Order_Qty INT NOT NULL,
  Delivery_date DATE,
  Request_date DATE NOT NULL,
  S_ID NUMERIC(6) NOT NULL,
  A_ID NUMERIC(10) NOT NULL,
  PRIMARY KEY (Order_ID),
  FOREIGN KEY (Item_ID) REFERENCES ITEMS(Item_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (S_ID) REFERENCES STORAGE(S_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (A_ID) REFERENCES ADMIN(A_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
CREATE TABLE SHIPMENT
(
  Order_ID NUMERIC(5) NOT NULL,
  S_Date DATE NOT NULL,
  PRIMARY KEY (Order_ID),
  FOREIGN KEY (Order_ID) REFERENCES ORDERS(Order_ID)
);

CREATE TABLE Monitors
(
  A_ID NUMERIC(10) NOT NULL,
  S_ID NUMERIC(6) NOT NULL,
  Loc_ID NUMERIC(2) NOT NULL,
  PRIMARY KEY (A_ID, S_ID),
  FOREIGN KEY (A_ID) REFERENCES ADMIN(A_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (S_ID) REFERENCES STORAGE(S_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Loc_ID) REFERENCES LOCATIONS(Loc_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);


CREATE TABLE LOCATIONS
(
  Loc_ID NUMERIC(2) NOT NULL,
  Location VARCHAR(10) NOT NULL,
  PRIMARY KEY (Loc_ID)
);


INSERT INTO ADMIN VALUES
(1003165999 , 'Rakesh Gupta' , 'rakeshgupta456@pes.edu' , '#21, Church street, Bangalore',
5834992456),
```

(2007657138 , 'Animeha Shah' , 'animax23@yahoo.com' , 'A-1512, Wuthering heights, RR nagar, Bangalore', 9416819176),
(7001568165 , 'Srinath Reddy' , 'srini4125reddy@gmail.com' , '#31, Gopishankar layout, Bangalore', 6316703468),
(6004328643 , 'Eshaan Patel' , 'eshaanpatelxyz@gmail.com' , '749-21, SNN towers, Akshayanagar, Bangalore', 7654328915),
(5004367537 , 'Sneha Rao' , 'sneharao@pes.edu' , '#378, Bilekahalli, Bangalore', 8456329764);

INSERT INTO ITEMS VALUES
(321 , 'Apples' , 'Fresh seasonal Fuji apples', 2007657138),
(567 , 'Oranges' , 'Large size oranges for fresh juices' , 7001568165),
(946 , 'Watermelons' , 'Fresh large watermelons', 7001568165),
(726 , 'CerealX' , 'Popular cereal brand, stock up frequently', 1003165999),
(675 , 'Bread' , 'Freshly baked bread for sandwiches', 2007657138),
(834 , 'Milk' , 'Milk cartons, best quality', 5004367537),
(116 , 'Pickle' , 'Five varieties, popular brand', 6004328643);

INSERT INTO INVENTORY VALUES
(6425 , 567 , 50 , 7001568165),
(8846 , 946 , 30 , 7001568165),
(5763 , 726 , 20 , 1003165999),
(1429 , 834 , 10 , 5004367537);

INSERT INTO STORAGE VALUES
(176583 , 567 , 300 , 6425),
(240359 , 321 , 500 , null),
(814657 , 946 , 200 , 8846),
(648234 , 726 , 100 , 5763),
(346529 , 726 , 200 , 5763),
(943578 , 834 , 300 , 1429),
(653248 , 675 , 400 , null),
(556732 , 116 , 150 , null);

INSERT INTO ORDERS VALUES
(12345 , 116 , 150 , '2020-02-27' , '2020-02-13' , 556732, 6004328643),
(23456 , 946 , 100 , '2020-03-20' , '2020-03-17' , 814657, 7001568165),

(44653 , 675 , 200 , '2020-03-22' , '2020-03-17' , 653248, 2007657138),
(72653 , 726 , 100 , null , '2020-04-03' , 346529, 1003165999),
(95762 , 321 , 250 , null , '2020-04-03' , 240359 , 2007657138);

INSERT INTO SHIPMENT VALUES
(72653 , '2020-04-07'),
(95762 , '2020-04-05');

INSERT INTO LOCATIONS VALUES
(01 , 'Hassan'),
(02 , 'Mangalore'),
(03 , 'Mysore'),
(04 , 'Bangalore');

INSERT INTO Monitors VALUES
(7001568165 , 176583 , 01),
(7001568165 , 814657 , 04),
(2007657138 , 240359 , 03),
(1003165999 , 648234 , 02),
(1003165999 , 346529 , 04),
(5004367537 , 943578 , 01),
(2007657138 , 653248 , 04),
(6004328643 , 556732 , 04);

CONSTRAINTS:

CHECK:

The check constraints in this include:

- CONSTRAINT chk_email CHECK (Email LIKE '%_@__%.__%')
    - This makes sure that the user types in the email as per the given structure.
- Constraint CHECK (quantity > x) where x is a value given above.
    - This makes sure that the value of quantity in any of the tables is greater than x.

INTEGRITY:

- Cascade Functions such as ON DELETE CASCADE and ON UPDATE CASCADE are used to simultaneously make changes to those tables which have the foreign key constraint defined with them

TRIGGERS:

Triggers are special functions that can be used to monitor the database whenever any changes are made to it. The examples used in the Inventory Schema are as follows:

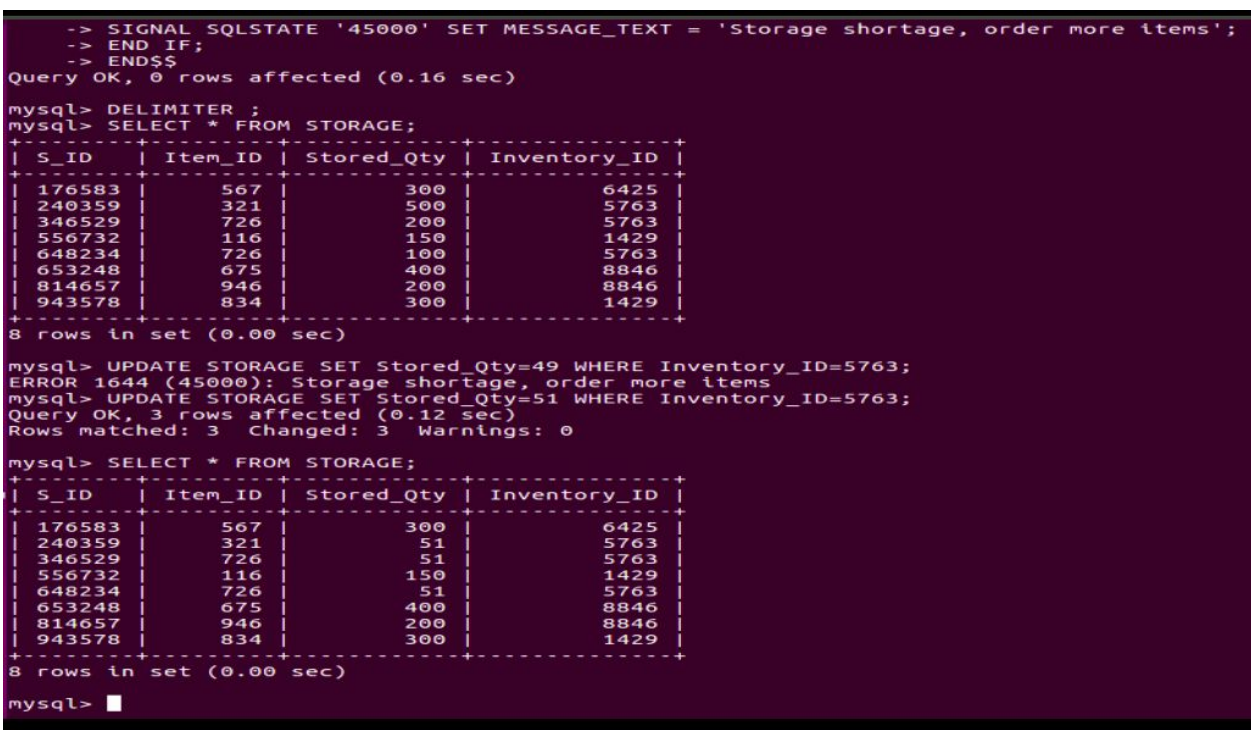
1)Trigger to check when quantity in storage is too low

```
DELIMITER $$
CREATE TRIGGER storage_violation
AFTER UPDATE ON STORAGE
FOR EACH ROW BEGIN
IF(NEW.Stored_Qty < 50) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Storage shortage, order more items';
END IF;
END$$
DELIMITER ;
```

2)Trigger to Check for delivery date validity

```
DELIMITER $$
CREATE TRIGGER date_violation
BEFORE UPDATE ON ORDERS
FOR EACH ROW BEGIN
IF ((NEW.Delivery_date < CURDATE()) OR (NEW.Delivery_date < Request_date)) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid date entered';
END IF;
END $$
DELIMITER;
```

3)Trigger to Check for request date validity:
DELIMITER $
CREATE TRIGGER exceed_date
BEFORE INSERT ON ORDERS
FOR EACH ROW BEGIN
IF NEW.Request_date < CURDATE() THEN
SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Invalid date entered';
END IF;
END $$
DELIMITER;



```
    -> SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Storage shortage, order more items';
    -> END IF;
    -> END$$
Query OK, 0 rows affected (0.16 sec)

mysql> DELIMITER ;
mysql> SELECT * FROM STORAGE;
+--------+---------+------------+--------------+
| S_ID   | Item_ID | Stored_Qty | Inventory_ID |
+--------+---------+------------+--------------+
| 176583 |     567 |        300 |         6425 |
| 240359 |     321 |        500 |         5763 |
| 346529 |     726 |        200 |         5763 |
| 556732 |     116 |        150 |         1429 |
| 648234 |     726 |        100 |         5763 |
| 653248 |     675 |        400 |         8846 |
| 814657 |     946 |        200 |         8846 |
| 943578 |     834 |        300 |         1429 |
+--------+---------+------------+--------------+
8 rows in set (0.00 sec)

mysql> UPDATE STORAGE SET Stored_Qty=49 WHERE Inventory_ID=5763;
ERROR 1644 (45000): Storage shortage, order more items
mysql> UPDATE STORAGE SET Stored_Qty=51 WHERE Inventory_ID=5763;
Query OK, 3 rows affected (0.12 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM STORAGE;
+--------+---------+------------+--------------+
| S_ID   | Item_ID | Stored_Qty | Inventory_ID |
+--------+---------+------------+--------------+
| 176583 |     567 |        300 |         6425 |
| 240359 |     321 |         51 |         5763 |
| 346529 |     726 |         51 |         5763 |
| 556732 |     116 |        150 |         1429 |
| 648234 |     726 |         51 |         5763 |
| 653248 |     675 |        400 |         8846 |
| 814657 |     946 |        200 |         8846 |
| 943578 |     834 |        300 |         1429 |
+--------+---------+------------+--------------+
8 rows in set (0.00 sec)

mysql> 
```

Example execution for the first trigger

SQL QUERIES:

OUTER JOINS

1) Simple query to check all request dates of respective items being shipped

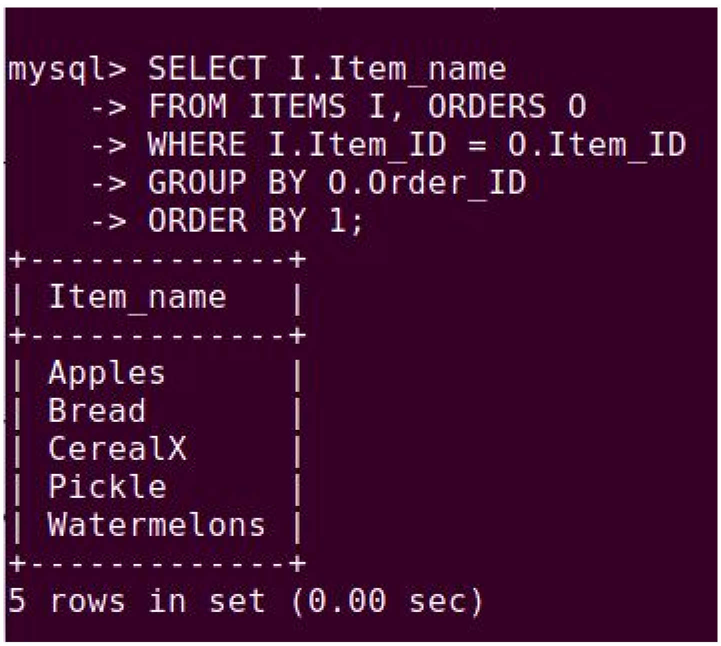
SELECT O.Request_date, I.Item_name
FROM ITEMS I,
ORDERS O RIGHT OUTER JOIN SHIPMENT AS H
ON O.Order_ID = H.Order_ID
WHERE I.Item_ID = O.Item_ID;

```
mysql>
mysql> SELECT O.Request_date, I.Item_name
    -> FROM ITEMS I,
    -> ORDERS O RIGHT OUTER JOIN SHIPMENT AS H
    -> ON O.Order_ID = H.Order_ID
    -> WHERE I.Item_ID = O.Item_ID;
+--------------+-----------+
| Request_date | Item_name |
+--------------+-----------+
| 2020-03-25   | CerealX   |
| 2020-04-03   | Apples    |
+--------------+-----------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM ITEMS
    -> ;
+---------+-------------+--------------------------------------------+------------+
| Item_ID | Item_name   | Description                                | A_ID       |
+---------+-------------+--------------------------------------------+------------+
|     116 | Pickle      | Five varieties, popular brand              | 6004328643 |
|     321 | Apples      | Fresh seasonal Fuji apples                 | 2007657138 |
|     567 | Oranges     | Large size oranges for fresh juices        | 7001568165 |
|     675 | Bread       | Freshly baked bread for sandwitches        | 2007657138 |
|     726 | CerealX     | Popular cereal brand, stock up frequently  | 1003165999 |
|     834 | Milk        | Milk cartons, best quality                 | 5004367537 |
|     946 | Watermelons | Fresh large watermelons                    | 7001568165 |
+---------+-------------+--------------------------------------------+------------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM SHIPMENT;
+----------+------------+
| Order_ID | S_Date     |
+----------+------------+
|    72653 | 2020-04-07 |
|    95762 | 2020-04-05 |
+----------+------------+
2 rows in set (0.00 sec)

mysql>
```

AGGREGATE FUNCTIONS

2) Give the item names of all ordered items in ascending alphabetical order.

SELECT I.Item_name
FROM ITEMS I, ORDERS O
WHERE I.Item_ID = O.Item_ID
GROUP BY O.Order_ID
ORDER BY 1;



3) To show all locations which hold more than 300 units of any item

SELECT L.Location
FROM LOCATIONS L, Monitors M, STORAGE S
WHERE M.Loc_ID = L.Loc_ID AND
M.S_ID = S.S_ID AND

S.Stored_Qty > 300
GROUP BY L.Loc_ID;

```
mysql> SELECT L.Location
    -> FROM LOCATIONS L, Monitors M, STORAGE S
    -> WHERE M.Loc_ID = L.Loc_ID AND
    -> M.S_ID = S.S_ID AND
    -> S.Stored_Qty > 300
    -> GROUP BY L.Loc_ID;
+-----------+
| Location  |
+-----------+
| Mysore    |
| Bangalore |
+-----------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM LOCATIONS;
+--------+-----------+
| Loc_ID | Location  |
+--------+-----------+
|      1 | Hassan    |
|      2 | Mangalore |
|      3 | Mysore    |
|      4 | Bangalore |
+--------+-----------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM STORAGE;
+--------+---------+------------+--------------+
| S_ID   | Item_ID | Stored_Qty | Inventory_ID |
+--------+---------+------------+--------------+
| 176583 |     567 |        300 |         6425 |
| 240359 |     321 |        500 |         NULL |
| 346529 |     726 |        200 |         5763 |
| 556732 |     116 |        150 |         NULL |
| 648234 |     726 |        100 |         5763 |
| 653248 |     675 |        400 |         NULL |
| 814657 |     946 |        200 |         8846 |
| 943578 |     834 |        300 |         1429 |
+--------+---------+------------+--------------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM Monitors;
+------------+--------+--------+
| A_ID       | S_ID   | Loc_ID |
+------------+--------+--------+
| 5004367537 | 943578 |      1 |
| 7001568165 | 176583 |      1 |
| 1003165999 | 648234 |      2 |
| 2007657138 | 240359 |      3 |
| 1003165999 | 346529 |      4 |
| 2007657138 | 653248 |      4 |
| 6004328643 | 556732 |      4 |
| 7001568165 | 814657 |      4 |
```

NESTED CORRELATED QUERIES

4) SQL Query to show which admin is responsible for taking care of which location

```
SELECT A.A_Name, L.Location
FROM ADMIN A, LOCATIONS L
WHERE EXISTS
(SELECT *
 FROM Monitors M
WHERE M.A_ID = A.A_ID AND
            M.Loc_ID = L.Loc_ID);
```

```
mysql> SELECT A.A_Name, L.Location
    -> FROM ADMIN A, LOCATIONS L
    -> WHERE EXISTS (SELECT *
    -> FROM Monitors M
    -> WHERE M.A_ID = A.A_ID AND
    -> M.Loc_ID = L.Loc_ID);
+---------------+-----------+
| A_Name        | Location  |
+---------------+-----------+
| Rakesh Gupta  | Mangalore |
| Rakesh Gupta  | Bangalore |
| Animeha Shah  | Mysore    |
| Animeha Shah  | Bangalore |
| Sneha Rao     | Hassan    |
| Eshaan Patel  | Bangalore |
| Srinath Reddy | Hassan    |
| Srinath Reddy | Bangalore |
+---------------+-----------+
8 rows in set (0.00 sec)

mysql> Select * from admin
    -> ;
ERROR 1146 (42S02): Table 'Inventory.admin' doesn't exist
mysql> select * from ADMIN;
+------------+---------------+--------------------------+---------------------------------------------------+------------+
| A_ID       | A_Name        | Email                    | Address                                           | Ph_no      |
+------------+---------------+--------------------------+---------------------------------------------------+------------+
| 1003165999 | Rakesh Gupta  | rakeshgupta456@pes.edu   | #21, Church street, Bangalore                     | 5834992456 |
| 2007657138 | Animeha Shah  | animax23@yahoo.com       | A-1512, Wuthering heights, RR nagar, Bangalore    | 9416819176 |
| 5004367537 | Sneha Rao     | sneharao@pes.edu         | #378, Bilekahalli, Bangalore                      | 8456329764 |
| 6004328643 | Eshaan Patel  | eshaanpatelxyz@gmail.com | 749-21, SNN towers, Akshayanagar, Bangalore       | 7654328915 |
| 7001568165 | Srinath Reddy | srini4125reddy@gmail.com | #31, Gopishankar layout, Bangalore                | 6316703468 |
+------------+---------------+--------------------------+---------------------------------------------------+------------+
5 rows in set (0.00 sec)

mysql> select * from Monitors
    -> ;
+------------+--------+--------+
| A_ID       | S_ID   | Loc_ID |
+------------+--------+--------+
| 5004367537 | 943578 |      1 |
| 7001568165 | 176583 |      1 |
| 1003165999 | 648234 |      2 |
| 2007657138 | 240359 |      3 |
| 1003165999 | 346529 |      4 |
| 2007657138 | 653248 |      4 |
| 6004328643 | 556732 |      4 |
| 7001568165 | 814657 |      4 |
+------------+--------+--------+
8 rows in set (0.00 sec)
```

5) Select all item names which are not present in the inventory.

SELECT I.Item_name
FROM ITEMS I
WHERE NOT EXISTS(
 SELECT *
 FROM INVENTORY K
WHERE I.Item_ID = K.Item_ID);

```
mysql>
mysql> SELECT I.Item_name
    -> FROM ITEMS I
    -> WHERE NOT EXISTS(
    -> SELECT *
    -> FROM INVENTORY K
    -> WHERE I.Item_ID = K.Item_ID);
+-----------+
| Item_name |
+-----------+
| Pickle    |
| Apples    |
| Bread     |
+-----------+
3 rows in set (0.00 sec)
```

CONCLUSION

CAPABILITIES

- The Inventory DBMS can keep track of multiple transactions simultaneously from ordering and tracking shipments, to checking for storage locations.
- It enables the stores to keep a refined and well maintained efficient system to take care of their product transactions
- It can store multiple item values and is useful to track down important details such as alerting the admin when the item stock is less in the storage.

LIMITATIONS AND IMPROVEMENTS

- It can use a lot more triggers to smoothen out the overall functioning of the system
- Cannot store the shipment orders and bills generated once the order is removed from the table
- Transferring of values has to be done manually from storage to inventory.
- More entities such as transport methods and cost of maintenance can be added to improvise on the existing system.