DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CS6106

DATABASE MANAGEMENT SYSTEMS

PROJECT REPORT

INVENTORY MANAGEMENT SYSTEM (IMS)

-BY

AISHWARYA S (2022103037)

SULOCHANA H (2022103580)

TEAM NO: 24

CSE – 'P' BATCH

2nd YEAR / SEM 4

Date of submission: 22/05/2024

OBJECTIVE:

The primary goal of inventory management is to ensure that all kinds of materials are accessible whenever the production department needs them, ensuring that production is not stopped or slowed down due to a lack of resources. Thus, it is prudent to maintain a buffer stock of all critical goods in order to keep production on track.

It is impossible to fulfil a received order if you do not have an accurate count of the items in your possession. In order to meet requests, you must have access to the appropriate goods at the right time. Otherwise, you may end in a state of confusion. To fulfil the needs for quality products, the concern must maintain an adequate supply of completed items to guarantee that customers' orders are fulfilled. It will increase the company's brand image and hence an effective solution.

PROBLEM STATEMENT:

In the dynamic landscape of small business development, entrepreneurs face a critical challenge; the inefficiency and complexity of traditional inventory management systems. Existing solutions often prove cumbersome, time-consuming, and require a steep learning curve, hindering the growth potential of emerging ventures. The need for a streamlined and user-friendly inventory management system is evident. It empowers entrepreneurs to effortlessly organize, monitor, and optimize their stock, enabling them to focus on what matters most resulting in business growth.

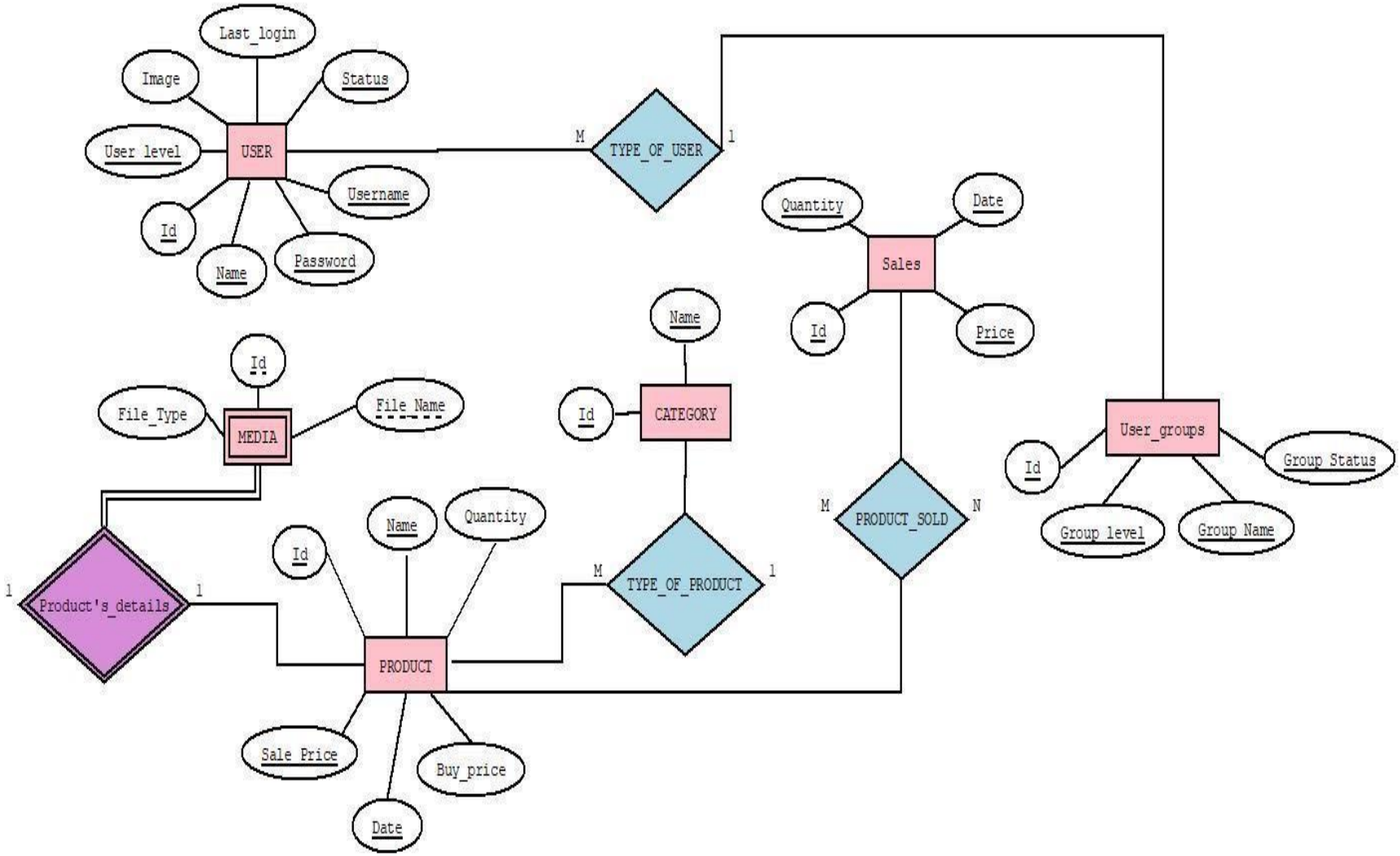
SOLUTION:

Introducing an innovative inventory management system designed for small businesses. Our solution streamlines stock control, minimizes waste, and optimizes inventory levels, empowering entrepreneurs to focus on growth while enhancing customer satisfaction. Imagine a seamless, user-friendly tool that revolutionizes the way small businesses manage their inventory, bringing efficiency and success to the forefront of their operations.

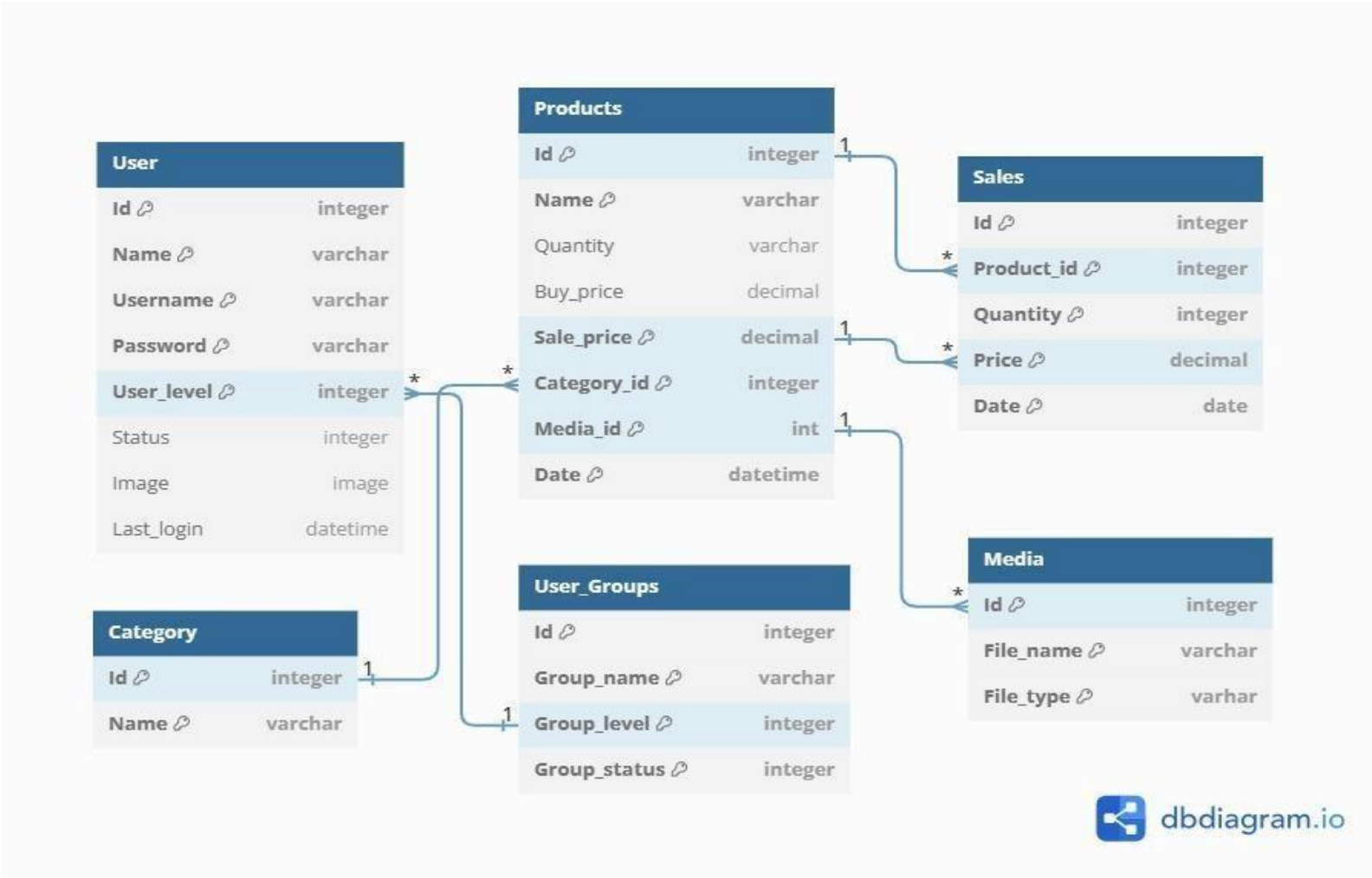
ER - ENTITIES:

- ADMIN/ USER/ SPECIAL
- PRODUCTS
- MEDIA (WEAK ENTITY)
- SALES
- USER_GROUPS
- CATEGORIES

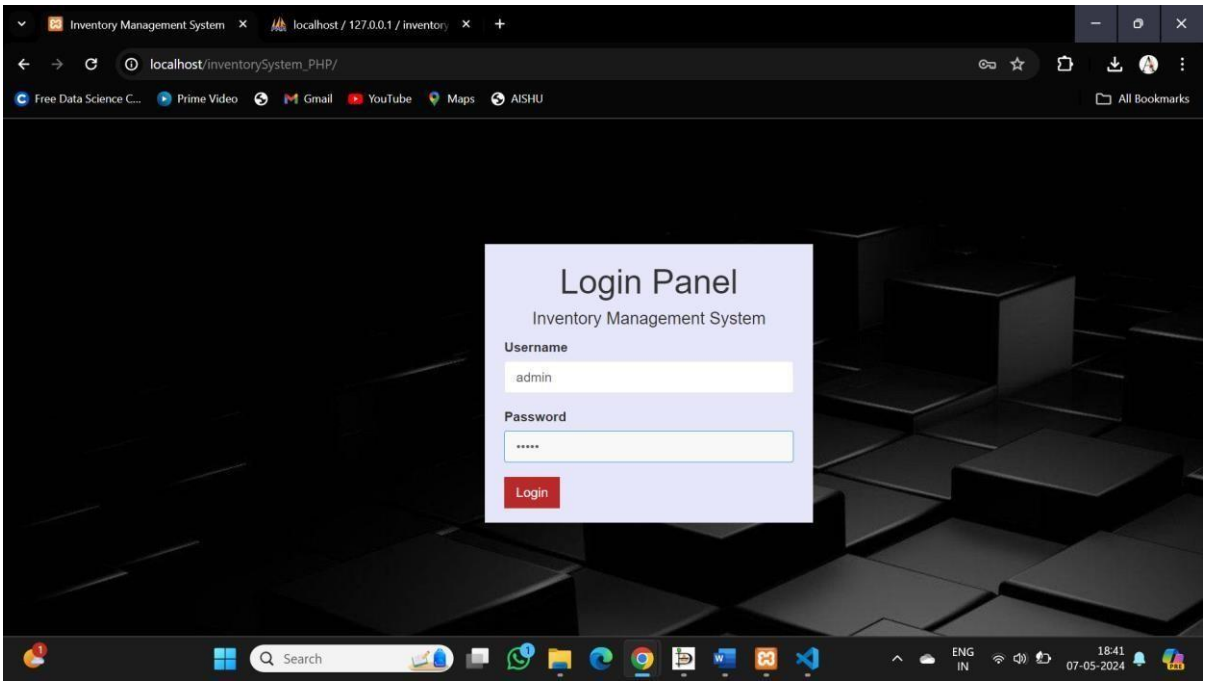
ER MODEL DIAGRAM:



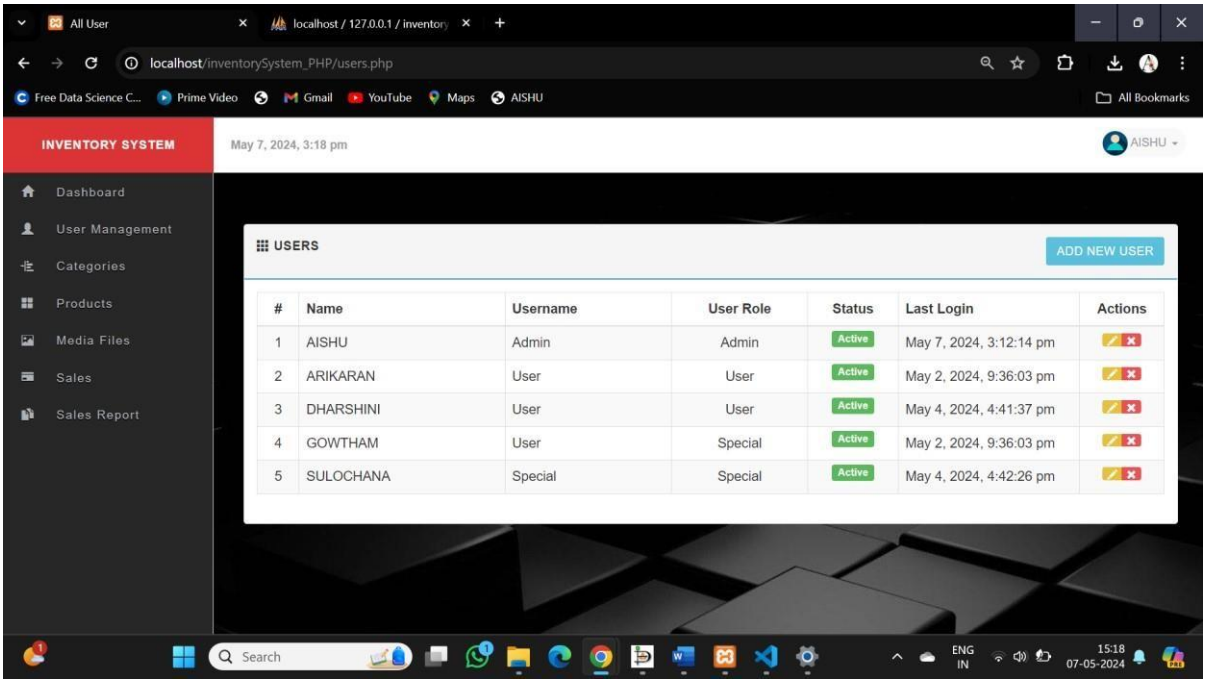
RELATIONAL SCHEMA :



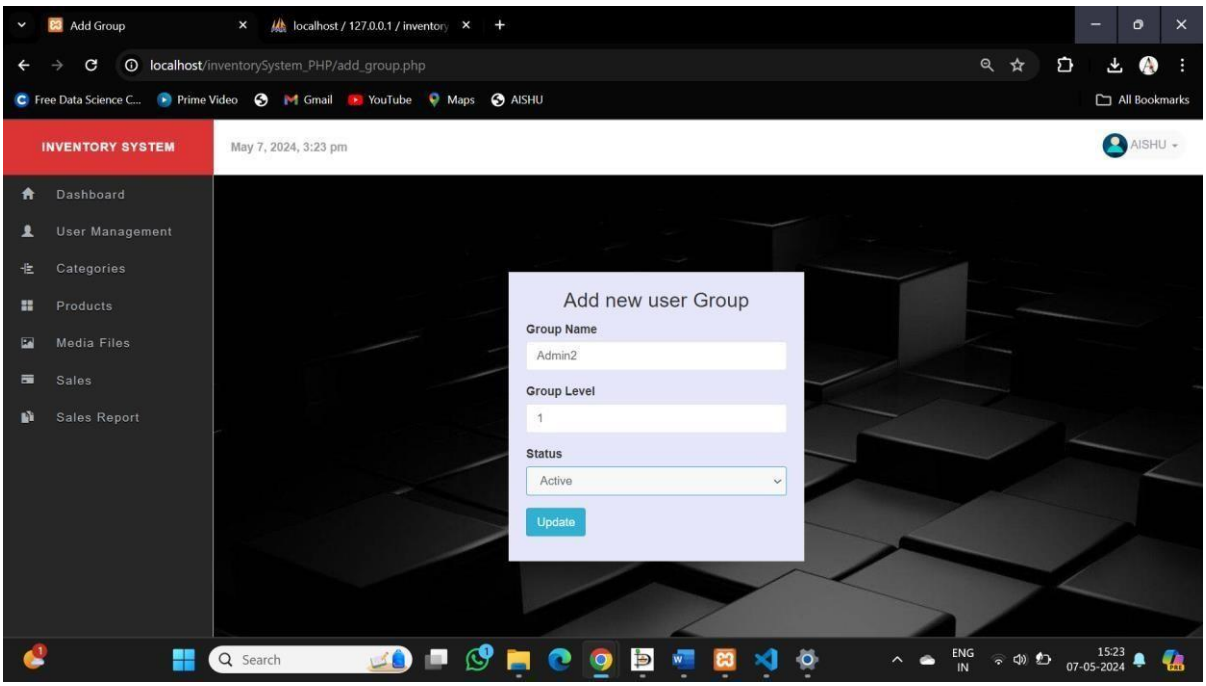
LOGIN PAGE:



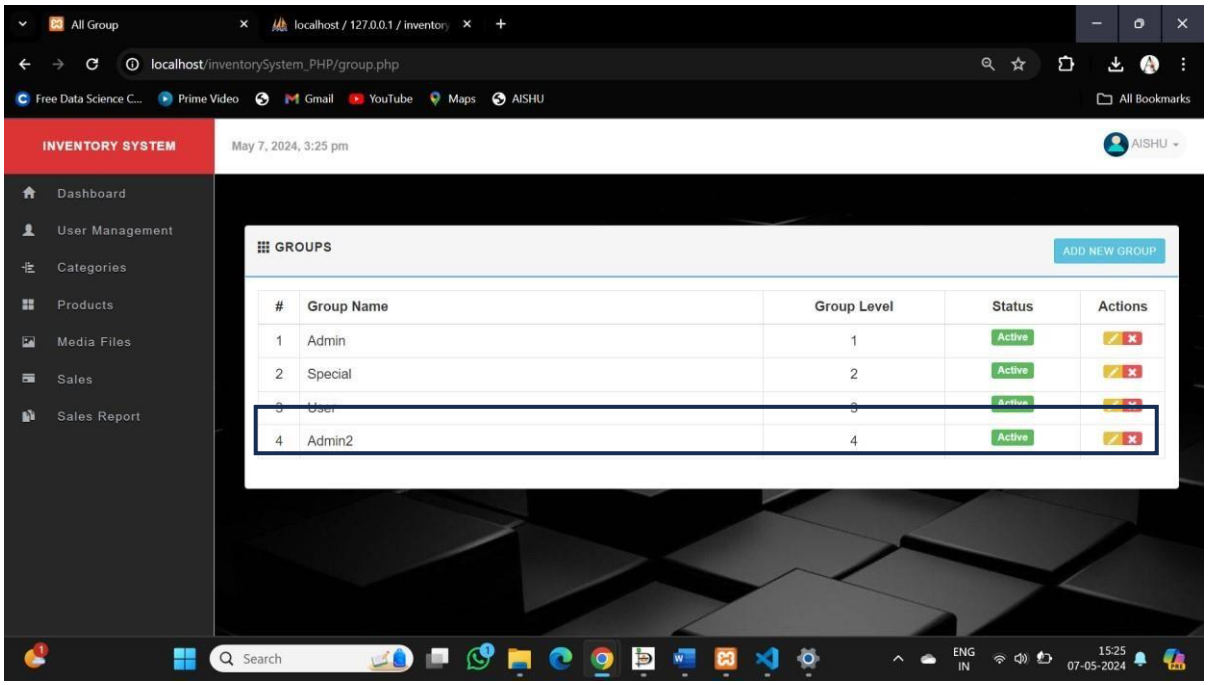
USER (Table):



ADDING NEW GROUPS TO USER_GROUP TABLE:



RECORD INSERTED:



TRIGGERS, FUNCTIONS, PROCEDURES, CURSORS & VIEWS

TRIGGERS:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

CREATING A TRIGGER:

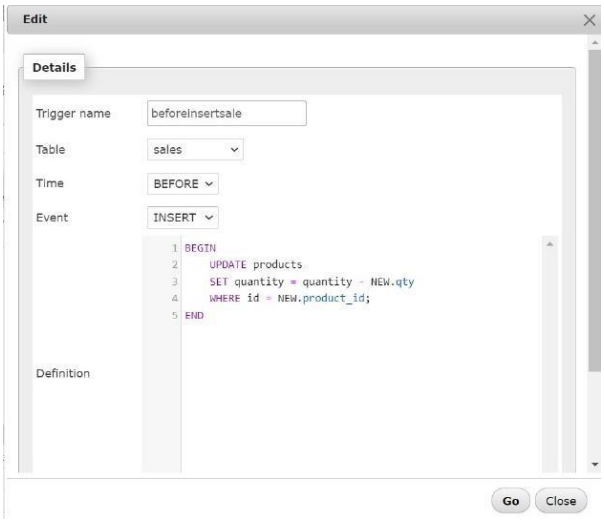
Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

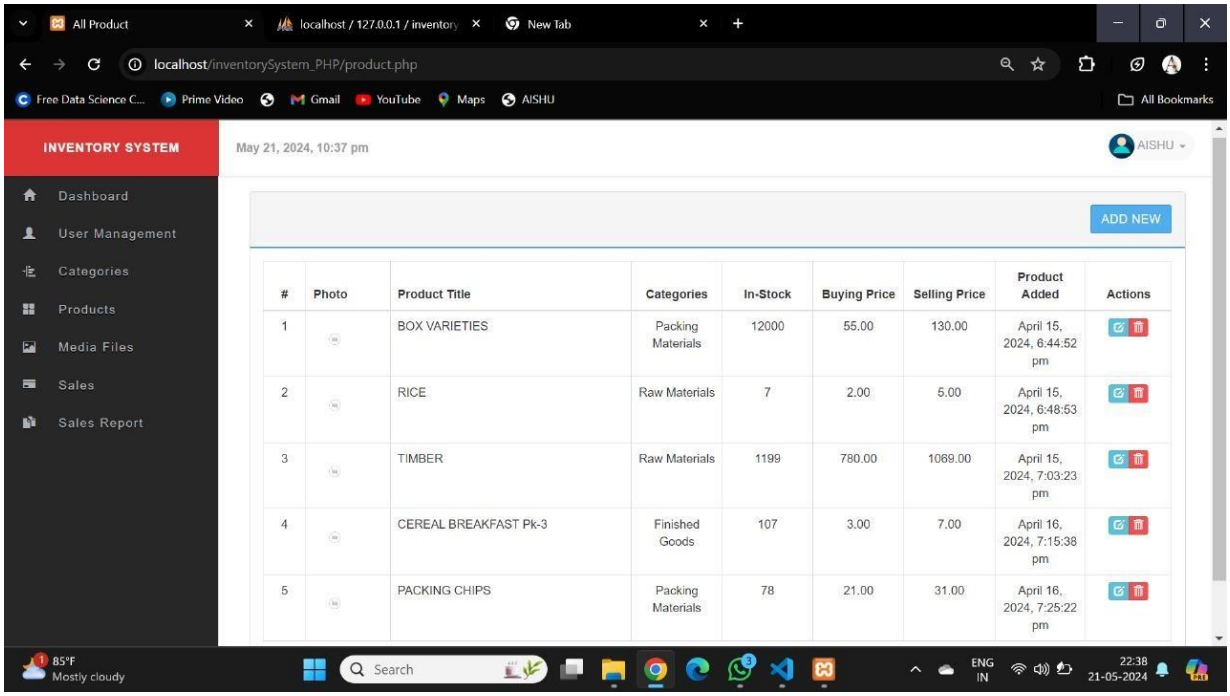
TRIGGERS CREATED:

Trigger to update the corresponding product in the products table after insertion of a new record in the sales table.

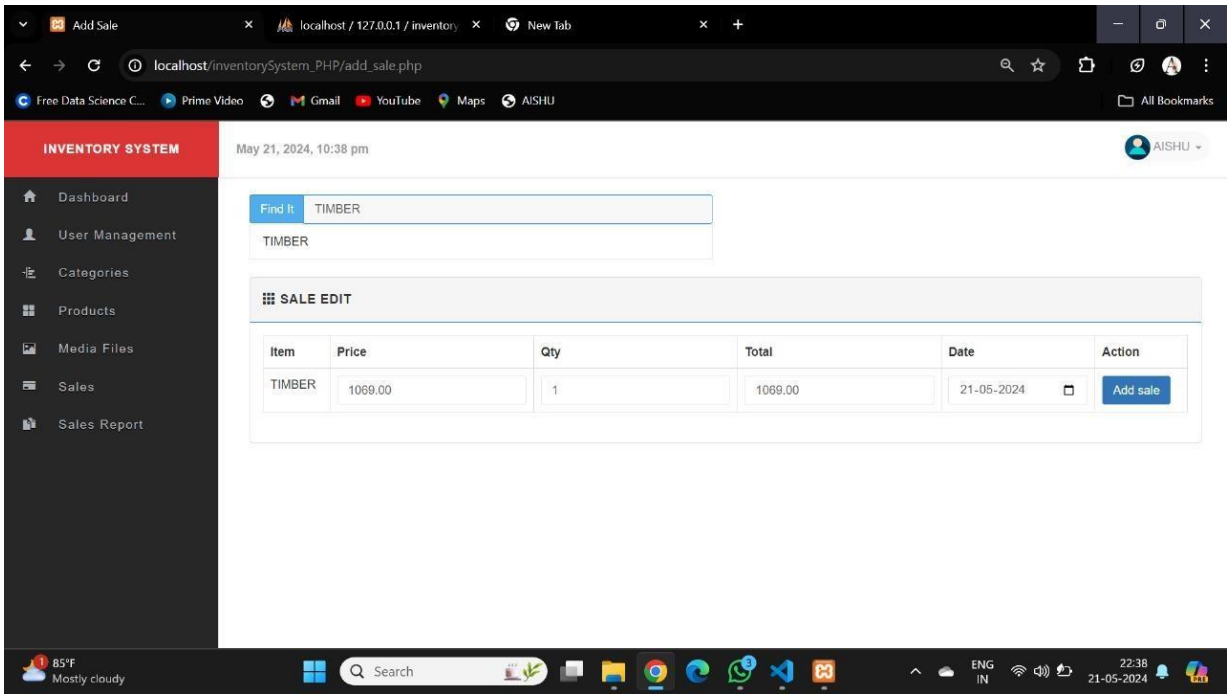
Creating trigger beforeinsertsale:



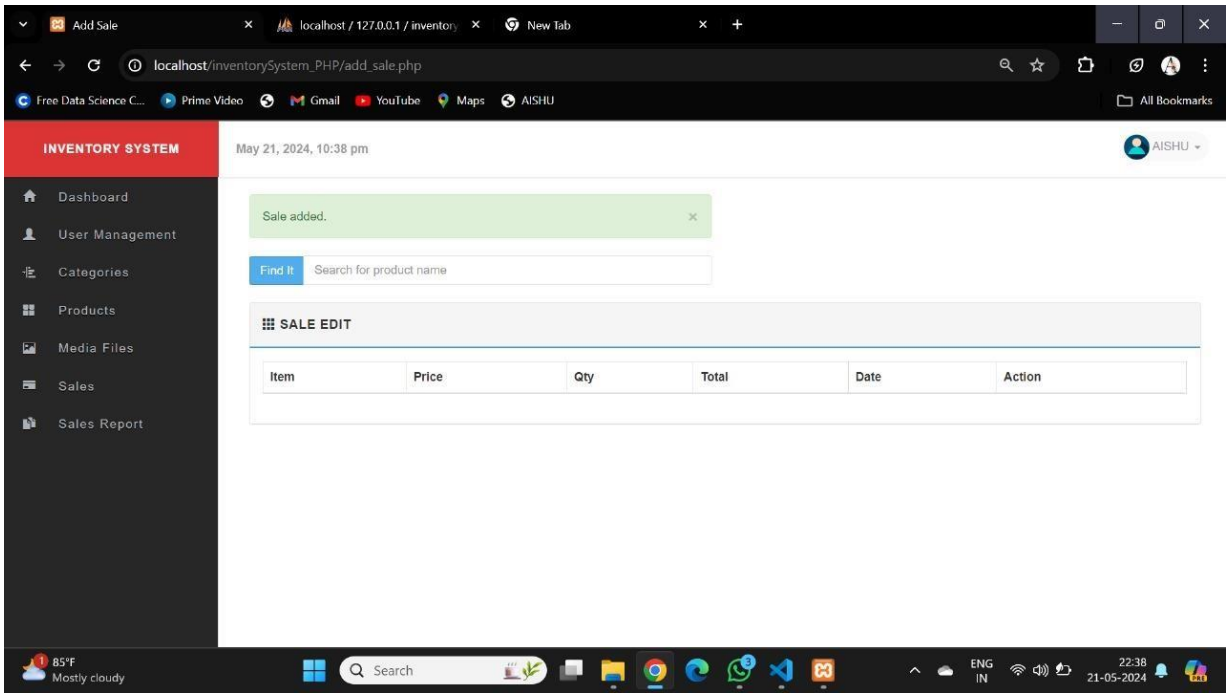
Before inserting record in sales table:



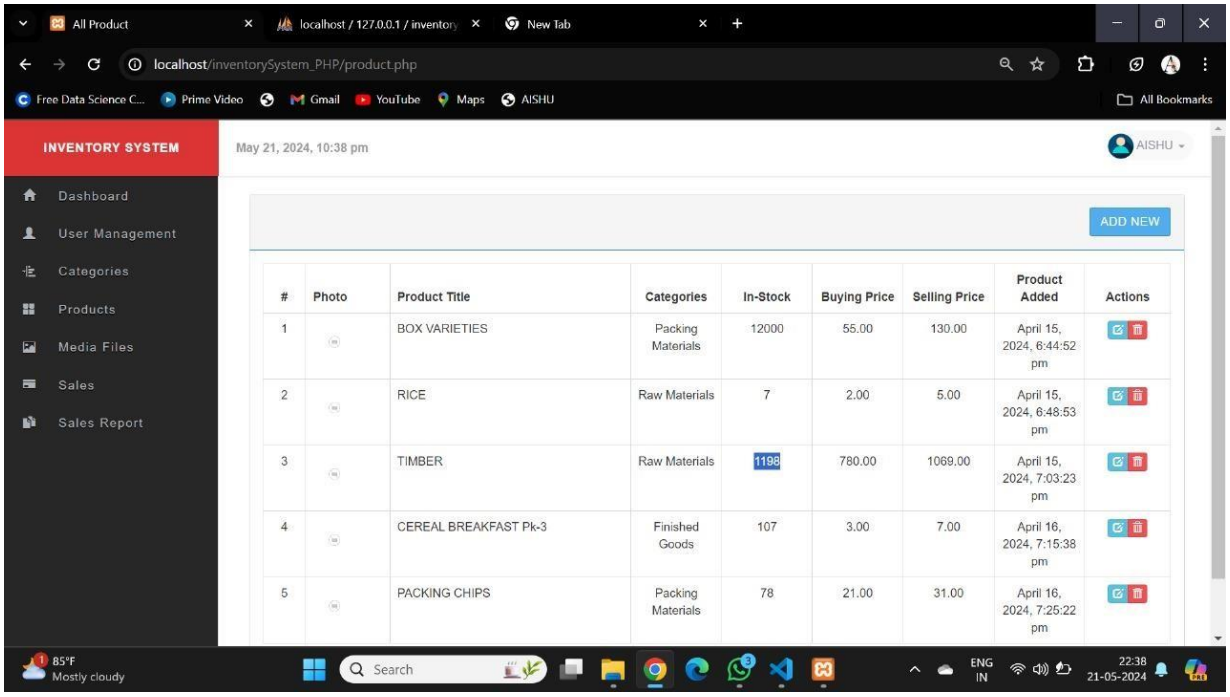
During Insertion:



Successful insertion:



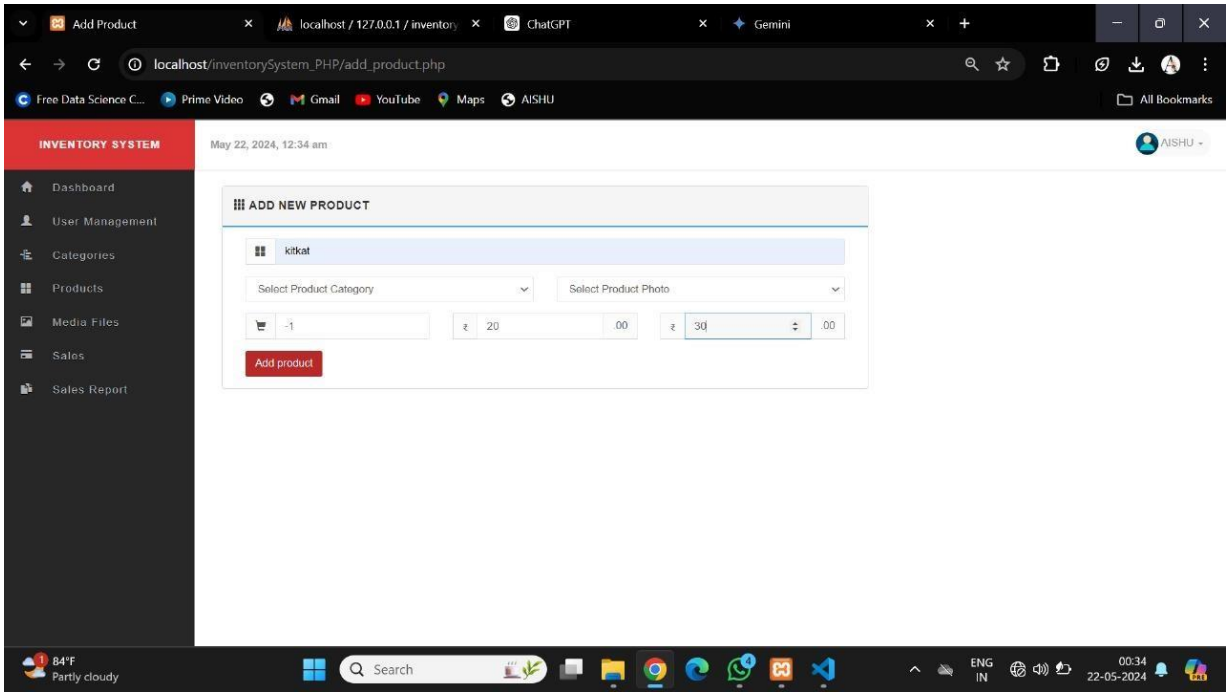
Updation of Product table:



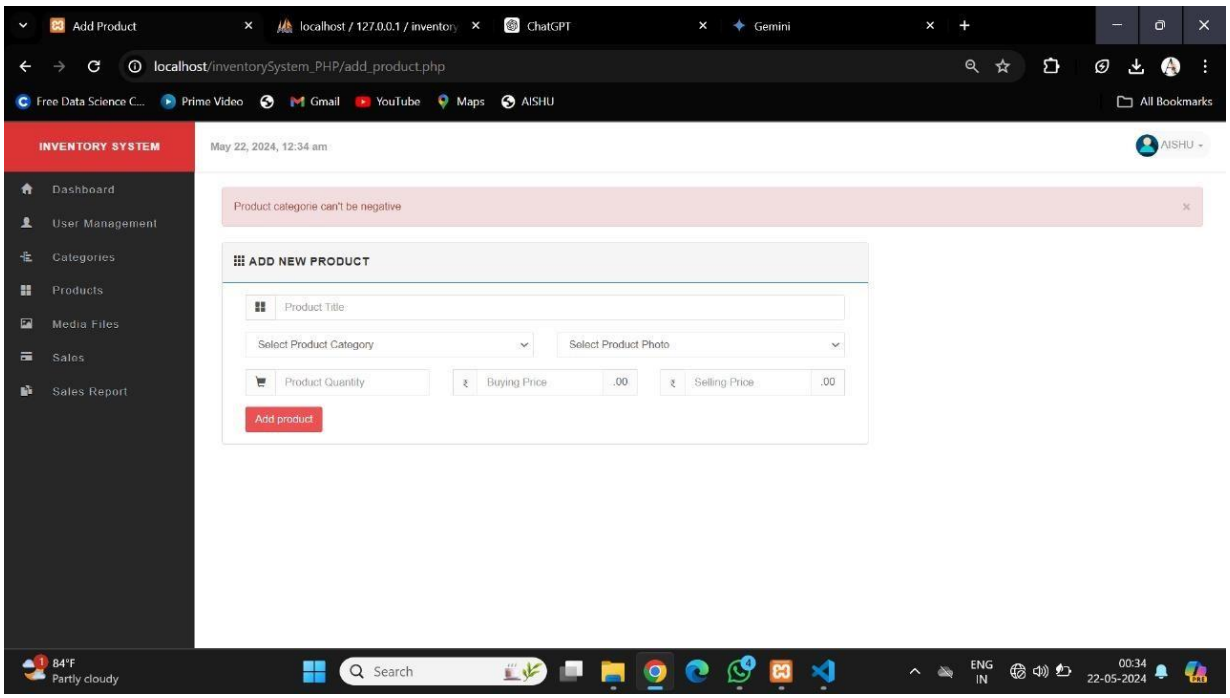
Trigger that throws an error prompt when negative values of product quantities are inserted:

```
DELIMITER //
CREATE TRIGGER non_negative_quantity BEFORE INSERT OR
UPDATE ON products FOR EACH ROW
BEGIN
  IF NEW.quantity < 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Product quantity
cannot be negative!';   END IF;
END //
DELIMITER ;
```

During insertion:



Error prompt:



PROCEDURES:

A stored procedure in SQL is a group of SQL queries that can be saved and reused multiple times. It is very useful as it reduces the need for rewriting SQL queries. It enhances efficiency, reusability, and security in database management.

Users can also pass parameters to stored procedures so that the stored procedure can act on the passed parameter values.

Stored Procedures are created to perform one or more DML operations on the Database.

CREATING A PROCEDURE:

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
 < procedure_body >  END procedure_name;
```

where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

DELETING A PROCEDURE:

Syntax:

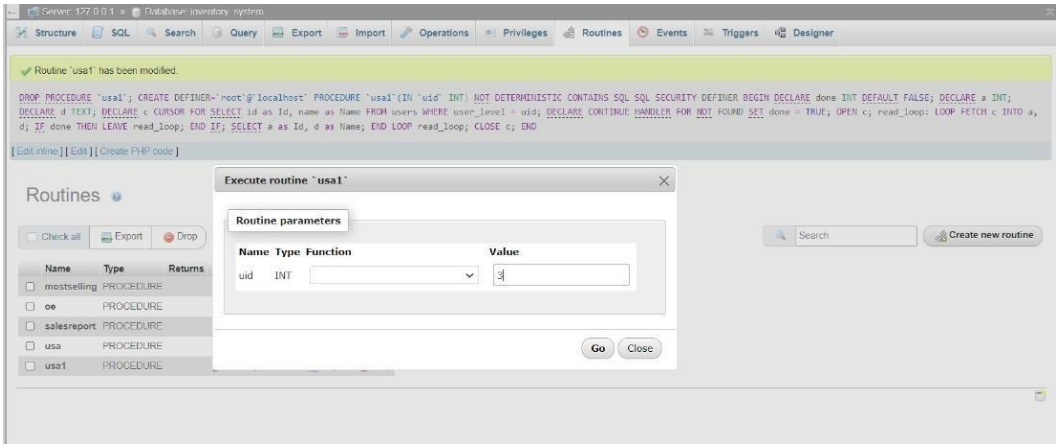
```
DROP PROCEDURE procedure_name;
```

PROEDURES CREATED:

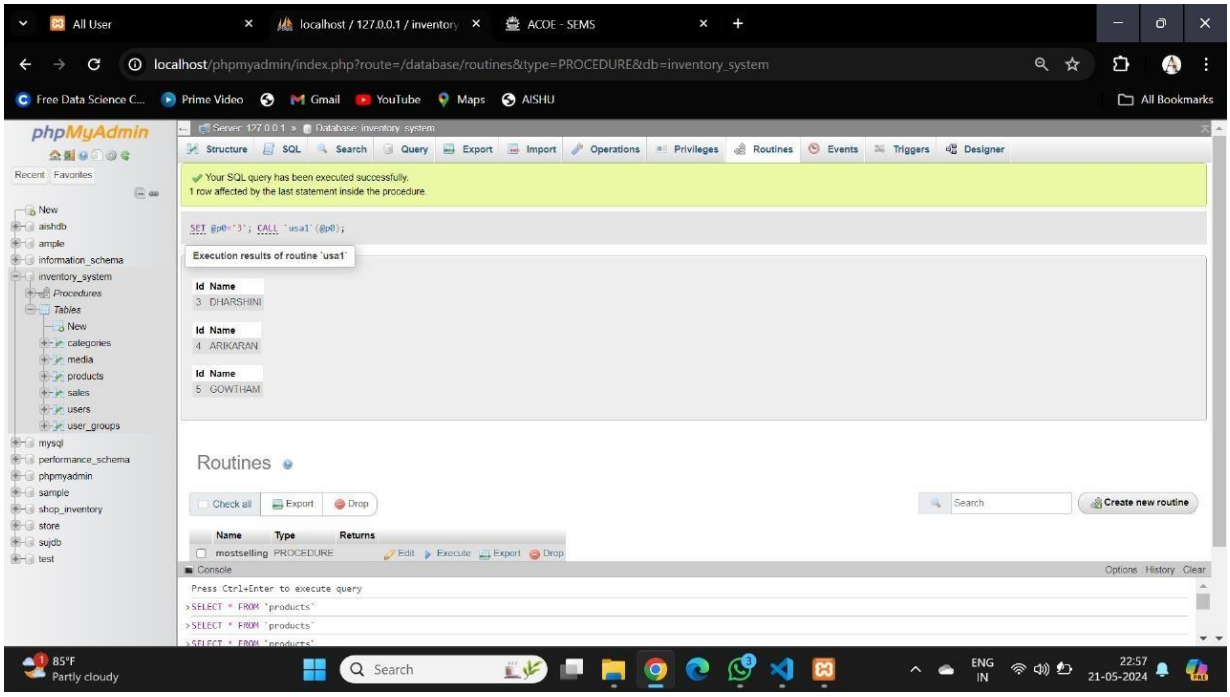
Procedure usa1 to retrieve details from users table whose user_level matches the input entered.

```
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE a INT;
   DECLARE d TEXT;
   DECLARE c CURSOR FOR
      SELECT id as Id, name as Name FROM users WHERE user_level =
uid;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   OPEN c;
read_loop: LOOP
FETCH c INTO a, d;
      IF done THEN
         LEAVE read_loop;
      END IF;
      SELECT a as Id, d as Name;
   END LOOP read_loop;
   CLOSE c;
END
```

Input from the user:



Output:



FUNCTIONS:

A function is same as a procedure except that it returns a value. When a function is created, the definition consists of what the function has to do. It gets executed only when the function is called.

CREATING A FUNCTION:

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a return statement.
- The *RETURN* clause specifies the data type returned by the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

DELETING A FUNCTION:

Syntax:

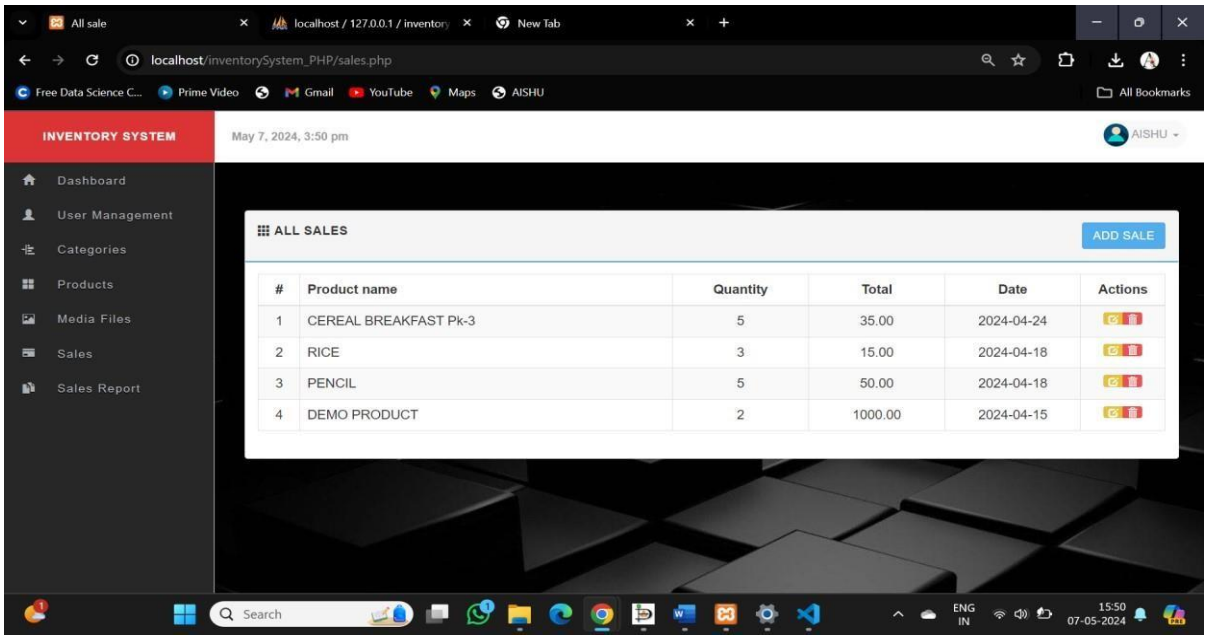
```
DROP FUNCTION function_name;
```

Function to compute the total count of each product sold using aggregate function

```
function count_by_id($table){
  global $db;
  if(tableExists($table))
  {
    $sql    = "SELECT COUNT(id) AS total FROM ".$db->escape($table);
    $result = $db->query($sql);
      return($db->fetch_assoc($result));
  }
}
/*--------------------------------------------------------------*/
/* Determine if database table exists
```

Function for User table and User_group table to display contents using join

```
    return $current_user;
  }
  /*--------------------------------------------------------------*/
  /* Find all user by
  /* Joining users table and user gropus table
  /*--------------------------------------------------------------*/
  function find_all_user(){
      global $db;
      $results = array();
      $sql = "SELECT u.id,u.name,u.username,u.user_level,u.status,u.last_login,";
      $sql .="g.group_name ";
      $sql .="FROM users u ";
      $sql .="LEFT JOIN user_groups g ";
      $sql .="ON g.group_level=u.user_level ORDER BY u.name ASC";
      $result = find_by_sql($sql);
      return $result;
  }
  /*--------------------------------------------------------------*/
  /* Function to update the last log in of a user
```



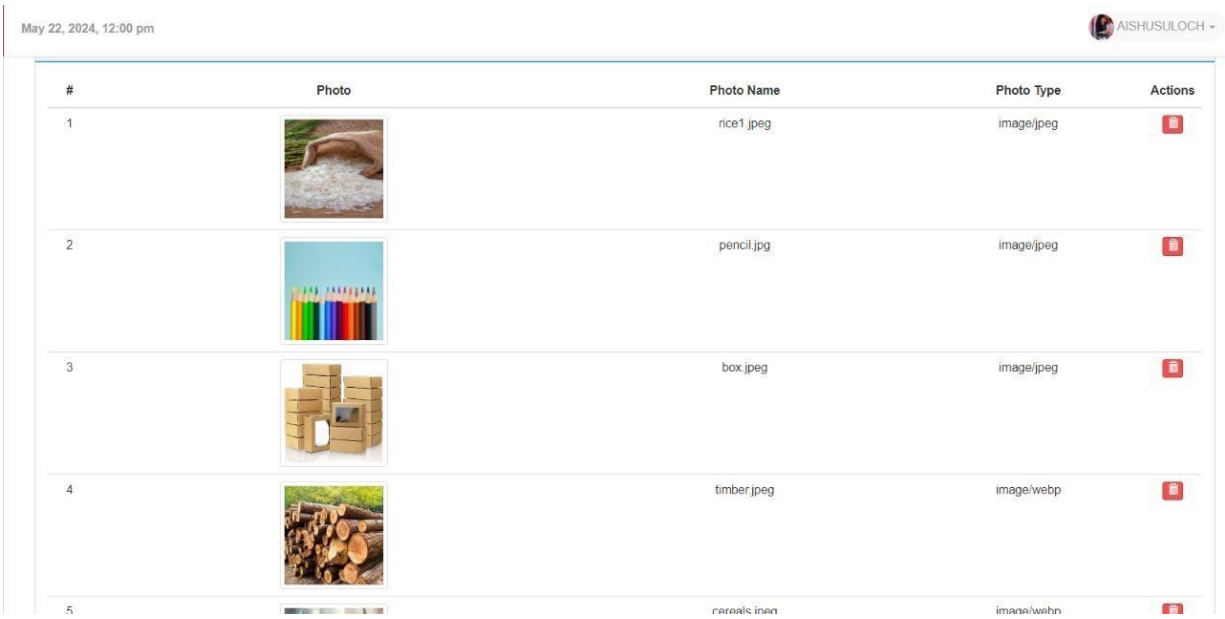
Function that updates image of product upon insertion of a media file

```
);
public$upload_extensions = array(
 'gif',
 'jpg',
 'jpeg',
 'png',
);
public function file_ext($filename){
    $ext = strtolower(substr( $filename, strrpos( $filename, '.' ) + 1 ) );
    if(in_array($ext, $this->upload_extensions)){
        return true;
    }
 }

public function upload($file)
{
  if(!$file || empty($file) || !is_array($file)):
    $this->errors[] = "No file was uploaded.";
    return false;
  elseif($file['error'] != 0):
    $this->errors[] = $this->upload_errors[$file['error']];
    return false;
  elseif(!$this->file_ext($file['name'])):
    $this->errors[] = 'File not right format ';
    return false;
  else:
    $this->imageInfo = getimagesize($file['tmp_name']);
    $this->fileName  = basename($file['name']);
    $this->fileType  = $this->imageInfo['mime'];
    $this->fileTempPath = $file['tmp_name'];
   return true;
  endif;

}
```

Images:

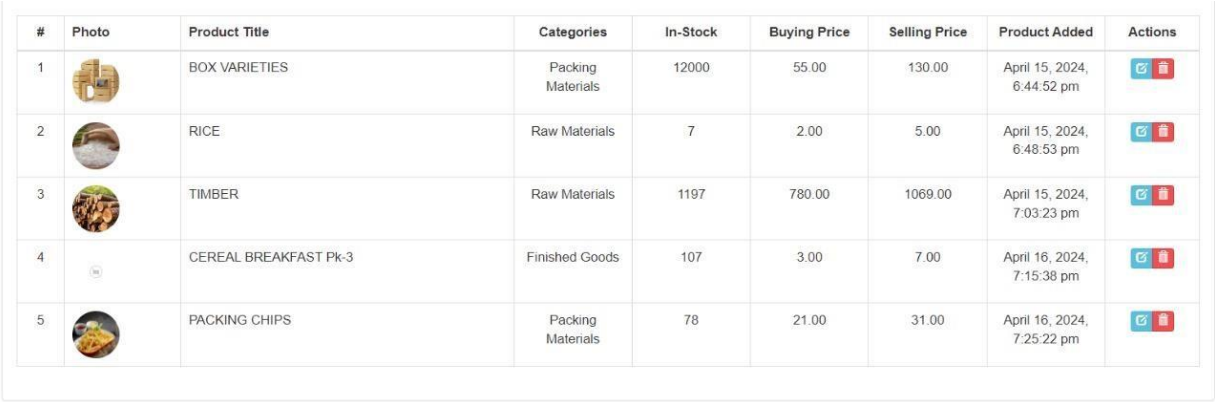


May 22, 2024, 12:00 pm

AISHUSULOCH

#	Photo	Photo Name	Photo Type	Actions
1		rice1.jpeg	image/jpeg	
2		pencil.jpg	image/jpeg	
3		box.jpeg	image/jpeg	
4		timber.jpeg	image/webp	
5		cereals.jpeg	image/webp	

After updation:

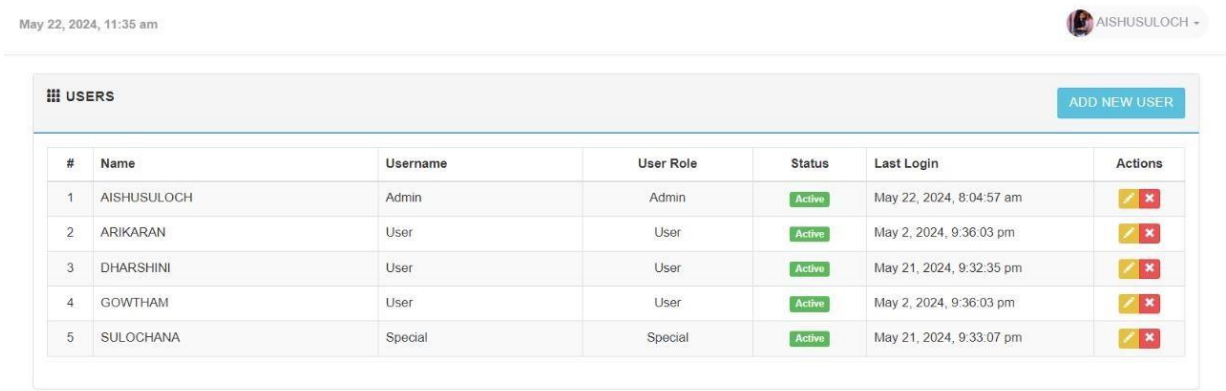


#	Photo	Product Title	Categories	In-Stock	Buying Price	Selling Price	Product Added	Actions
1		BOX VARIETIES	Packing Materials	12000	55.00	130.00	April 15, 2024, 6:44:52 pm	
2		RICE	Raw Materials	7	2.00	5.00	April 15, 2024, 6:48:53 pm	
3		TIMBER	Raw Materials	1197	780.00	1069.00	April 15, 2024, 7:03:23 pm	
4		CEREAL BREAKFAST Pk-3	Finished Goods	107	3.00	7.00	April 16, 2024, 7:15:38 pm	
5		PACKING CHIPS	Packing Materials	78	21.00	31.00	April 16, 2024, 7:25:22 pm	

Function that displays the log in time using session:

```
120
121 /*--------------------------------------------------------------*/
122 /* Find current log in user by session id
123 /*--------------------------------------------------------------*/
124 function current_user(){
125     static $current_user;
126     global $db;
127     if(!$current_user){
128         if(isset($_SESSION['user_id'])):
129             $user_id = intval($_SESSION['user_id']);
130             $current_user = find_by_id('users',$user_id);
131         endif;
132     }
133     return $current_user;
134 }
135 /*--------------------------------------------------------------*/
136 /* Find all user by
137 /* Joining users table and user gropus table
```

Output:



#	Name	Username	User Role	Status	Last Login	Actions
1	AISHUSULOCH	Admin	Admin	Active	May 22, 2024, 8:04:57 am	
2	ARIKARAN	User	User	Active	May 2, 2024, 9:36:03 pm	
3	DHARSHINI	User	User	Active	May 21, 2024, 9:32:35 pm	
4	GOWTHAM	User	User	Active	May 2, 2024, 9:36:03 pm	
5	SULOCHANA	Special	Special	Active	May 21, 2024, 9:33:07 pm	

CURSORS:

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

There are two types of cursors:

1. Implicit cursors
2. Explicit cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement whereas explicit cursors are programmer-defined cursors for gaining more control over the context area.

CREATING A CURSOR:

Syntax:

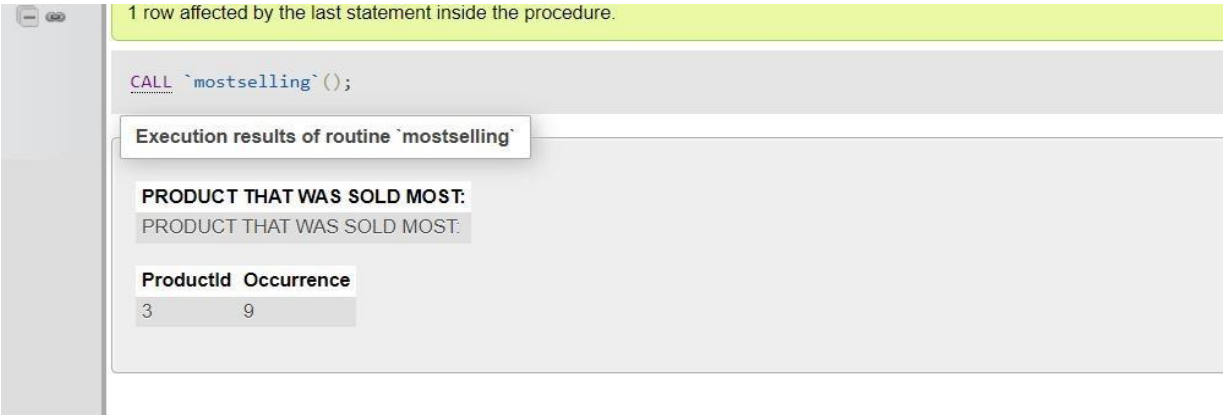
```
CURSOR cursor_name IS select_statement;
```

CURSORS CREATED:

Cursor to retrieve the most sold product from the sales table

```
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE a INT;
   DECLARE d INT;
   DECLARE first_occurrence_printed BOOLEAN DEFAULT FALSE;
   DECLARE c CURSOR FOR
      SELECT product_id, COUNT(*) AS occurrence
      FROM sales
      GROUP BY product_id
      ORDER BY occurrence DESC;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   OPEN c;
   SELECT 'PRODUCT THAT WAS SOLD MOST:';
   read_loop: LOOP
      FETCH c INTO a, d;
      IF done OR first_occurrence_printed THEN
         LEAVE read_loop;
      END IF;
     SELECT a AS ProductId, d AS Occurrence;
      SET first_occurrence_printed = TRUE;
   END LOOP read_loop;
   CLOSE c;
END
```

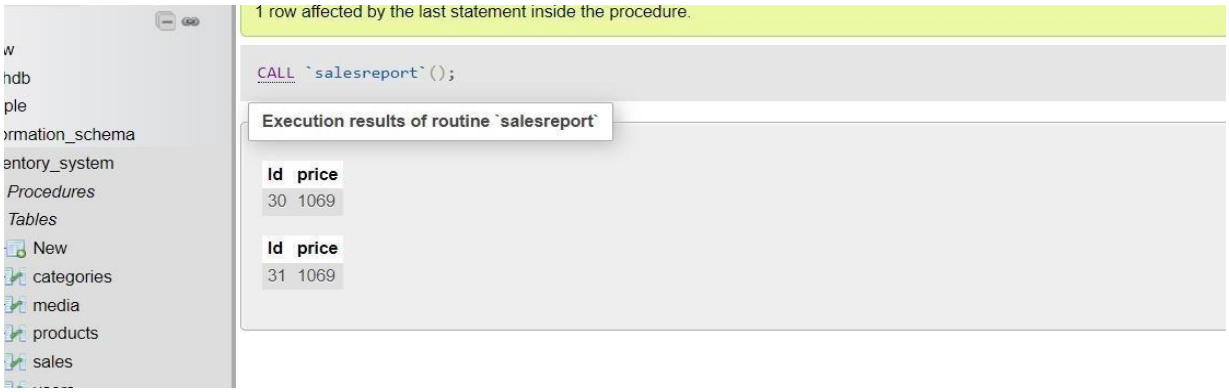
Output:



Cursor to display the costliest sold product from the sales table

```
BEGIN
 DECLARE done INT DEFAULT FALSE;
   DECLARE a INT;
   DECLARE d int;
   DECLARE c CURSOR FOR
      SELECT id,price  FROM sales where sales.price=(select max(price) from sales );
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   OPEN c;
read_loop: LOOP
FETCH c INTO a,d;
      IF done THEN
         LEAVE read_loop;
      END IF;
      SELECT a as Id,d as price;
   END LOOP read_loop;
   CLOSE c;
END
```

Output:

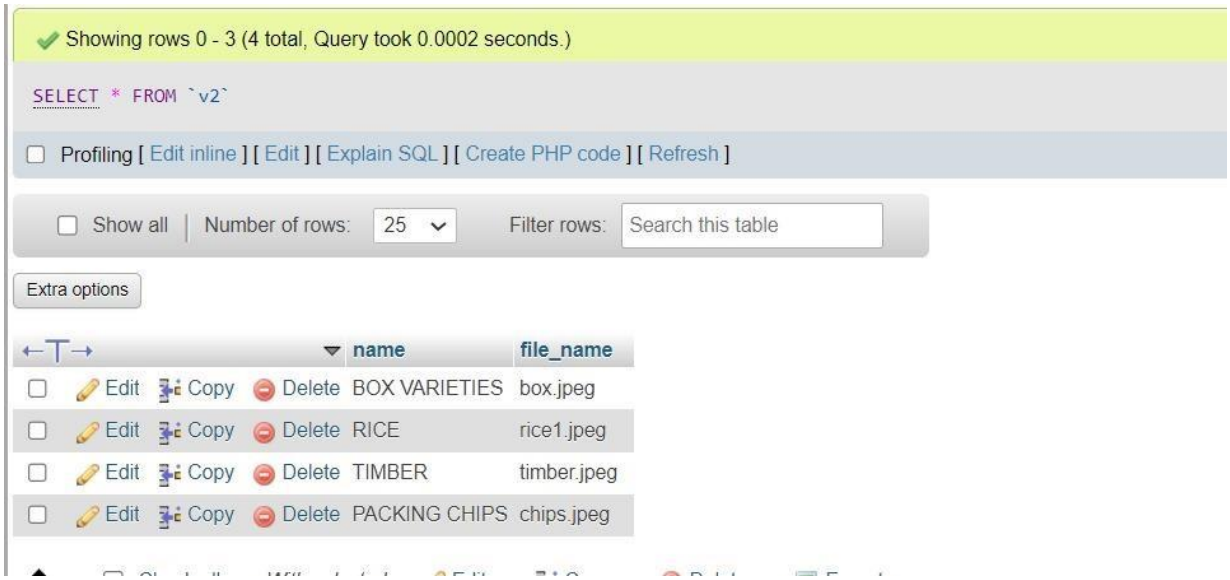


VIEWS:

View to display the names of products and their corresponding media images using catesian product.

```
Create view v2 as
select products.name,
media.file_name  from media,
products
where media.id= products. media_id;
```

Output:



INFERENCE:

This Inventory Management System (IMS) designed addresses the inefficiencies of traditional inventory systems for small businesses. Through streamlined stock control, waste minimization, and optimized inventory levels, the IMS empowers entrepreneurs to focus on growth while enhancing customer satisfaction, promising a user-friendly and efficient solution. By leveraging database components like triggers, functions, procedures, cursors, and views, the IMS ensures data integrity, automation of tasks, and simplified data access, ultimately revolutionizing inventory management processes for small ventures.