DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CS6106

DATABASE MANAGEMENT SYSTEMS

PROJECT

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:

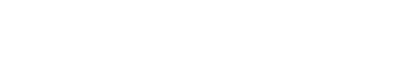
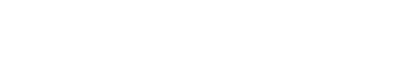
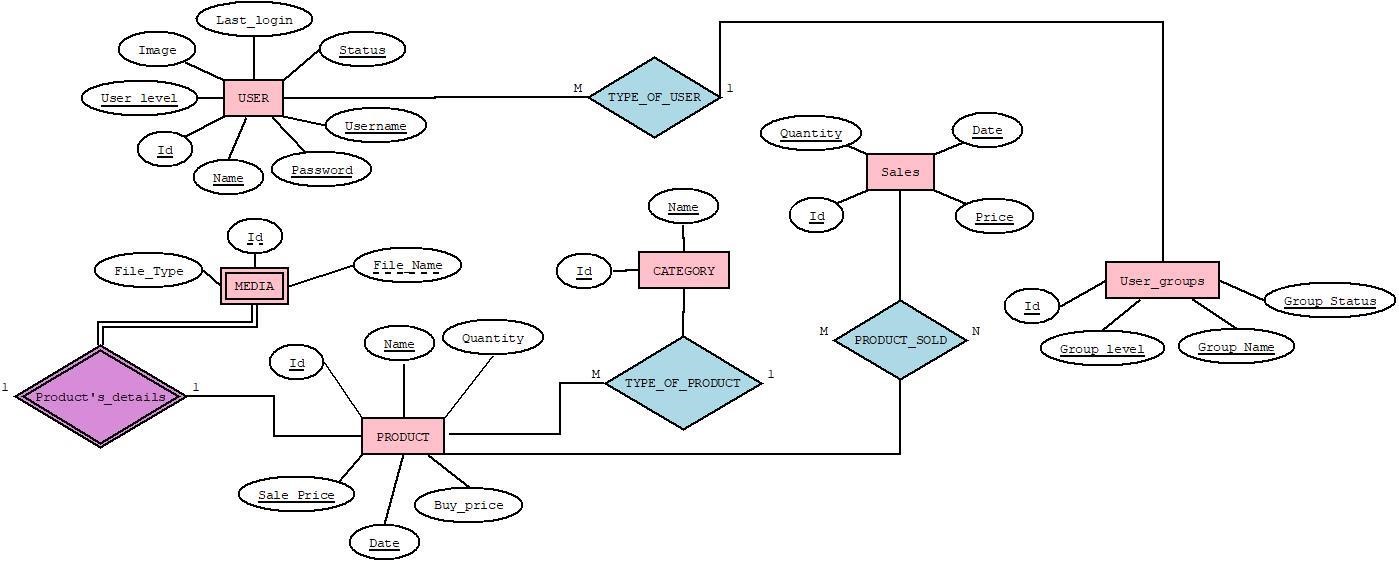
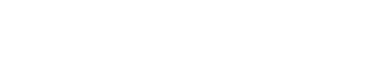
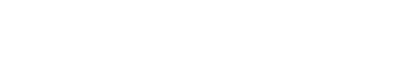
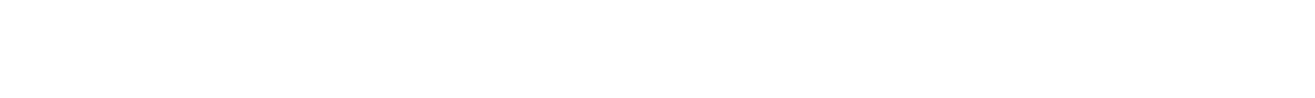
In the dynamic landscape of small business development, entrepreneurs face a critical challenge; the inefficiency and complexity of traditional inventory management systems. Existing solutions often prove cumbersome, time-consuming, and require a steep learning curve, hindering the growth potential of 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:

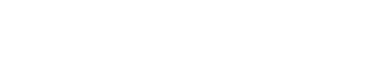
Introducing an innovative inventory management system designed for small businesses. Our solution streamlines stock control, minimizes waste, and optimizes inventory levels, empowering entrepreneurs to focus on growth while enhancing customer satisfaction. Imagine a seamless, user-friendly tool that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

ER - ENTITIES:

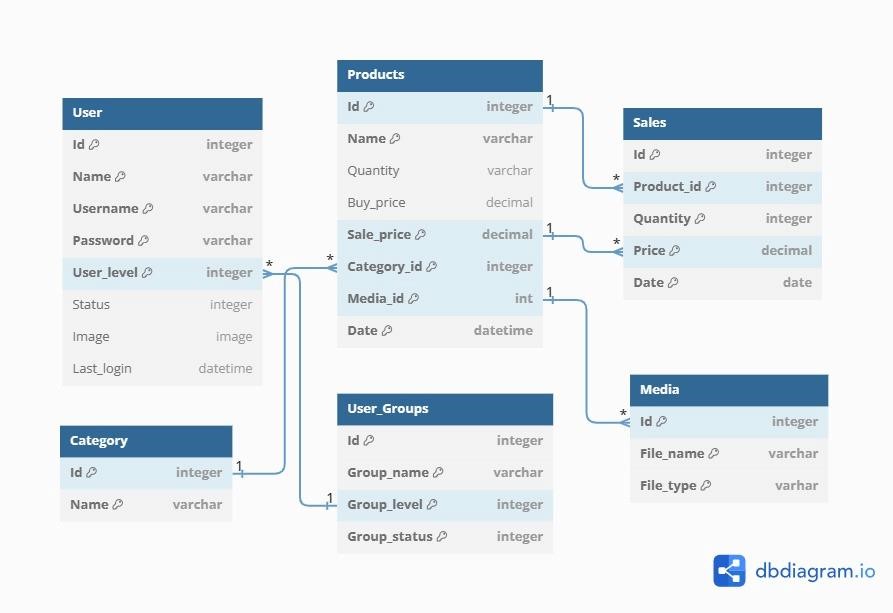
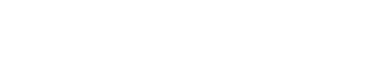
* ADMIN/ USER/ SPECIAL
* PRODUCTS
* MEDIA (WEAK ENTITY)
* SALES
* USER\_GROUPS
* CATEGORIES



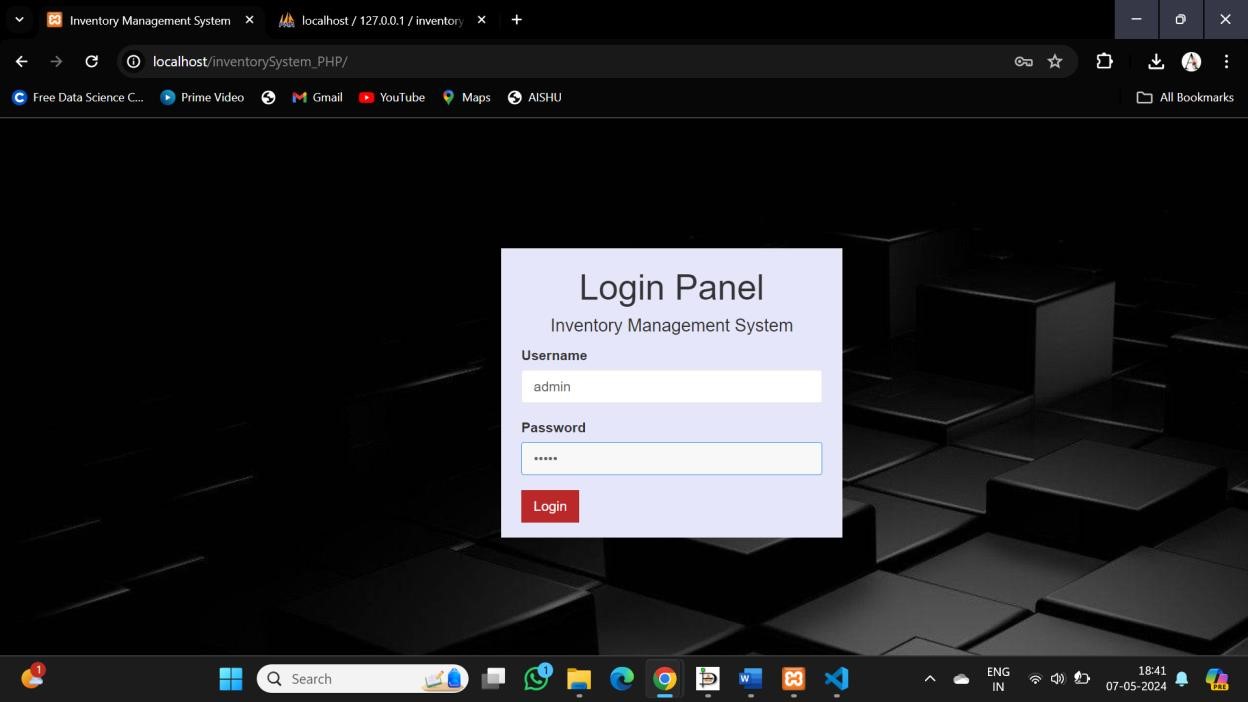
**RELATIONAL**



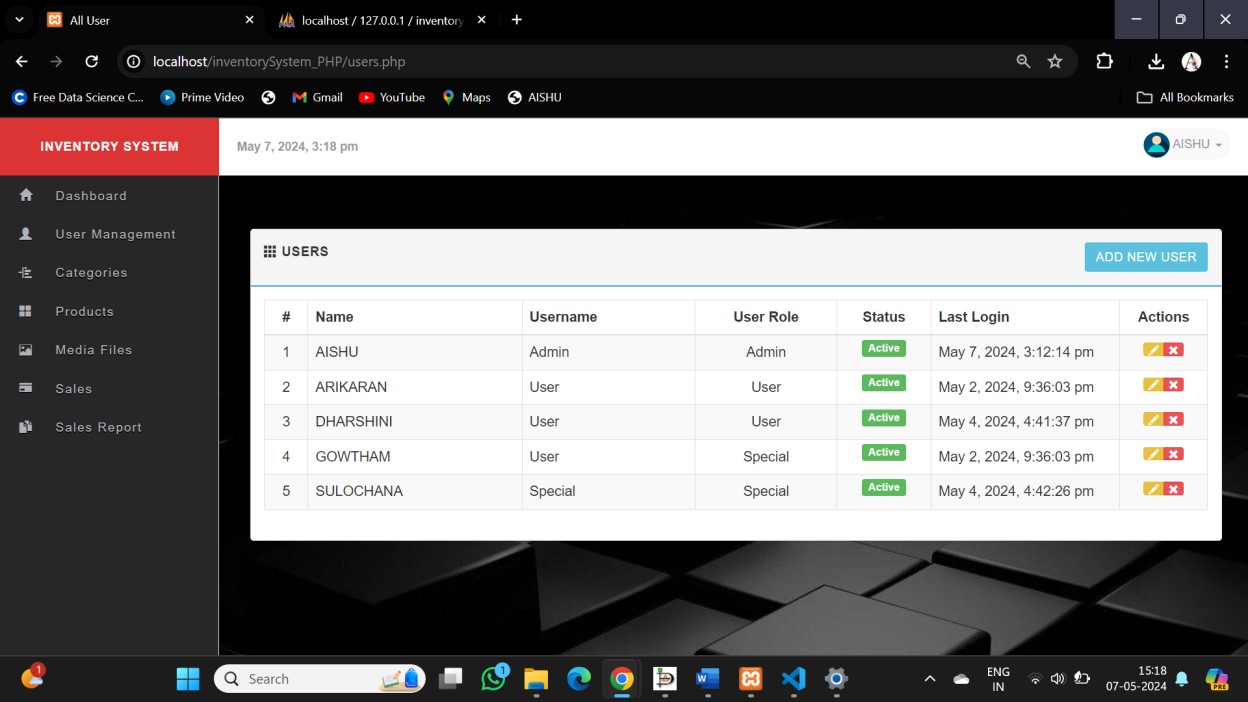
**SCHEMA**



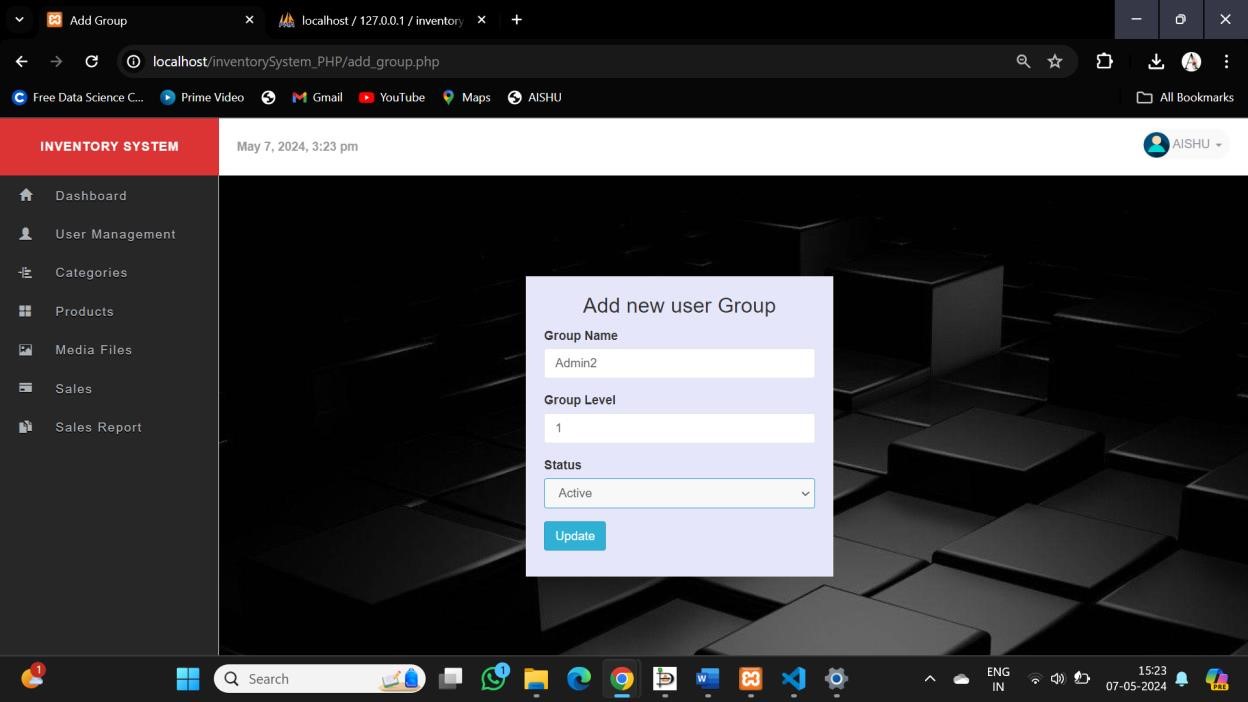
LOGIN PAGE:



USER (Table):

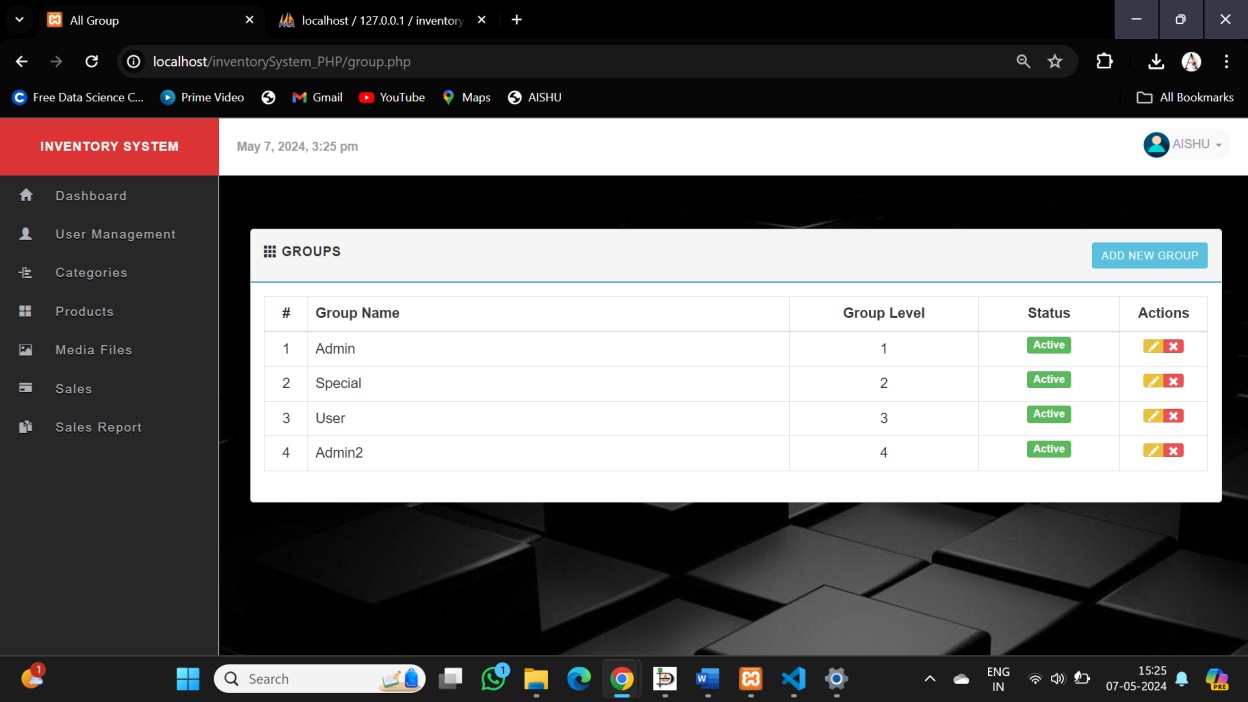


ADDING NEW GROUPS TO USER\_GROUP TABLE:



RECORD INSERTED

:



TRIGGERS, FUNCTIONS, PROCEDURES & CURSORS

**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

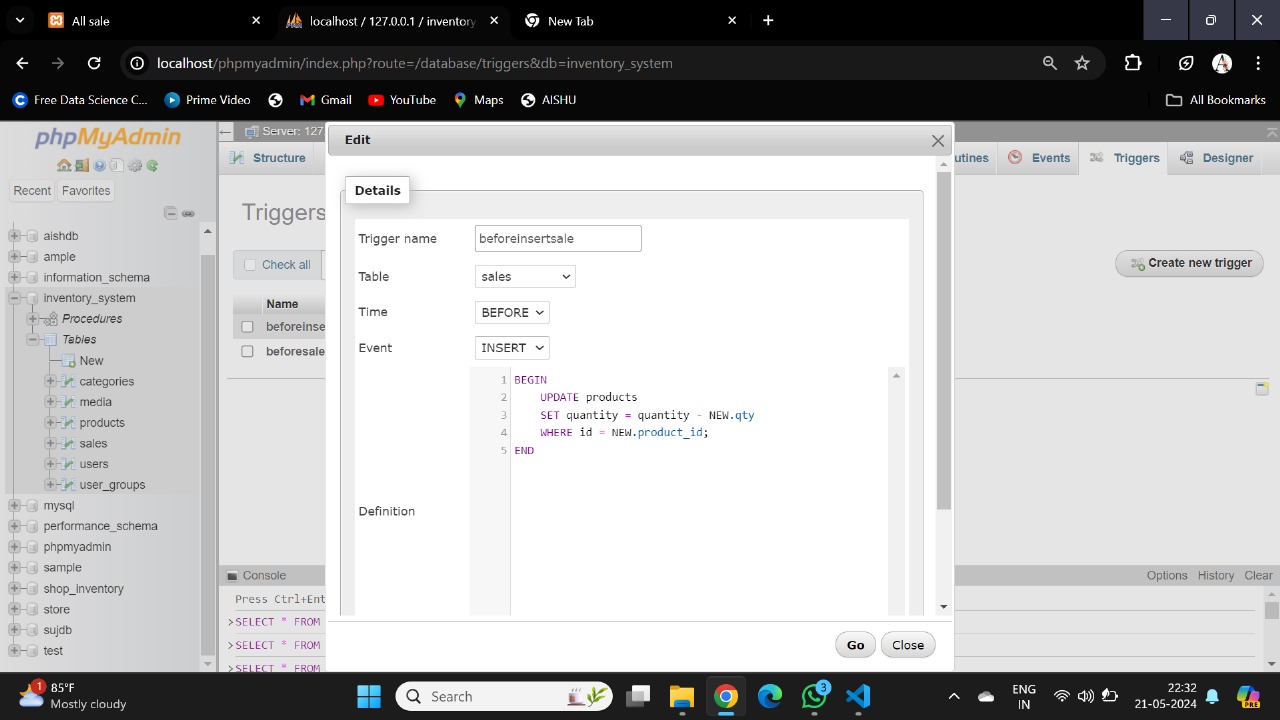
Exception-handling-statements

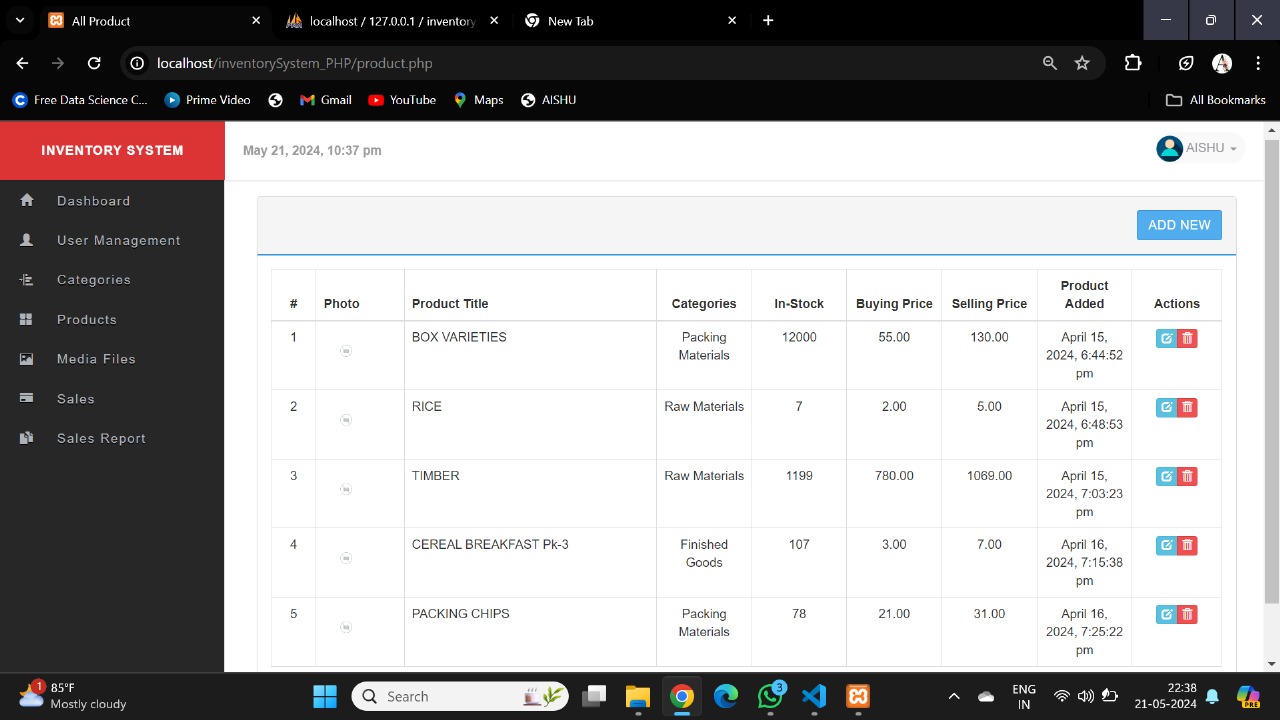
END;

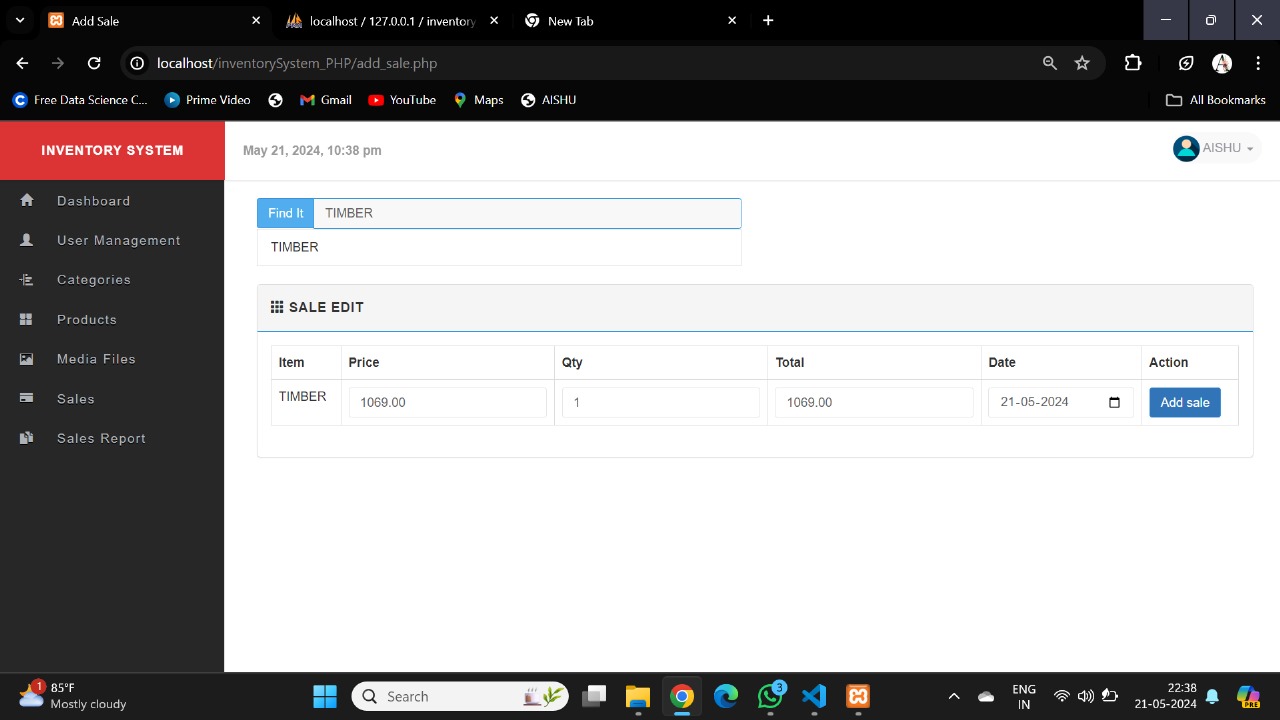
**TRIGGERS CREATED:**

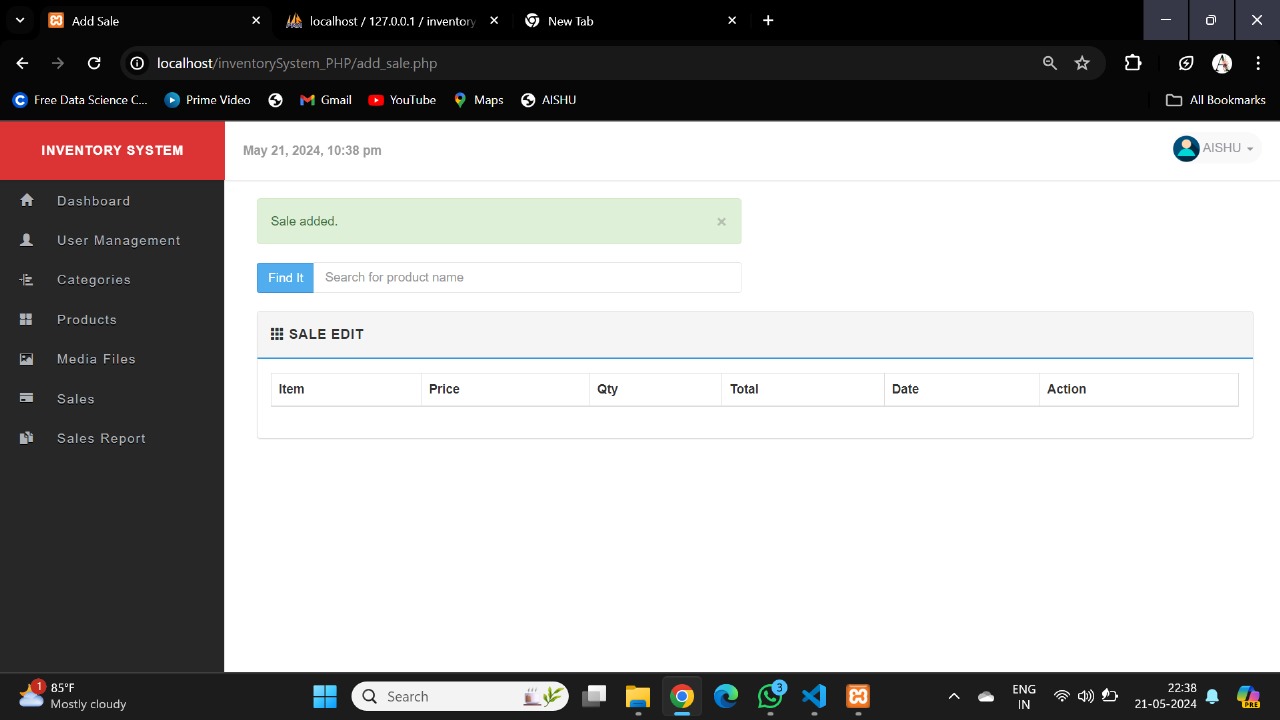
Trigger to update the corresponding product in the products table after insertion of a new record in the sales table.

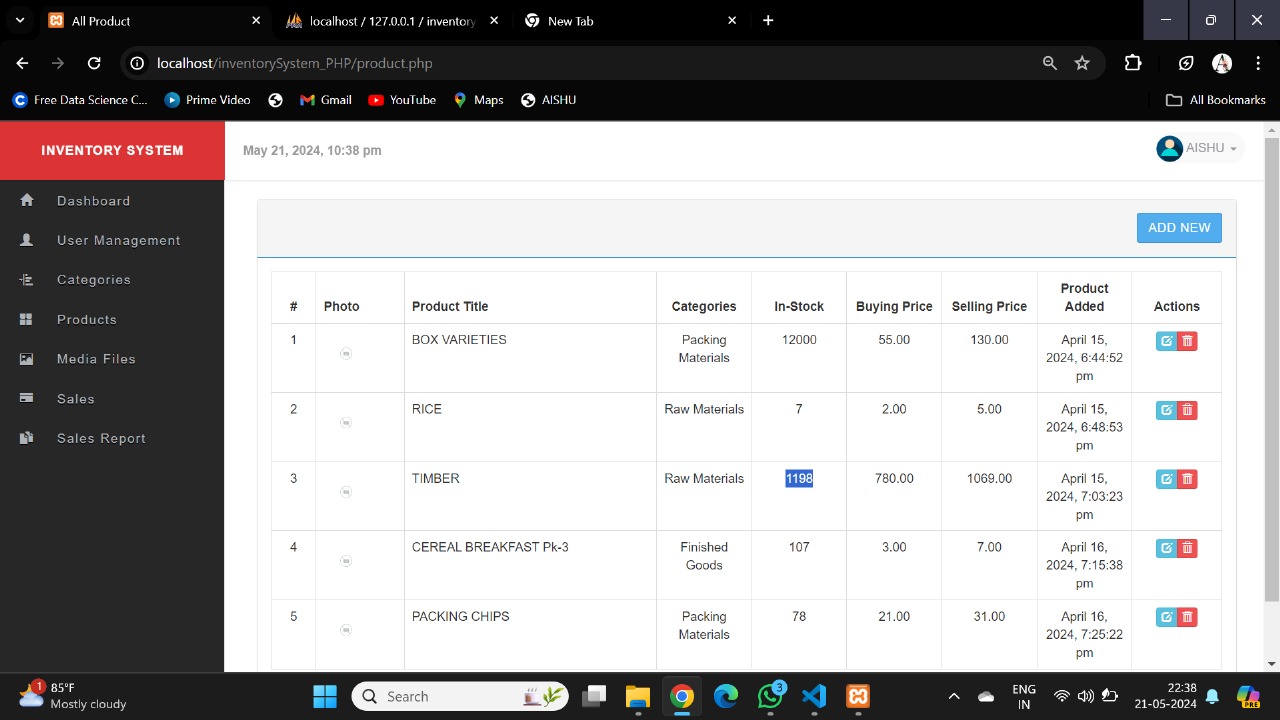
Creating trigger beforeinsertsale:



****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

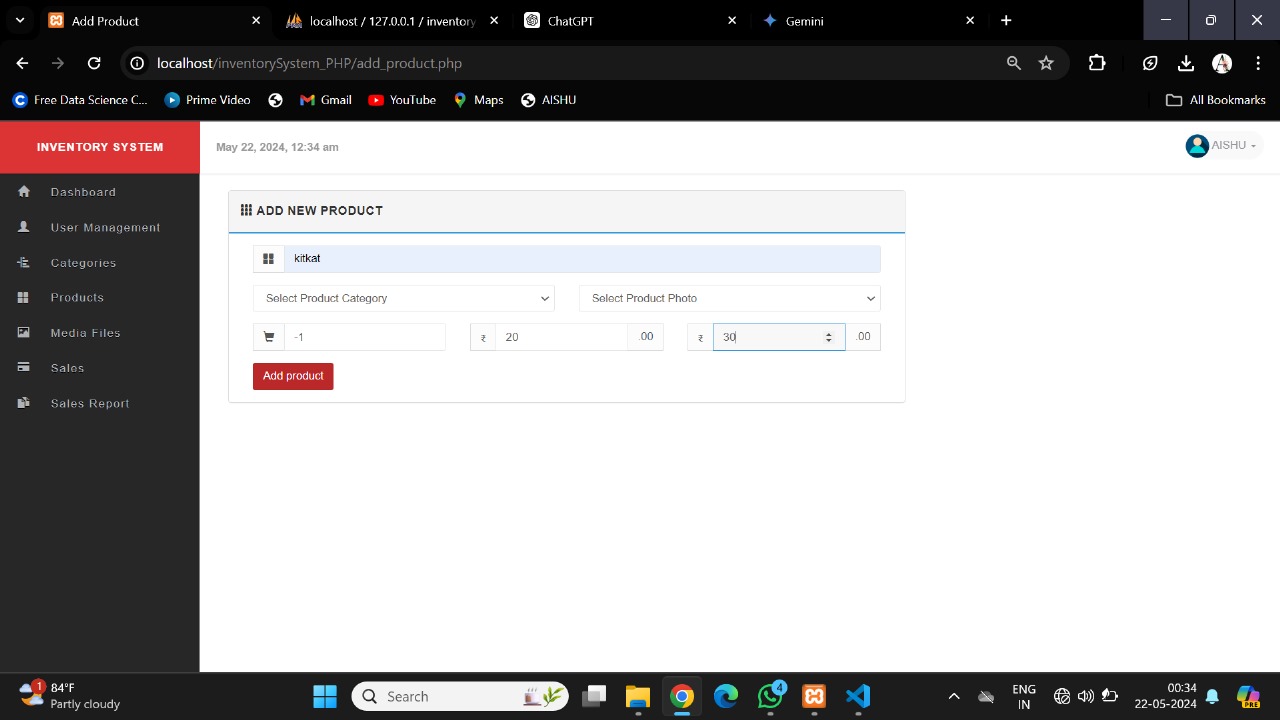
IF NEW.quantity < 0 THEN

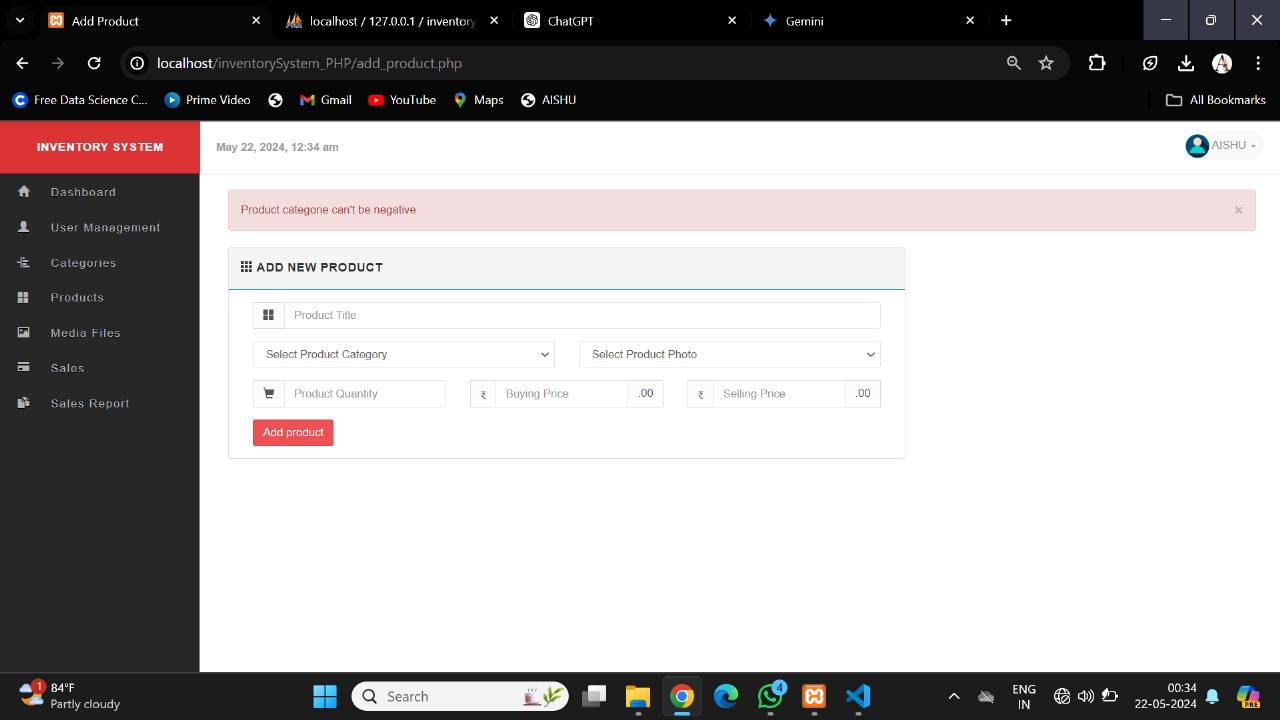
SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Product quantity cannot be negative!';

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:



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:



**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;

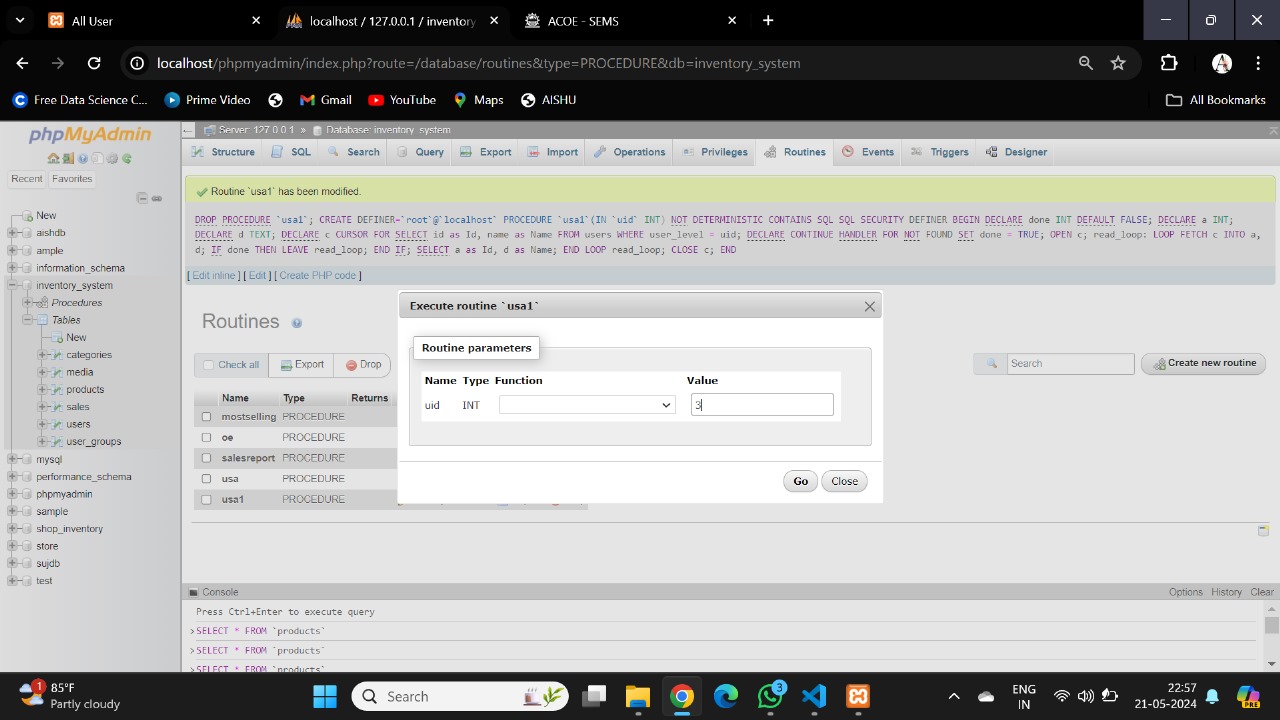
END IF;

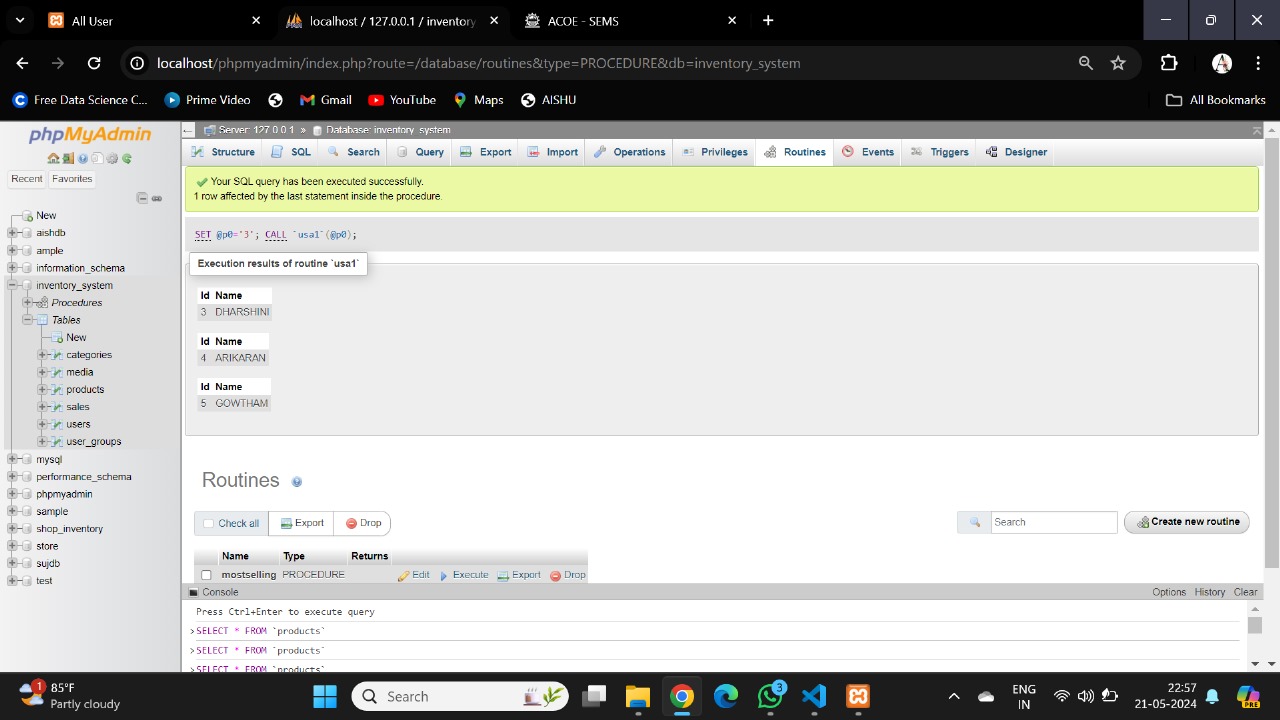
SELECT a as Id, d as Name;

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:



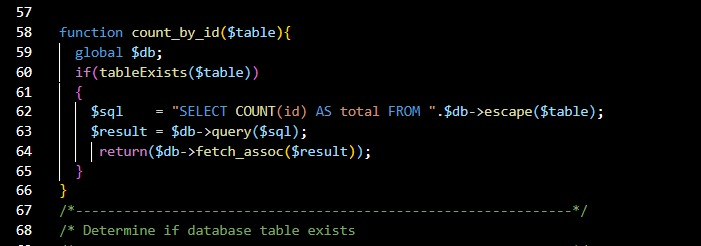
* where,
* *function-name* specifies the name of the function.
* [OR REPLACE] option allows the modification of an existing function.
* The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter 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.

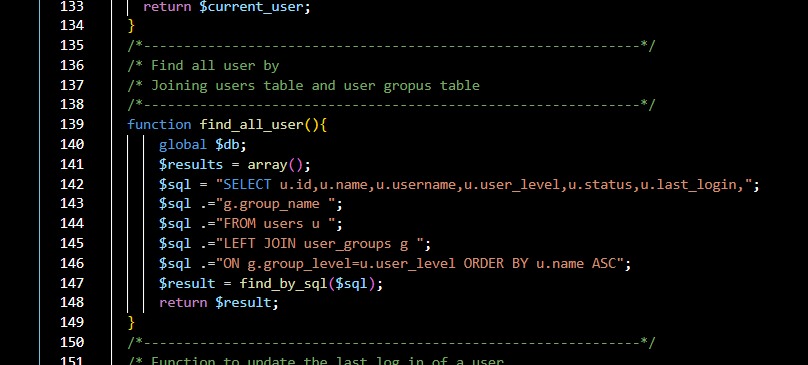
**DELETING A FUNCTION:**

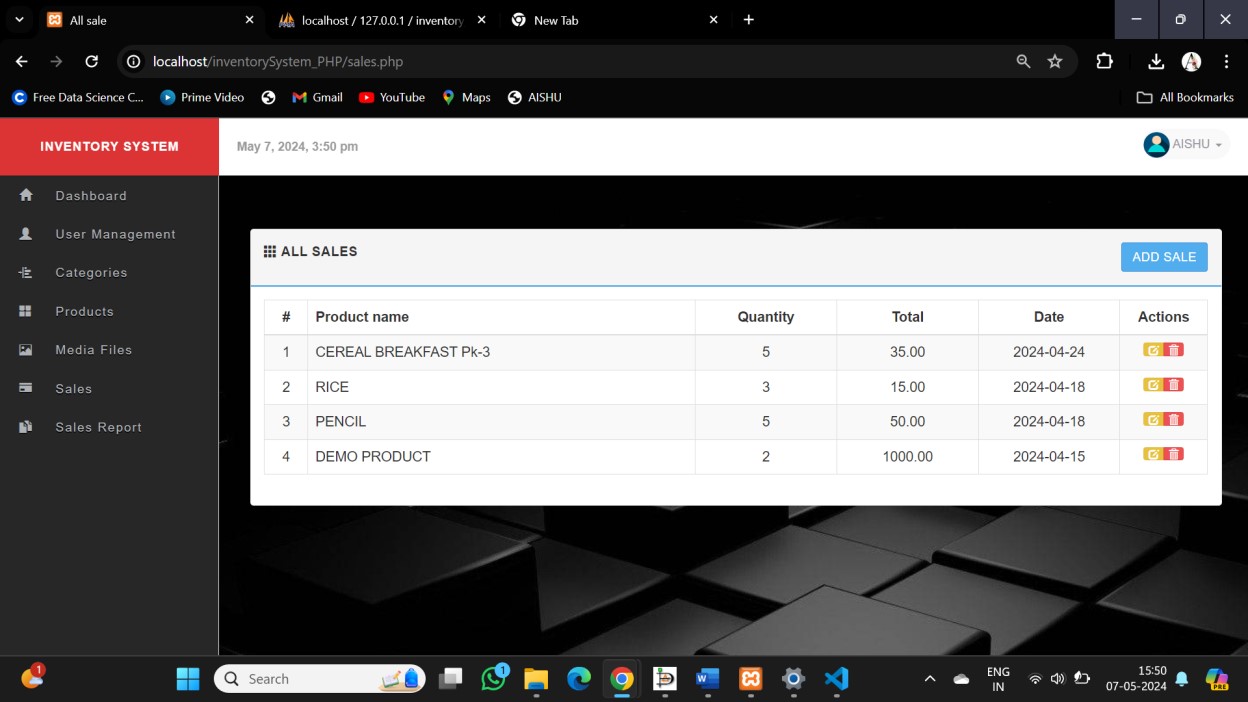
Syntax:



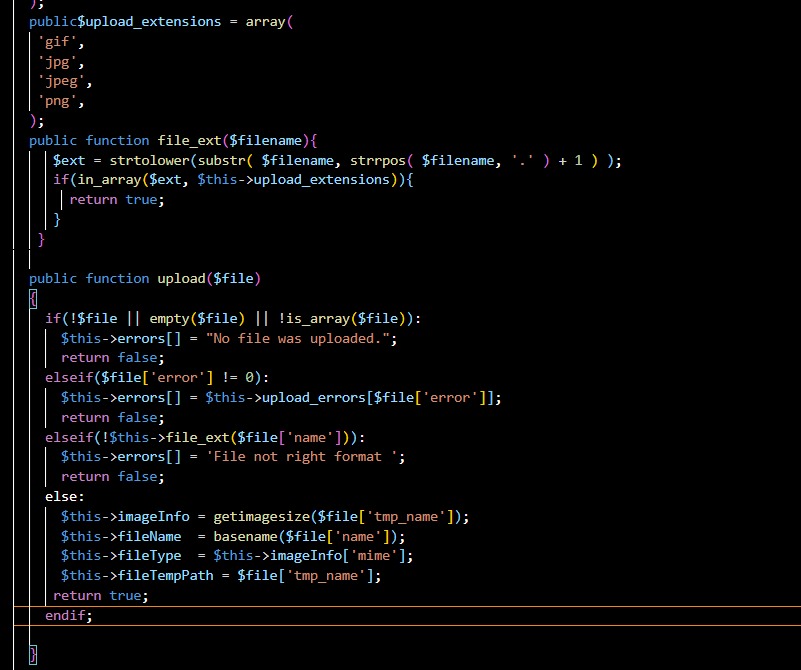
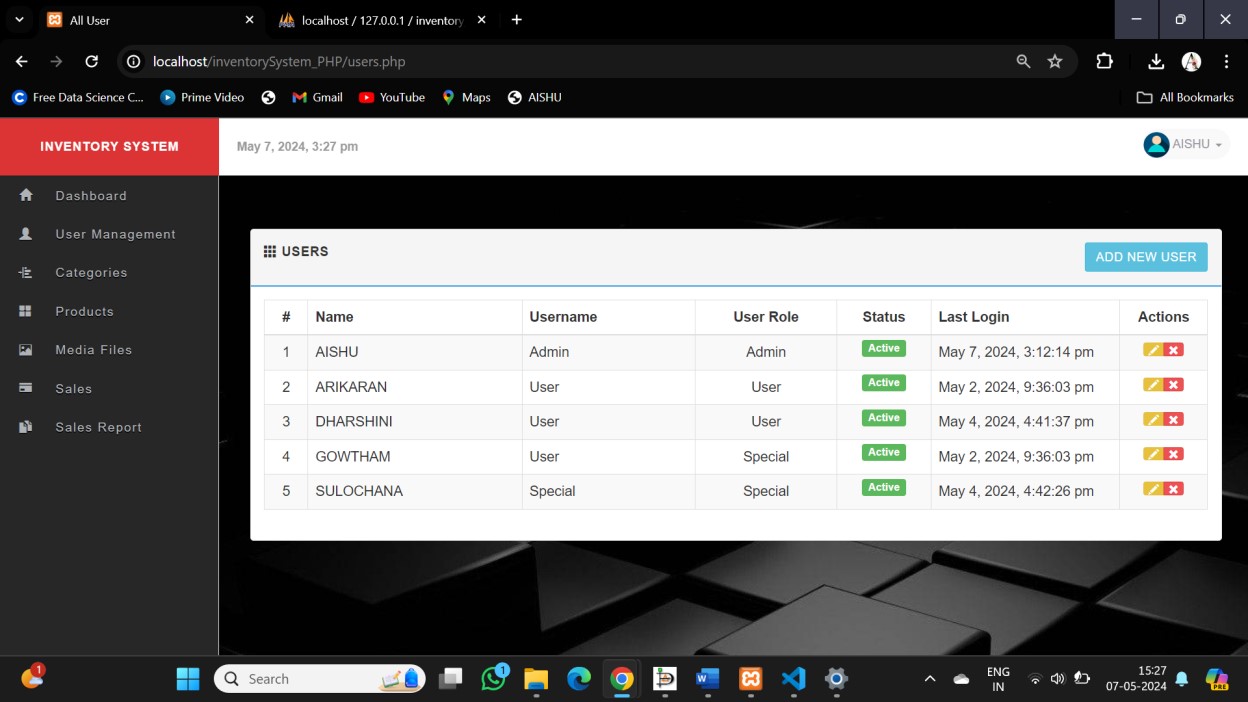
Function to compute the total count of each product sold using aggregate function



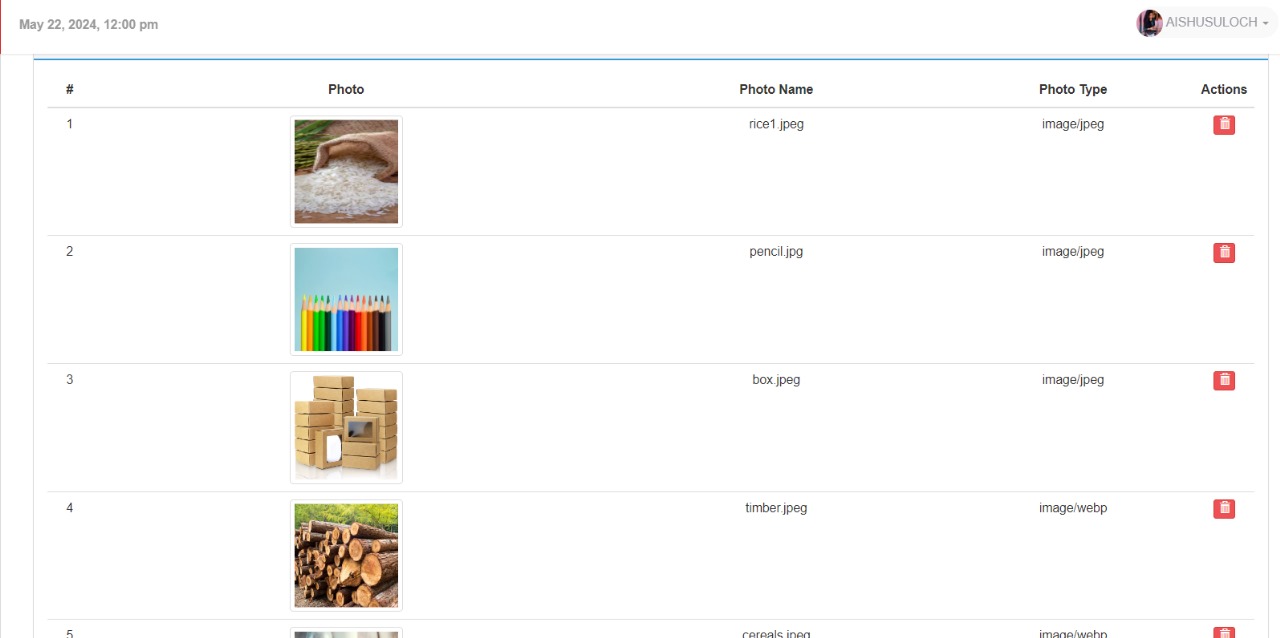
Function for User table and User\_group table to display contents using join

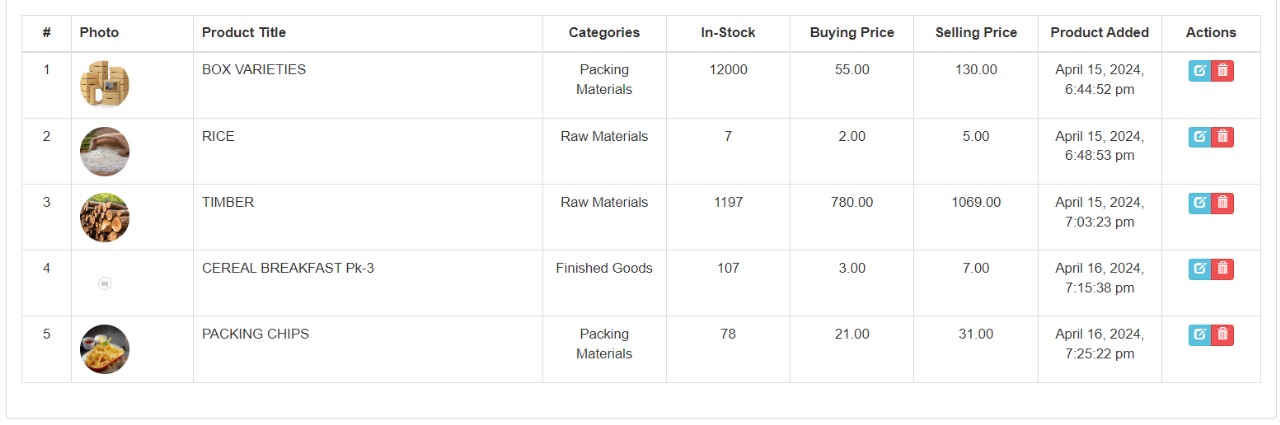


Function that updates image of product upon insertion of a media file

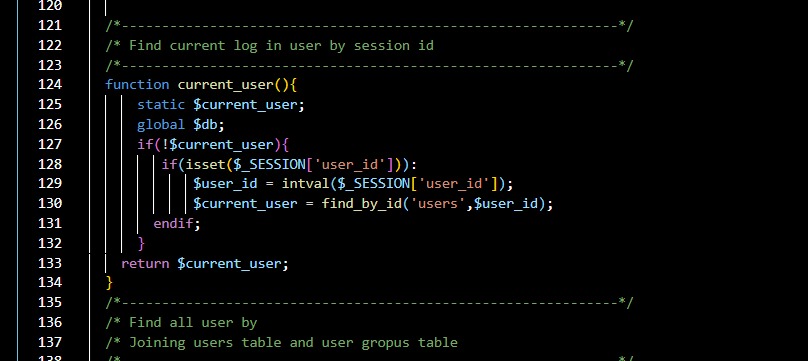


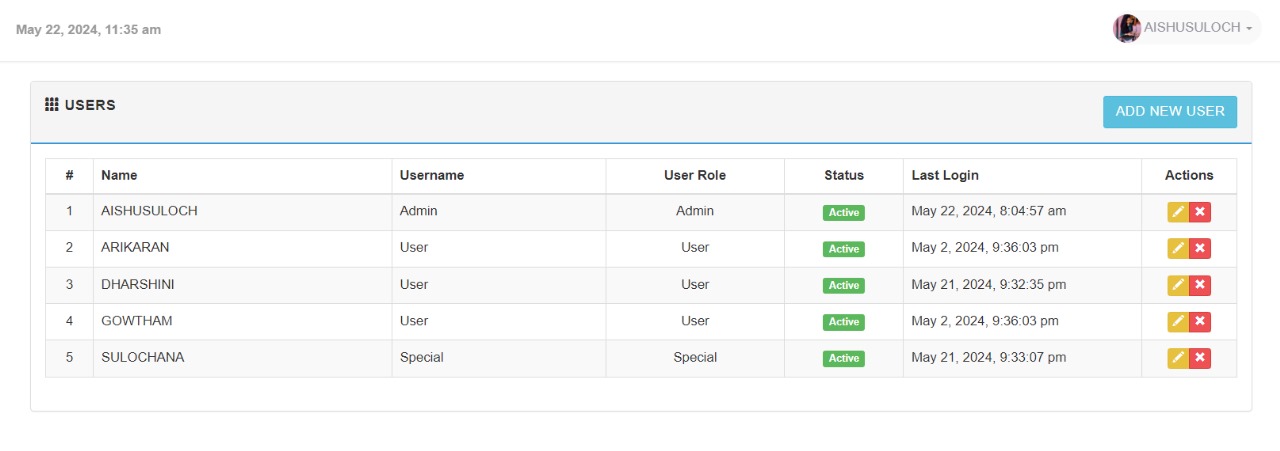
Images:



After updation:

Function that displays the log in time using session:



Output:

**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:

* Implicit cursors
* 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.

Syntax:



**CURSORS CREATED:**

Cursor to retrieve the most sold product from the sales table

Most sold product

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;

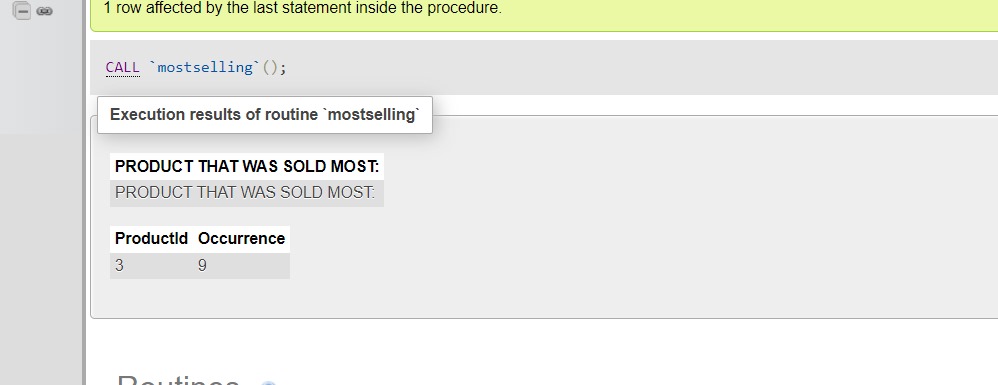
SELECT a AS ProductId, d AS Occurrence;

SET first\_occurrence\_printed = TRUE;

END LOOP read\_loop;

CLOSE c;

END

Output:

Cursor to display the costliest sold product from the sales table:

sales report

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;

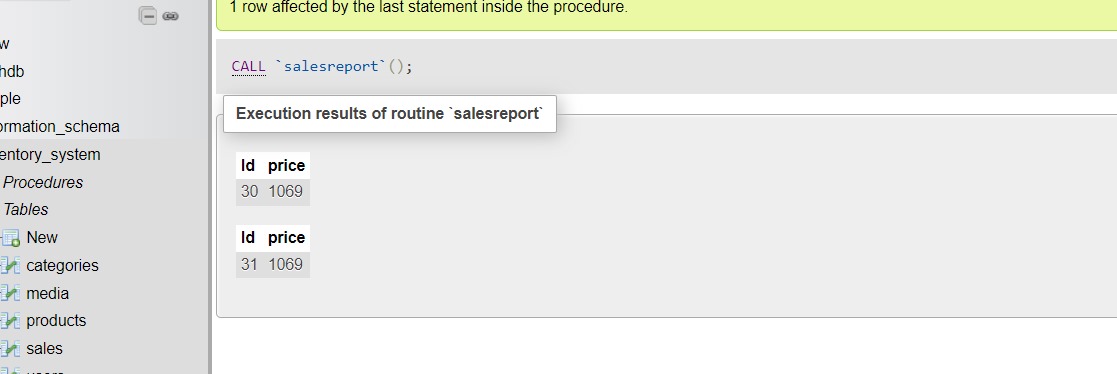
SELECT a as Id,d as price;

END LOOP read\_loop;

CLOSE c;

END

Output:



**VIEWS**:

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

Query:

Create view v2 as

select products.name, media.file\_name

from media, products

where media.id= products. media\_id;

Output:

