DBMS - Mini Project Title of the Project

**Submitted by:**

**Name –** Yash Vardhan Soni

**SRN –** PES1UG20CS518

**V Semester**

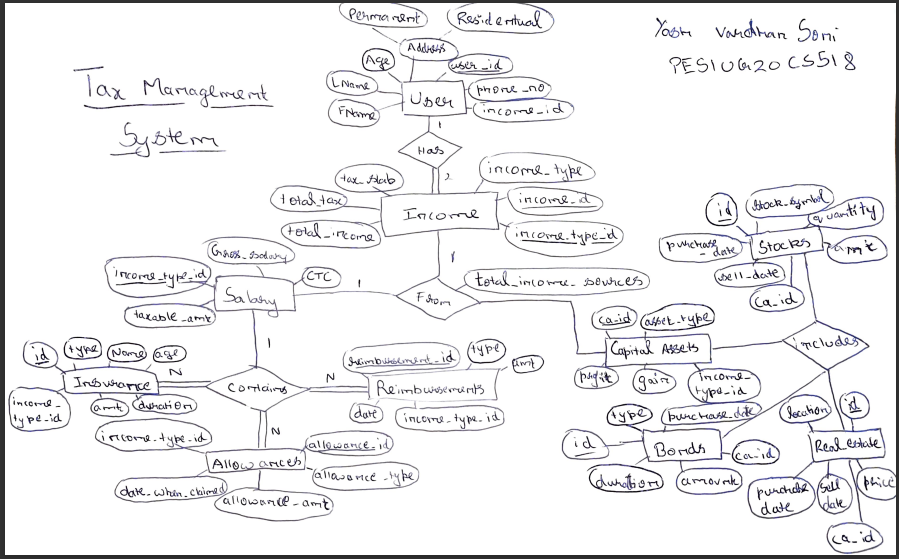
**Section -** I

# Short Description and Scope of the Project

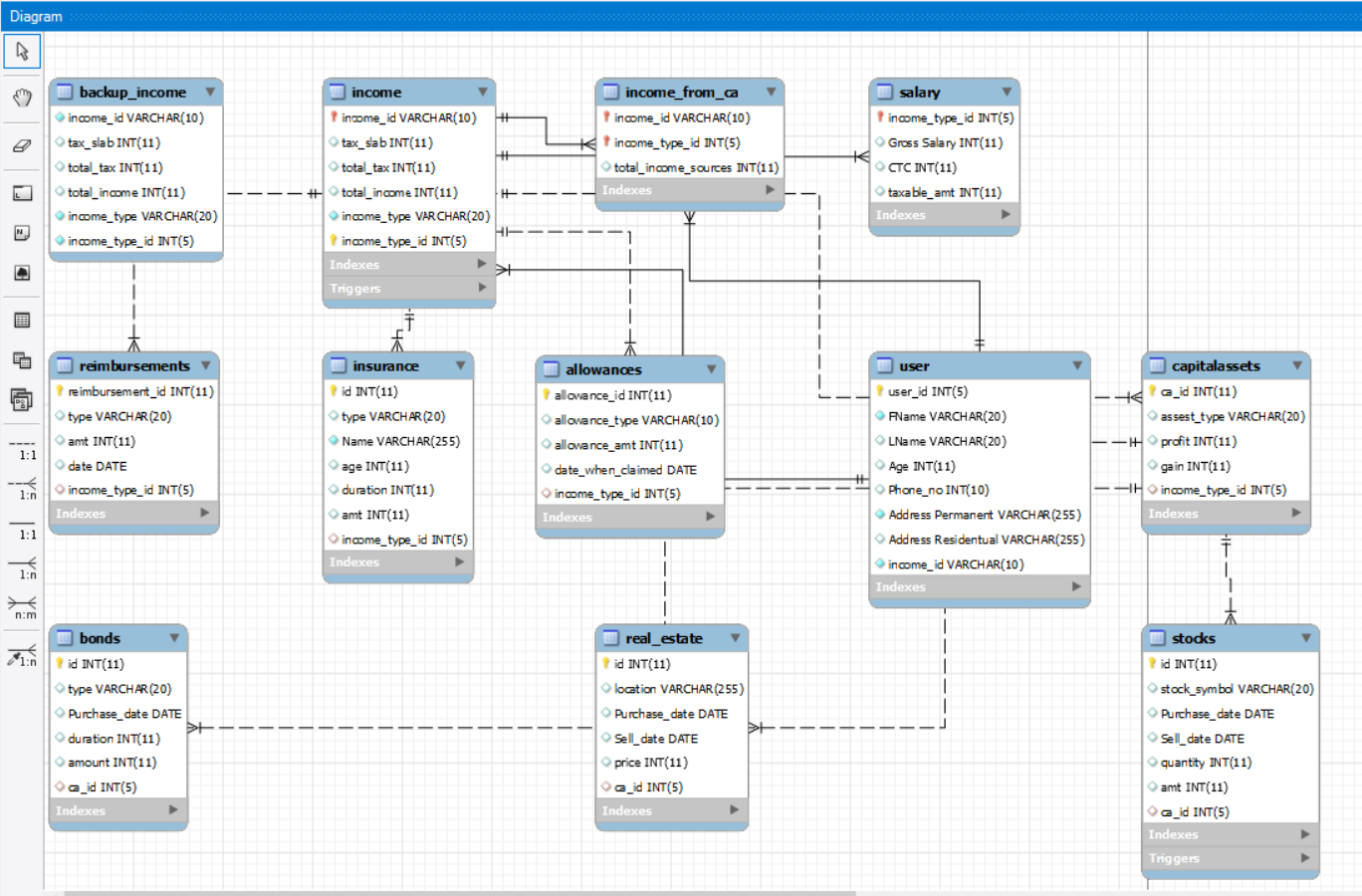
**Tax Management System** is a relational database system that stores the income details of a person from multiple sources like salary, capital assets, etc. Salary of a person has certain benefits like allowances, reimbursements and insurance schemes which can be exempted from taxes. These are provided to every salaried employee and with the right knowledge they can help in minimizing the tax that must be paid. Income from other sources such as real estate, stocks and bonds is also stored in this system. The total tax that has to be paid is calculated and updated in the table as soon as an entry in made. There are different tax slabs for different incomes and the tax is calculated accordingly.

**SCOPE:** It is really important to calculate one’s taxes and file their ITR. Calculating this can be a tedious task considering the user can have income from multiple sources. It is necessary to keep a database which can store this information and automate the calculation for total tax. This can also be used to find the most optimum pay structure for a salaried employee since there are multiple types of exemptions and deductions that can be applied. This makes the database slightly complex and is included in the future scope of the project. Extending this to a financials management is a long term goal since it would make sense to keep track of dividend received from savings account and employee PPF. This can also be extended for companies as the owner will then also be able to show his profits/losses from business.

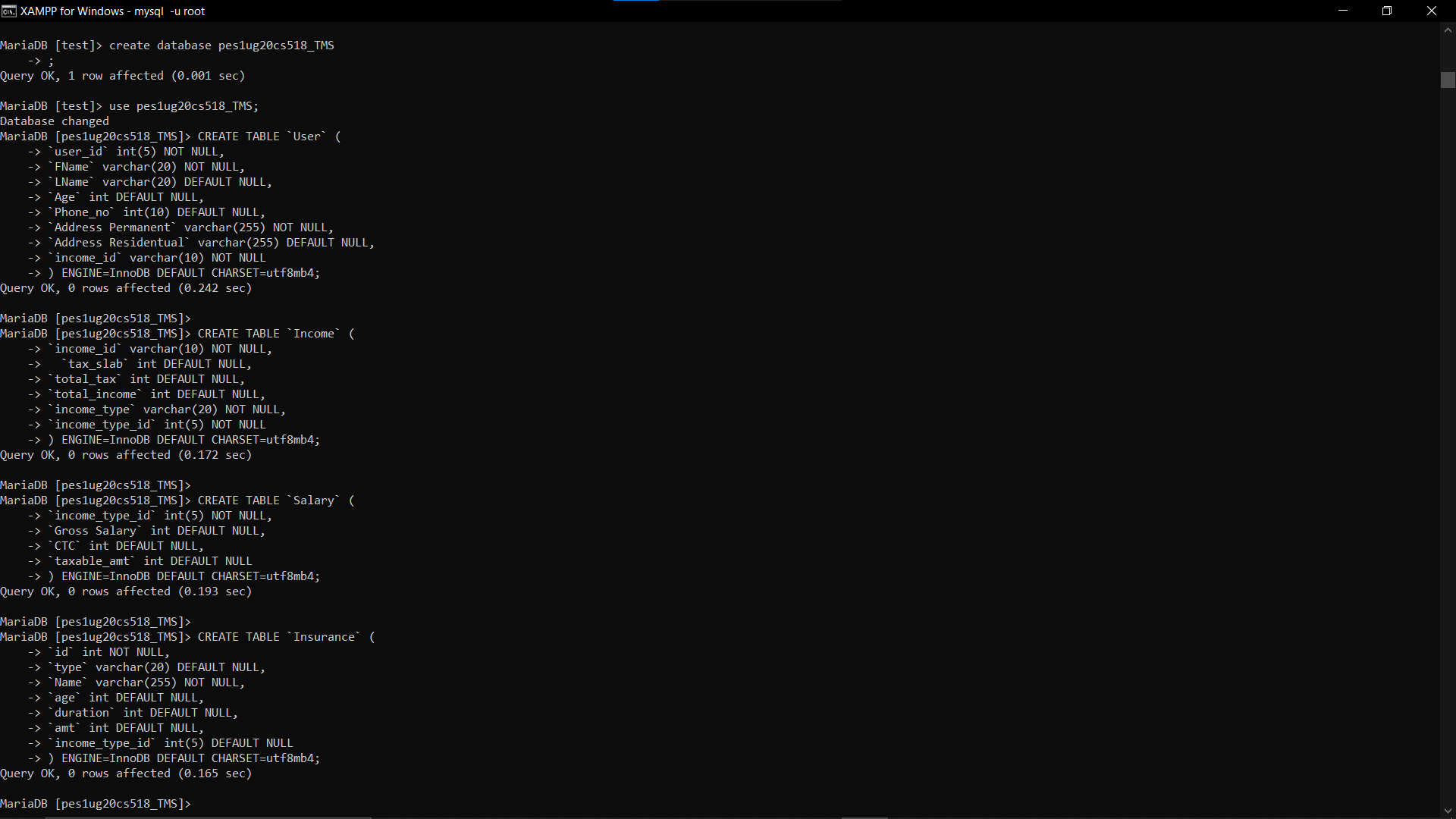
**ER Diagram**

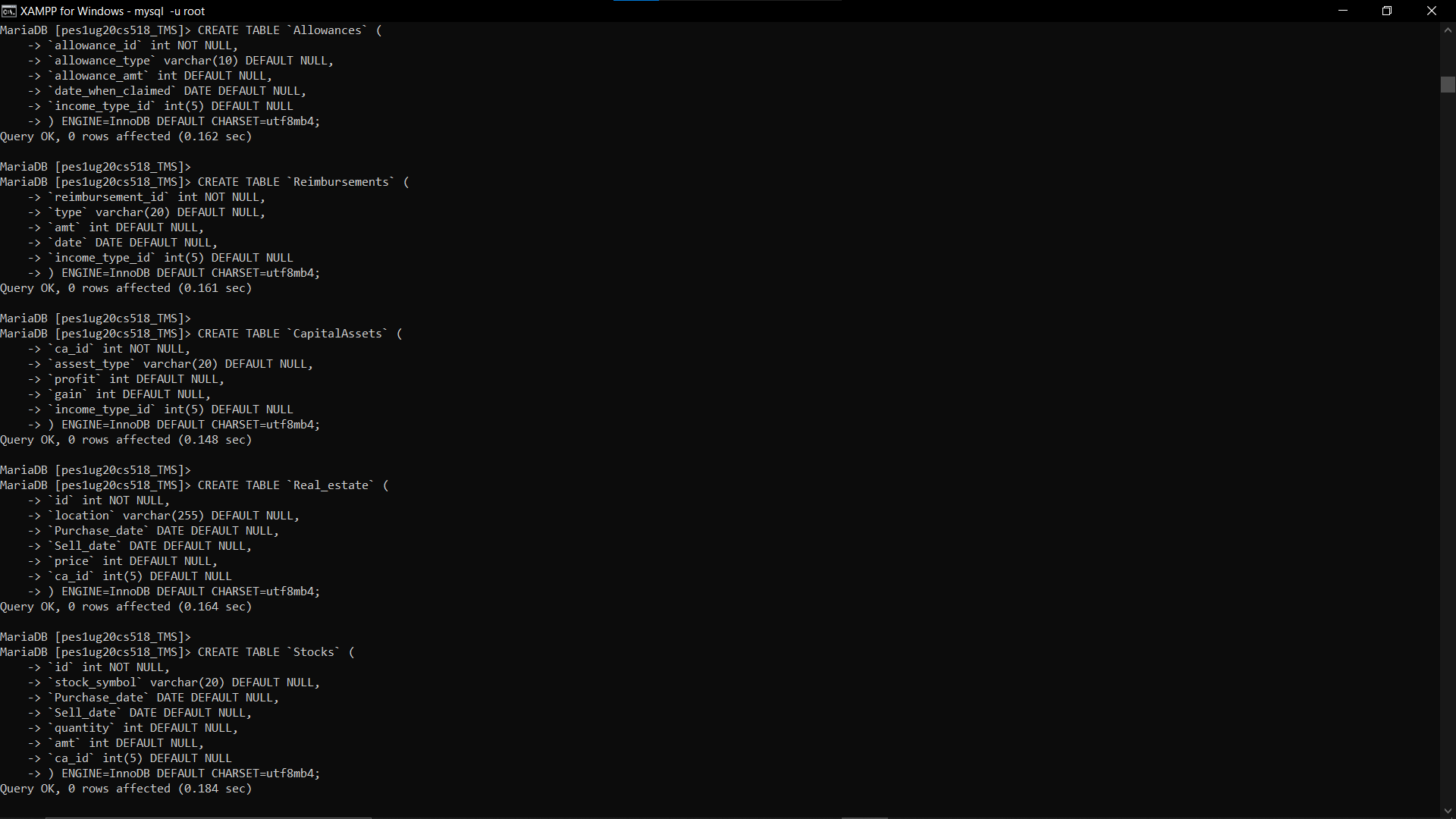


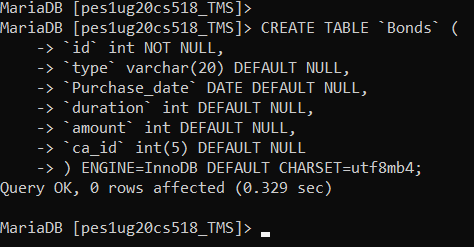
# Relational Schema

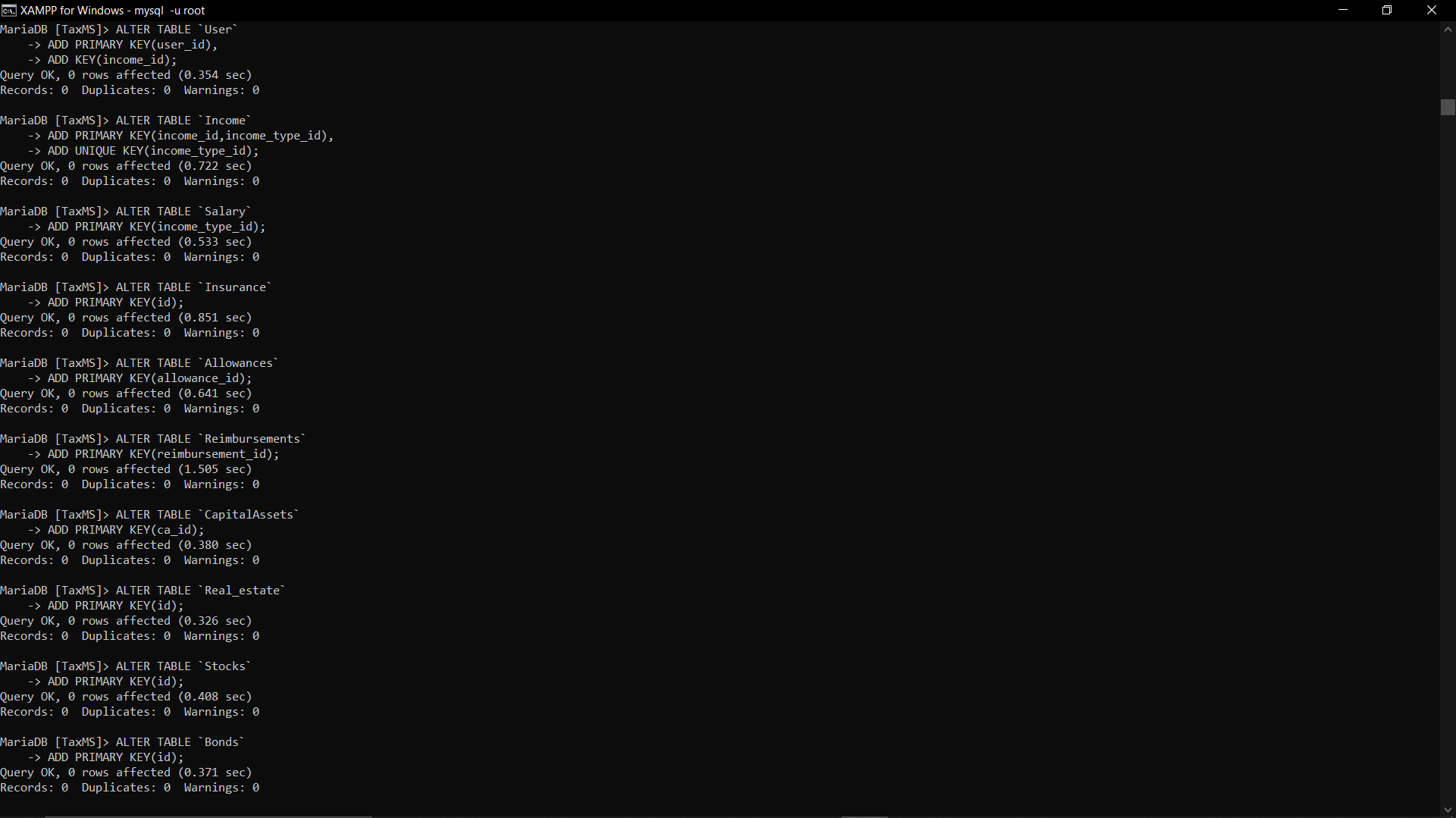


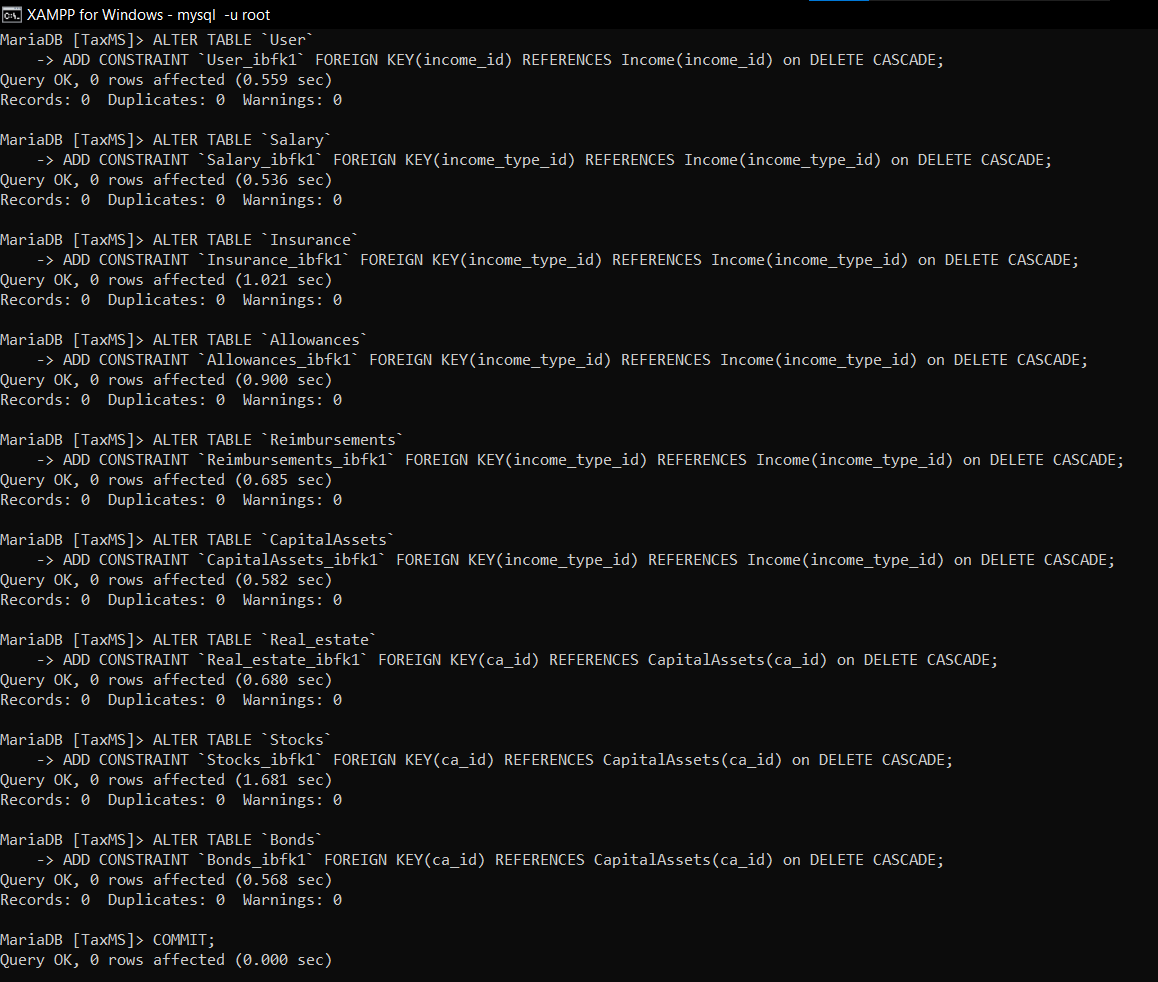
**DDL statements - Building the database**

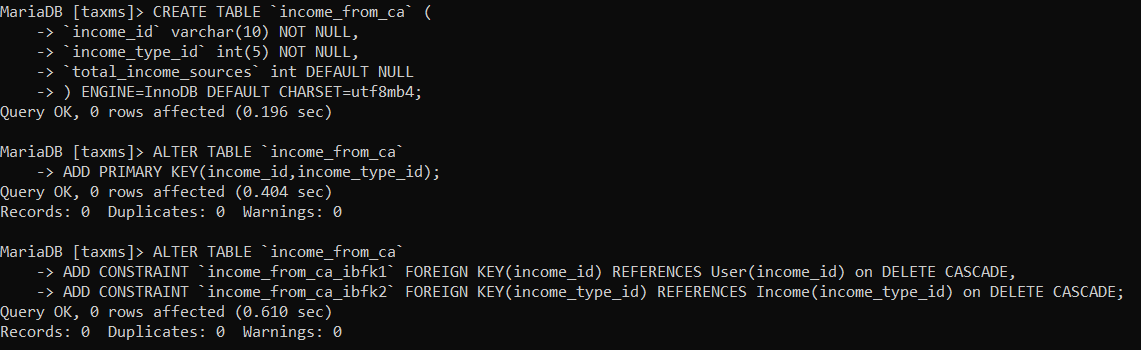




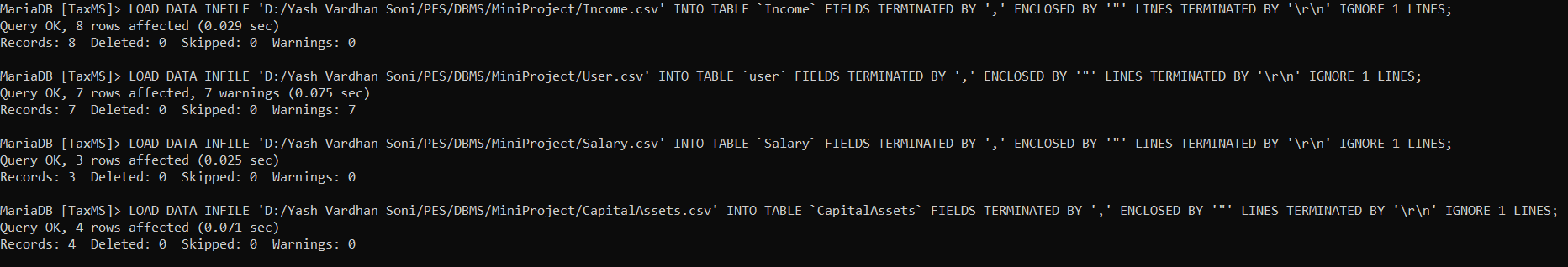


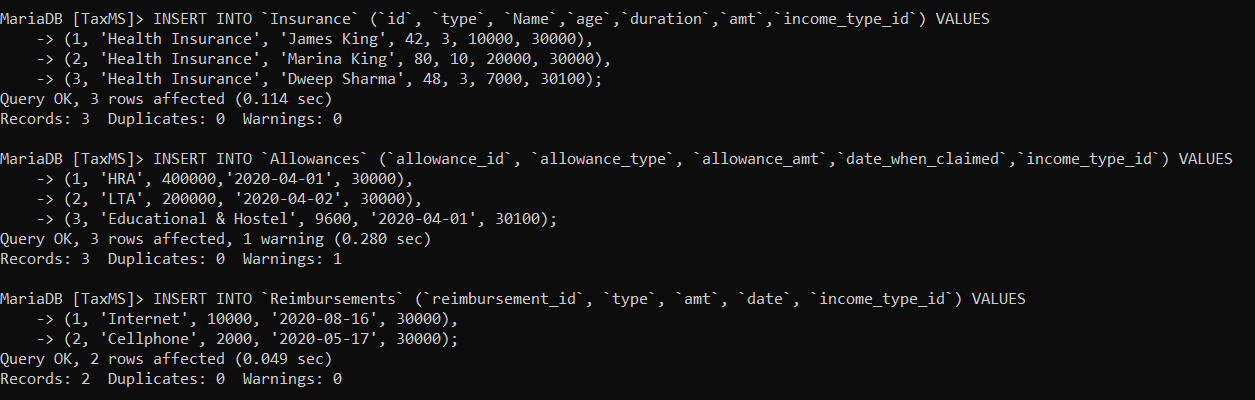


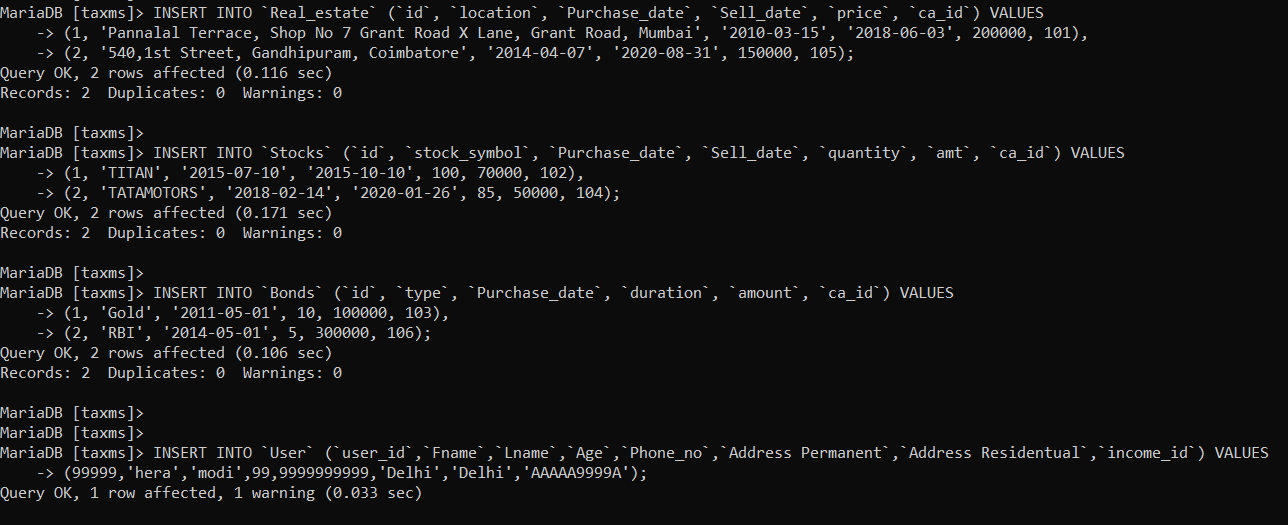


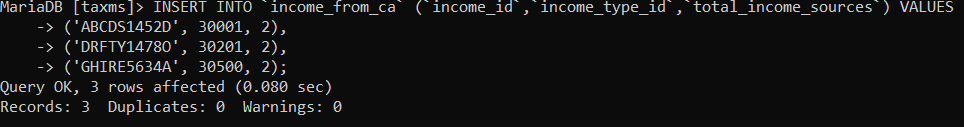


# Populating the Database









**Join Queries**

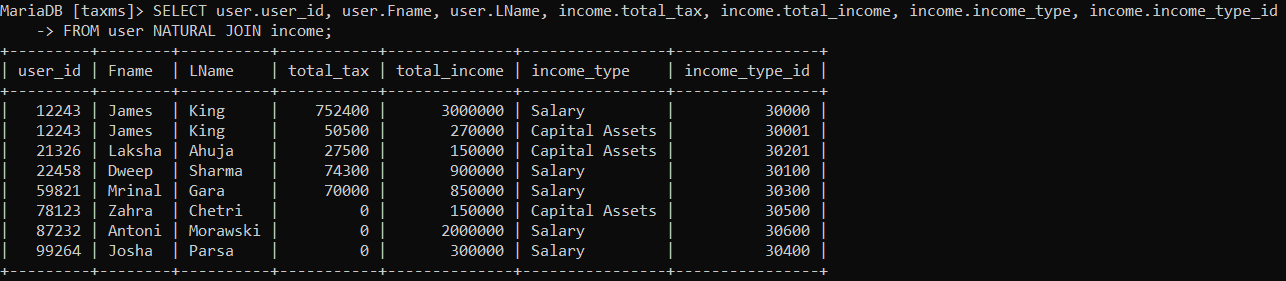
Showcase at least 4 join queries

Write the query in English Language, Show the equivalent SQL statement and also a screenshot of the query and the results

1. User natural join income to find the name of the user with his income details

SELECT user.user\_id, user.Fname, user.LName, income.total\_tax, income.total\_income, income.income\_type, income.income\_type\_id

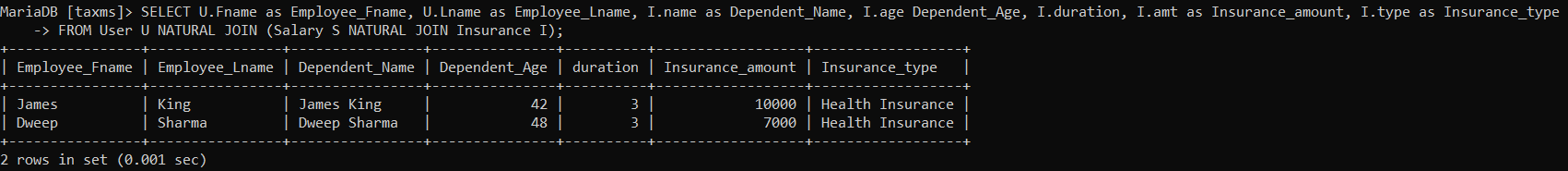
FROM user NATURAL JOIN income;



1. To find employees that have insurances and their dependents name

SELECT U.Fname as Employee\_Fname, U.Lname as Employee\_Lname, I.name as Dependent\_Name, I.age Dependent\_Age, I.duration, I.amt as Insurance\_amount, I.type as Insurance\_type

FROM User U NATURAL JOIN (Salary S NATURAL JOIN Insurance I);

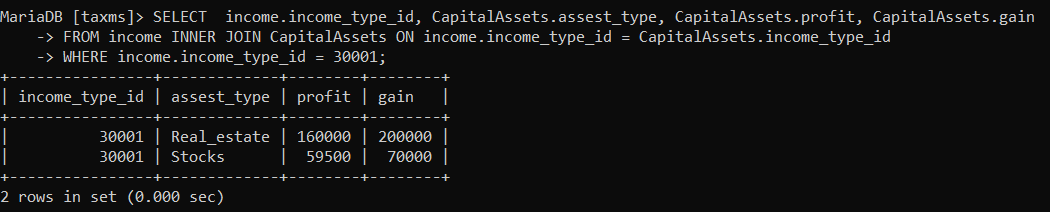


1. To find those capital assets for user that has income id 3001

SELECT  income.income\_type\_id, CapitalAssets.assest\_type, CapitalAssets.profit, CapitalAssets.gain

FROM income INNER JOIN CapitalAssets ON income.income\_type\_id = CapitalAssets.income\_type\_id

WHERE income.income\_type\_id = 30001;

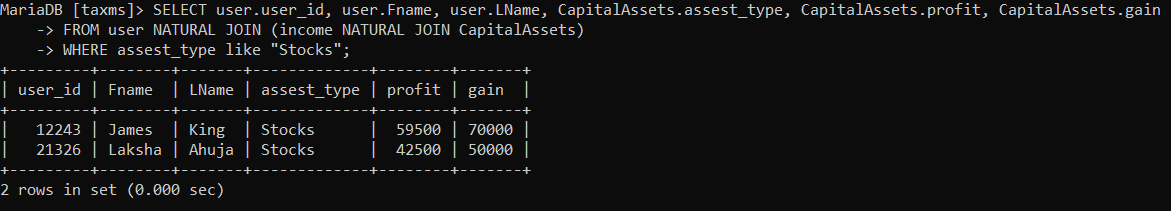


1. To find the user who has invested in stocks and the stocks he has invested in.

SELECT user.user\_id, user.Fname, user.LName, CapitalAssets.assest\_type, CapitalAssets.profit, CapitalAssets.gain

FROM user NATURAL JOIN (income NATURAL JOIN CapitalAssets)

WHERE assest\_type like "Stocks";



# Aggregate Functions

Showcase at least 4 Aggregate function queries

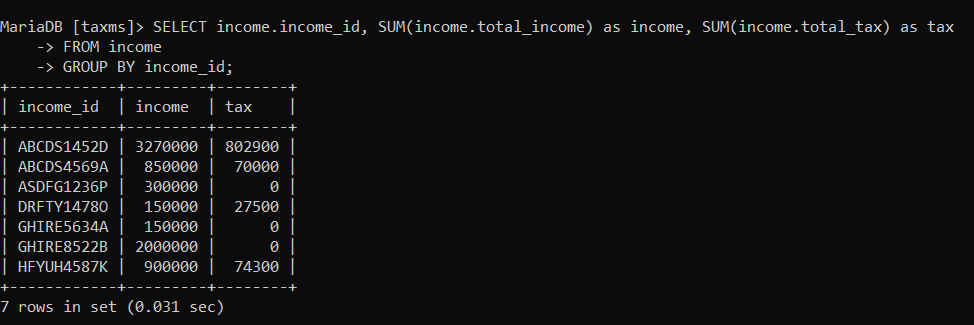
Write the query in English Language, Show the equivalent SQL statement and also a screenshot of the query and the results

1. To find the total income and total tax paid for each income id

SELECT income.income\_id, SUM(income.total\_income) as income, SUM(income.total\_tax) as tax

FROM income

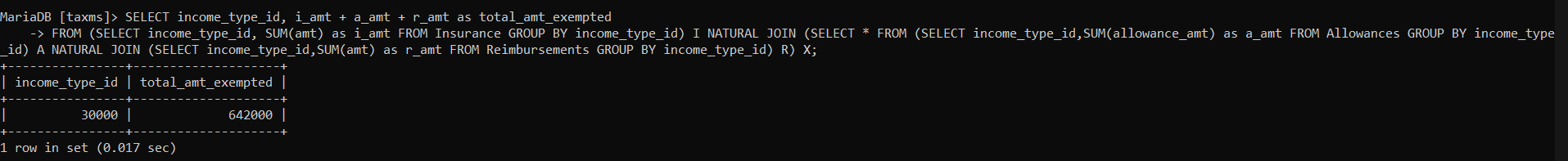
GROUP BY income\_id;



1. To find the total amount exempted for tax for a given income id.

SELECT income\_type\_id, i\_amt + a\_amt + r\_amt as total\_amt\_exempted

FROM (SELECT income\_type\_id, SUM(amt) as i\_amt FROM Insurance GROUP BY income\_type\_id) I NATURAL JOIN (SELECT \* FROM (SELECT income\_type\_id,SUM(allowance\_amt) as a\_amt FROM Allowances GROUP BY income\_type\_id) A NATURAL JOIN (SELECT income\_type\_id,SUM(amt) as r\_amt FROM Reimbursements GROUP BY income\_type\_id) R) X;

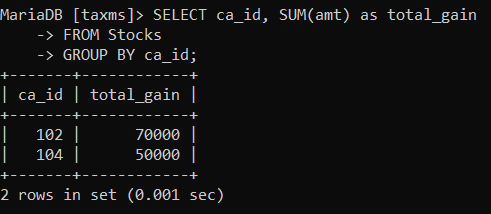


1. To find the total gain each person has made from stocks.

SELECT ca\_id, SUM(amt) as total\_gain

FROM Stocks

GROUP BY ca\_id;

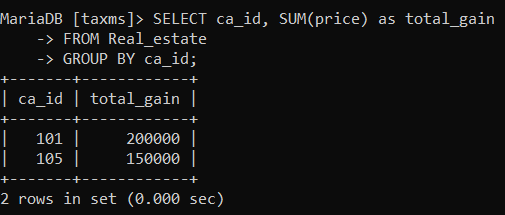


1. To find the total gain made by each person from selling real estates.

SELECT ca\_id, SUM(price) as total\_gain

FROM Real\_estate

GROUP BY ca\_id;



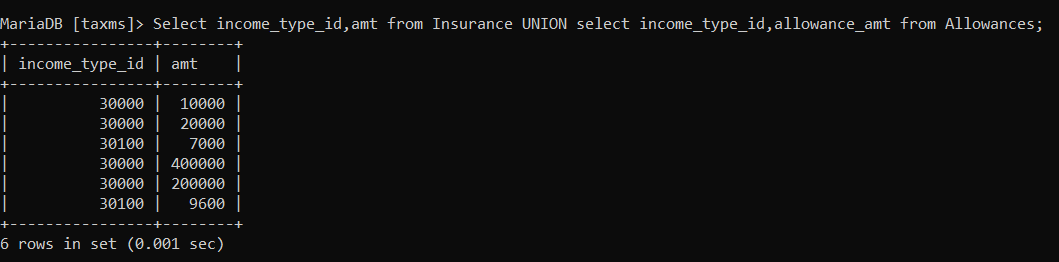
# Set Operations

Showcase at least 4 Set Operations queries

Write the query in English Language, Show the equivalent SQL statement and also a screenshot of the query and the results

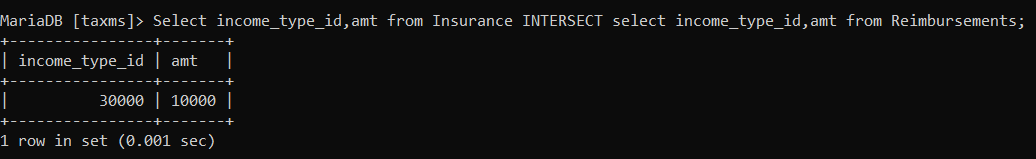
1. To find users who are getting tax exempted from either insurance or allowance

Select income\_type\_id,amt from Insurance UNION select income\_type\_id,allowance\_amt from Allowances;



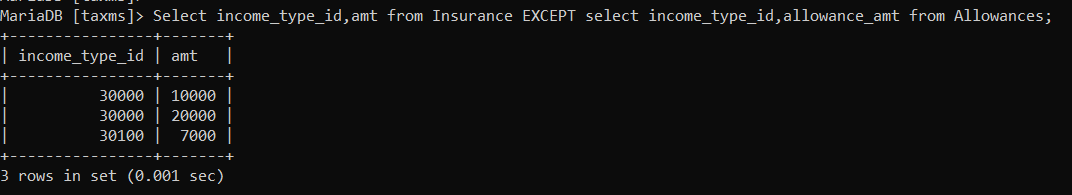
1. To find users who have applied for both insurance and reimbursements

Select income\_type\_id,amt from Insurance INTERSECT select income\_type\_id,amt from Reimbursements;



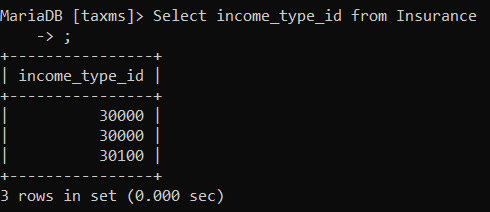
1. To find Users who have insurances but have not applied for allowances

Select income\_type\_id,amt from Insurance EXCEPT select income\_type\_id,allowance\_amt from Allowances;



1. To find those income type ids that are not in capital assets

Select income\_type\_id from Insurance EXCEPT (select distinct income\_type\_id from CapitalAssets);



# Functions and Procedures

Create a Function and Procedure. State the objective of the function / Procedure. Run and display the results.

**PROCEDURE :**To update the total tax for capital assets type of income which takes tax caluculated on total income from capital assets.

DELIMITER $$

CREATE PROCEDURE update\_total\_tax(IN id VARCHAR(30), IN type\_id int, OUT msg VARCHAR(50))

BEGIN

    SET @inc\_type = "";

    SELECT income\_type INTO @inc\_type

    FROM income

    WHERE income.income\_id like id AND income.income\_type\_id = type\_id;

    IF @inc\_type like "Capital Assets" THEN

        SET @after\_tax = 0;

        SET @before\_tax = 0;

        SELECT SUM(CapitalAssets.profit) INTO @after\_tax

        FROM income NATURAL JOIN CapitalAssets

        WHERE income.income\_type\_id = type\_id

        GROUP BY income.income\_type\_id;

        SELECT SUM(CapitalAssets.gain) INTO @before\_tax

        FROM income NATURAL JOIN CapitalAssets

        WHERE income.income\_type\_id = type\_id

        GROUP BY income.income\_type\_id;

        UPDATE income

        SET total\_tax = @before\_tax - @after\_tax

        WHERE income.income\_id = id AND income.income\_type\_id = type\_id;

        SET msg = 'Total tax has been updated.';

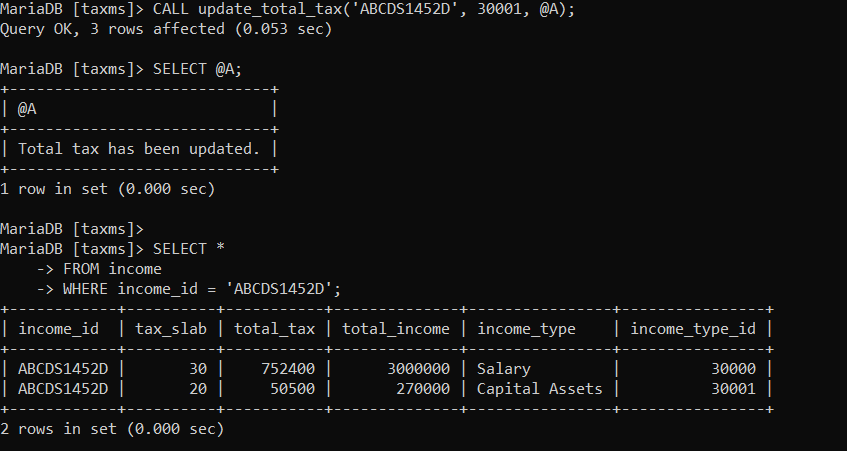
    ELSE

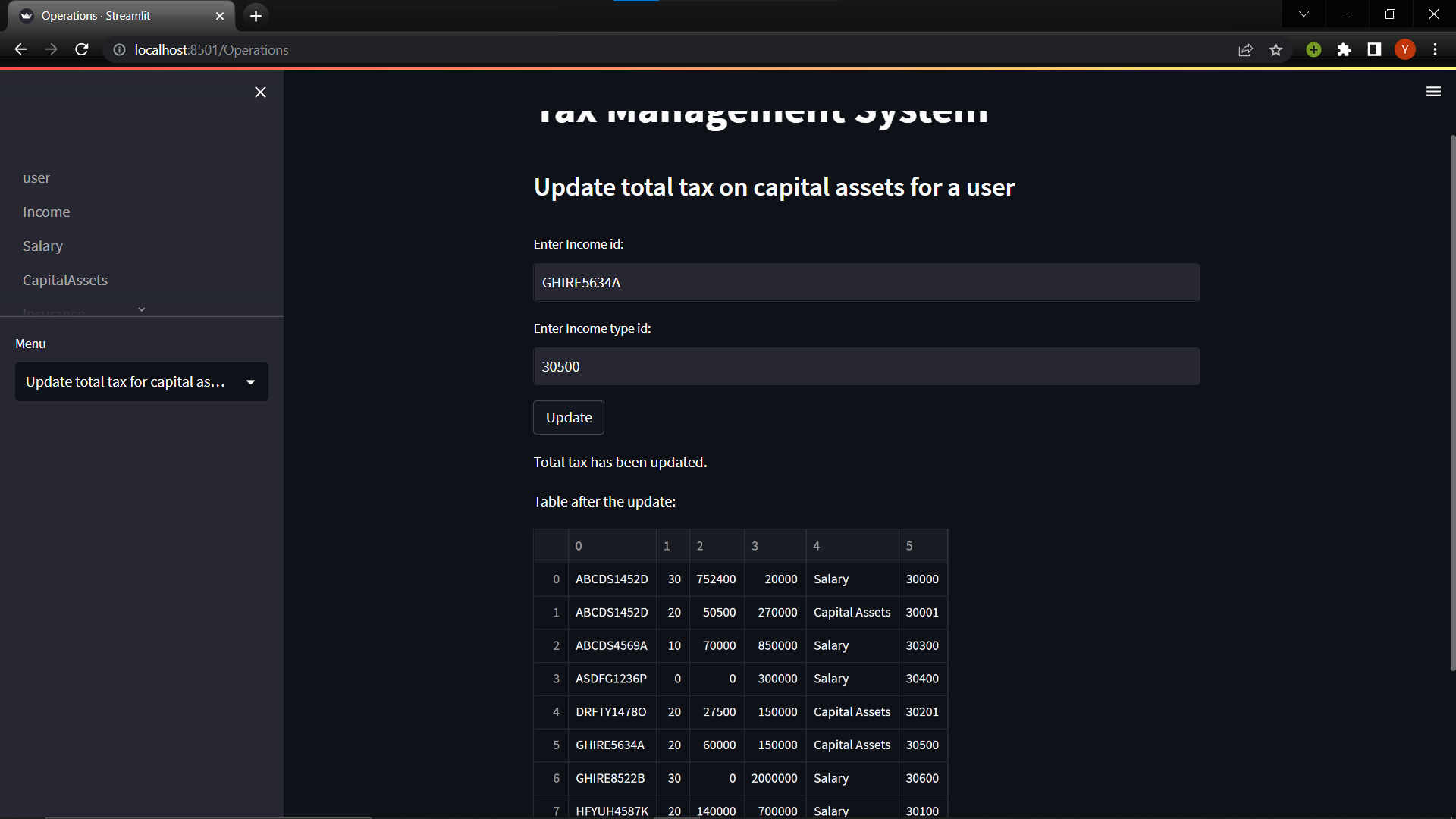
        SET msg = 'No functionality for Salary type of income!' ;

    END IF;

END; $$

DELIMITER ;





**FUNCTION:**To find the total number of assets per user

DELIMITER $$

CREATE FUNCTION no\_of\_assets(type\_id INT)

RETURNS INT

DETERMINISTIC

BEGIN

    DECLARE n int default 0;

    SELECT COUNT(\*) into n

    FROM CapitalAssets

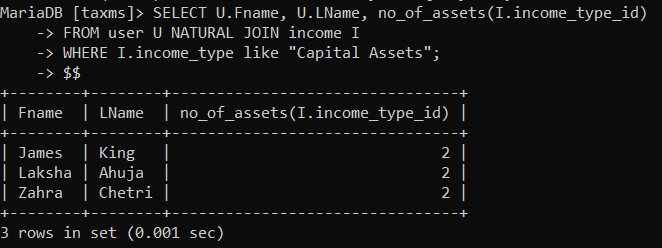
    WHERE CapitalAssets.income\_type\_id = type\_id

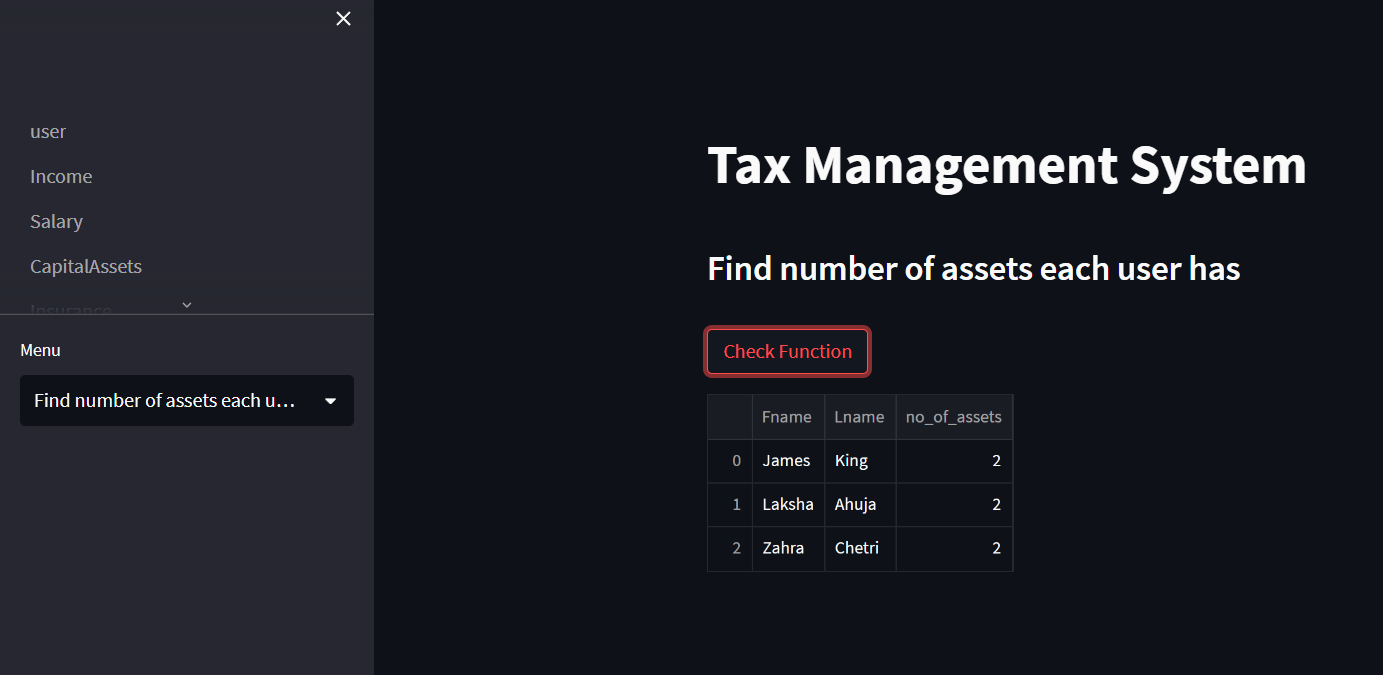
    GROUP BY CapitalAssets.income\_type\_id;

    RETURN n;

END; $$

DELIMITER;





# Triggers and Cursors

Create a Trigger and a Cursor. State the objective. Run and display the results.

**TRIGGER: If the sell date is before purchase date for stocks it gives a warning**

DELIMITER $$

CREATE TRIGGER date\_check

BEFORE INSERT

ON Stocks FOR EACH ROW

BEGIN

    DECLARE err\_msg VARCHAR(255);

    declare count int;

    SET err\_msg = ('Sell Date is before purchase date');

    IF new.Purchase\_date > new.Sell\_date THEN

    SIGNAL SQLSTATE '45000'

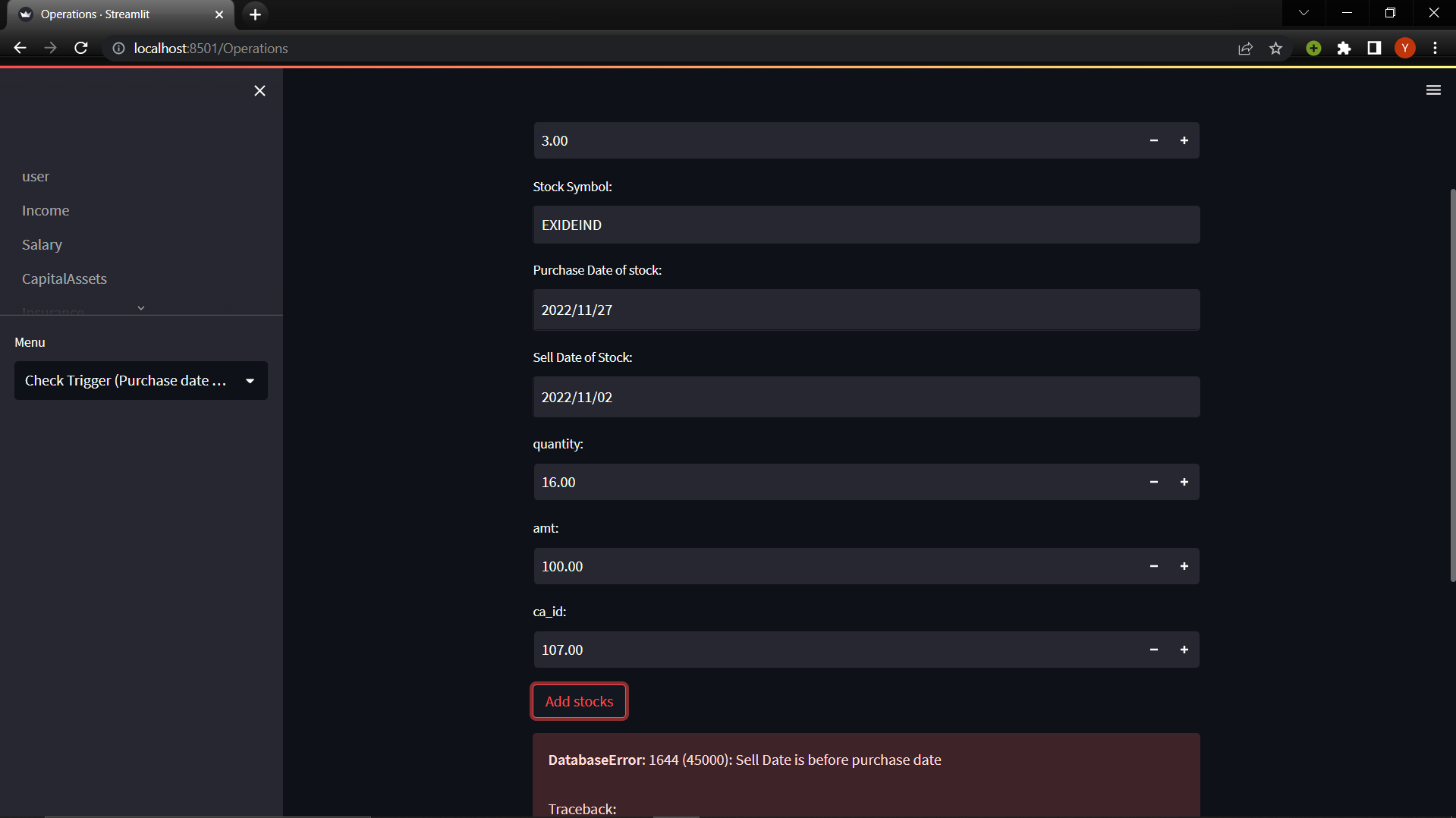
    SET MESSAGE\_TEXT = err\_msg;

    END IF;

END $$

DELIMITER ;





**CURSOR:**

Create a backup table for income .

CREATE table `backup\_income`(

    `income\_id` varchar(10) NOT NULL,

    `tax\_slab` int DEFAULT NULL,

    `total\_tax` int DEFAULT NULL,

    `total\_income` int DEFAULT NULL,

    `income\_type` varchar(20) NOT NULL,

    `income\_type\_id` int(5) NOT NULL

    )ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

DELIMITER //

CREATE PROCEDURE backup()

BEGIN

    DECLARE done INT DEFAULT 0;

    DECLARE income\_id, income\_type varchar(20);

    DECLARE tax\_slab, total\_tax, total\_income, income\_type\_id INTEGER;

    DECLARE cur CURSOR FOR SELECT \* FROM income;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    label: LOOP

    FETCH cur INTO income\_id, tax\_slab, total\_tax, total\_income, income\_type, income\_type\_id;

    INSERT INTO backup\_income VALUES(income\_id, tax\_slab, total\_tax, total\_income, income\_type, income\_type\_id);

    IF done = 1 THEN LEAVE label;

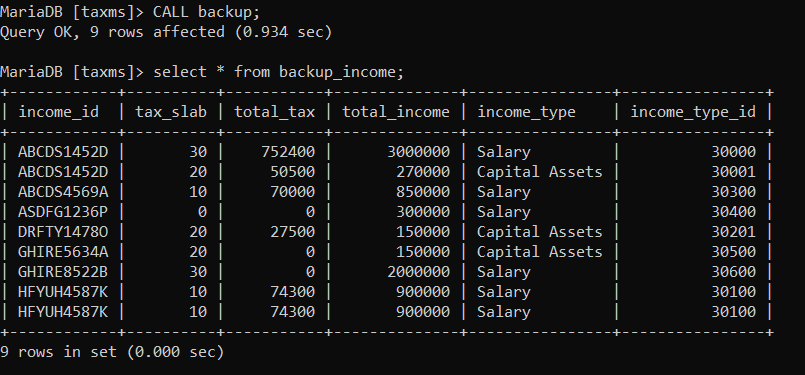
    END IF;

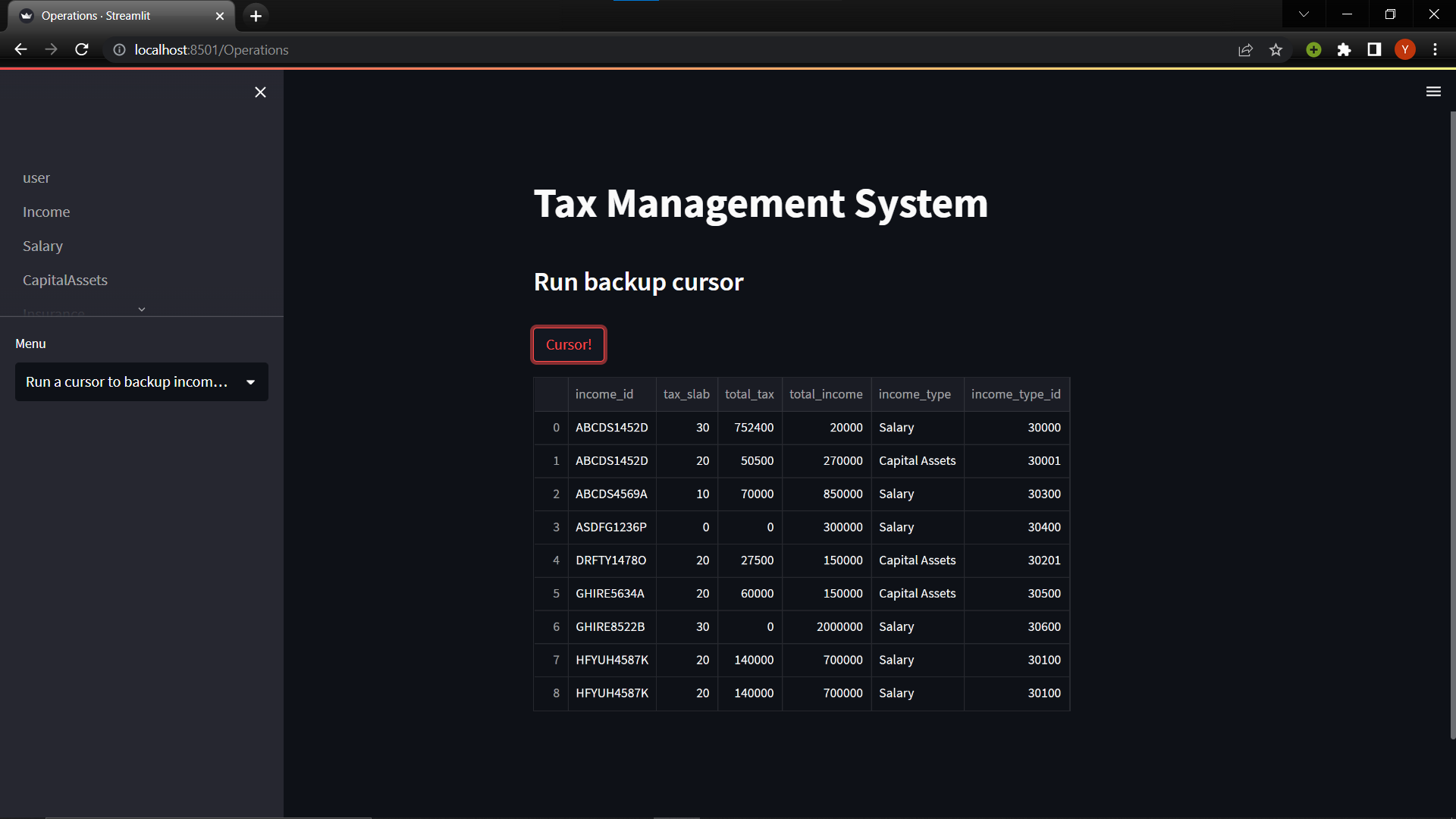
    END LOOP;

    CLOSE cur;

    END//

DELIMITER ;



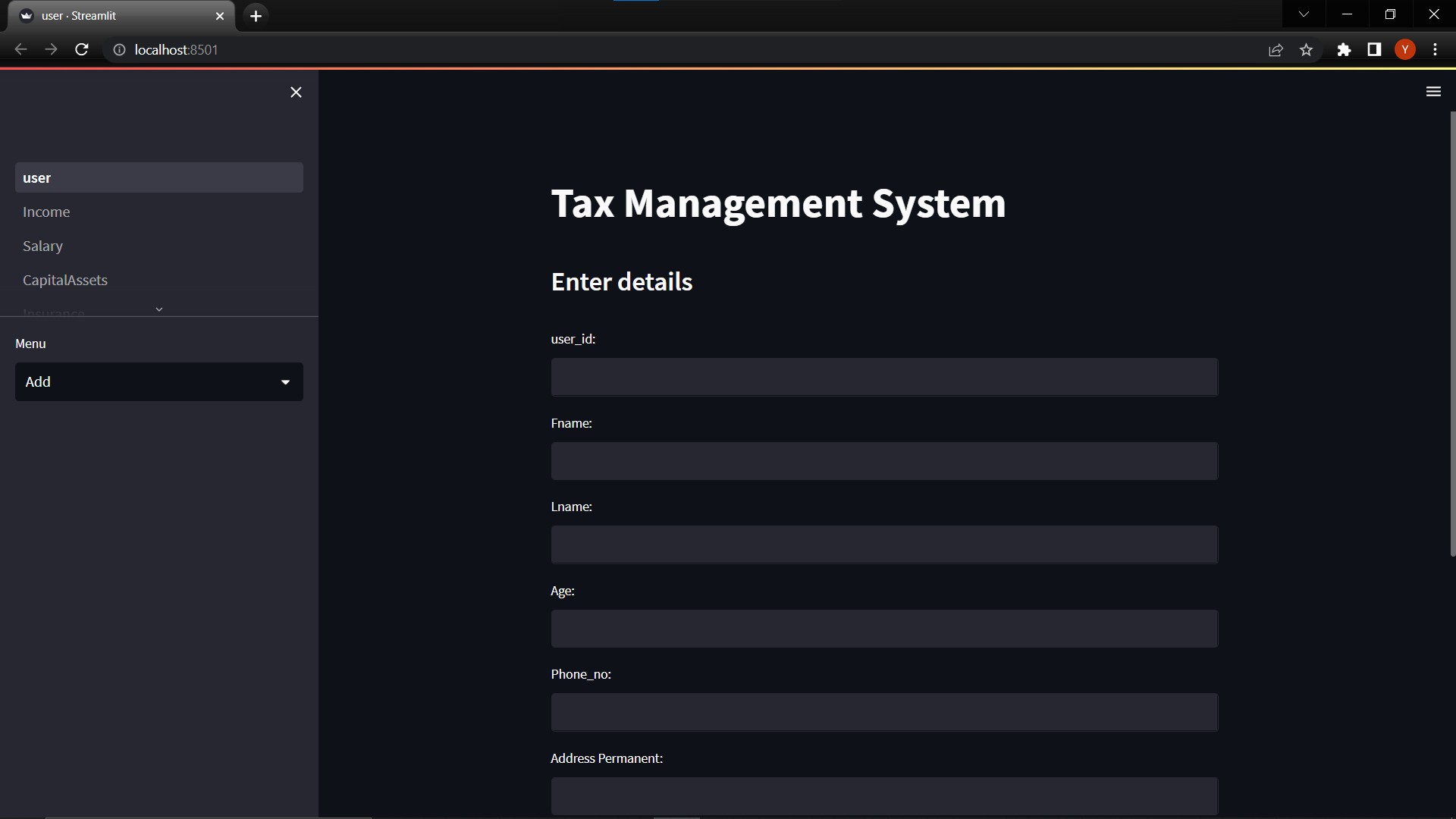


# Developing a Frontend

The frontend should support

1. Addition, Modification and Deletion of records from any chosen table
2. There should be an window to accept and run any SQL statement and display the result

**FRONTEND:**

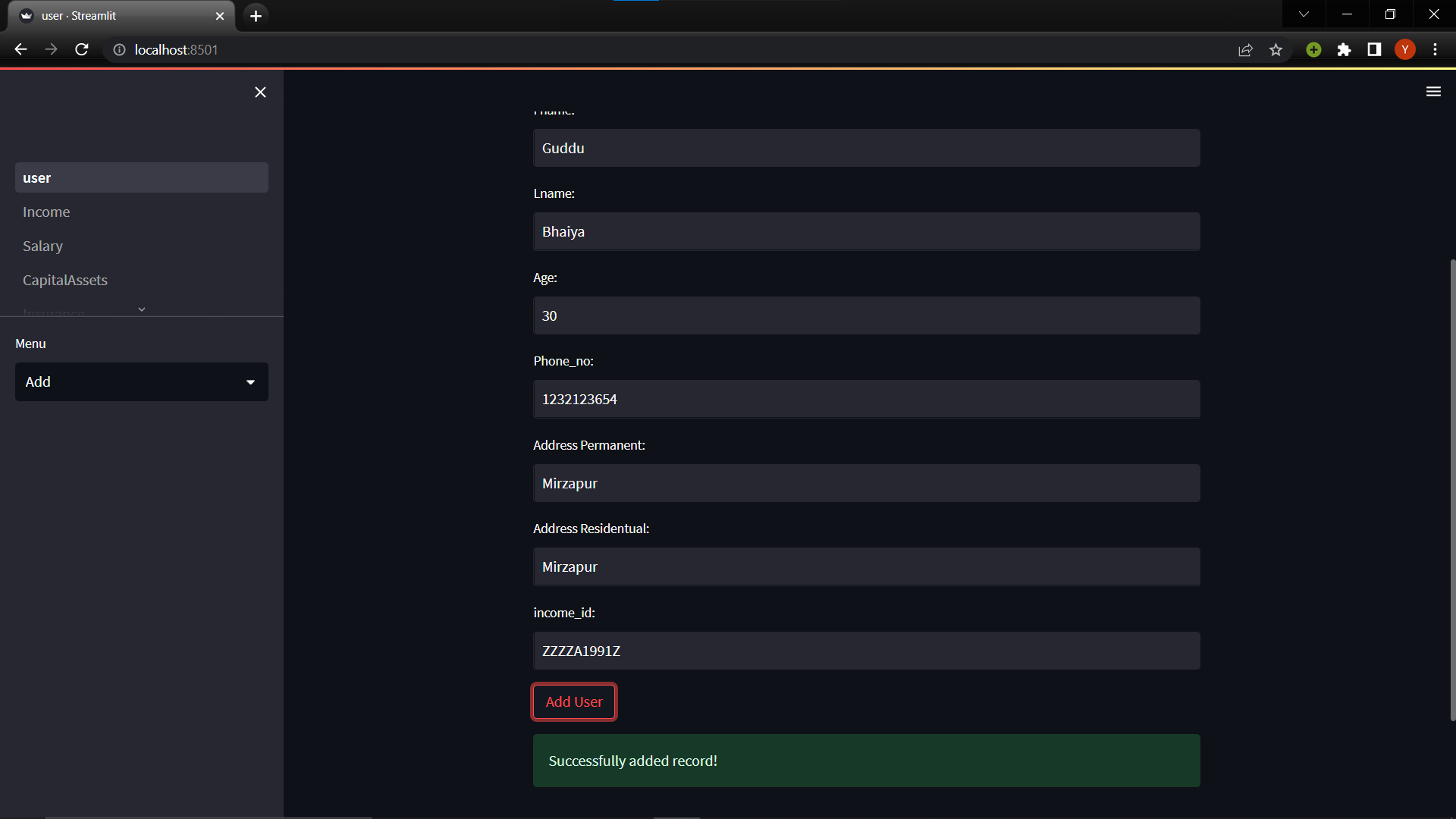


**CRUD OPERATIONS:**

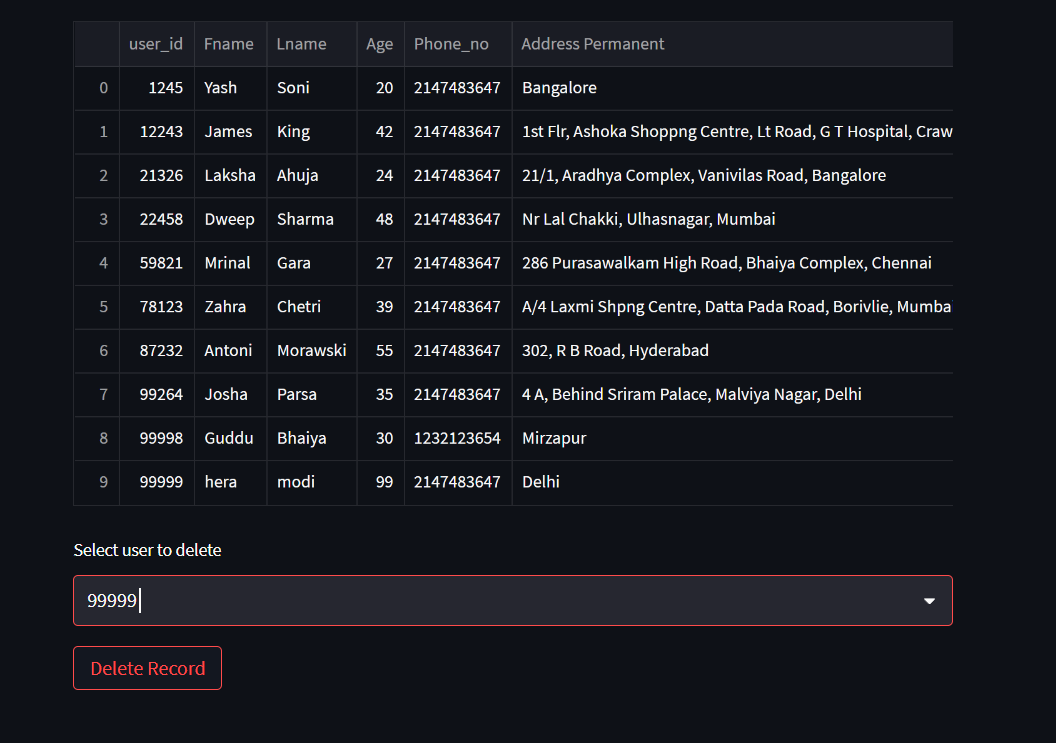
**View –**

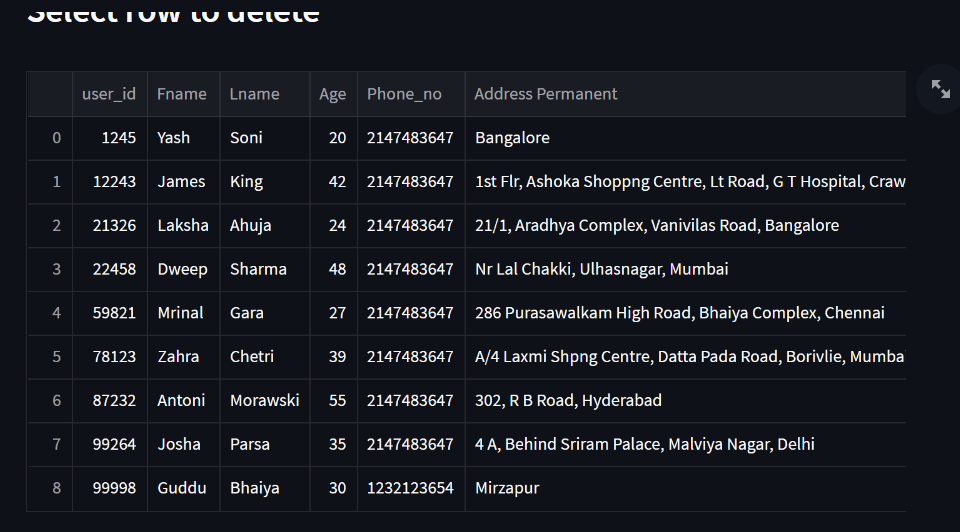
****

**ADD –**

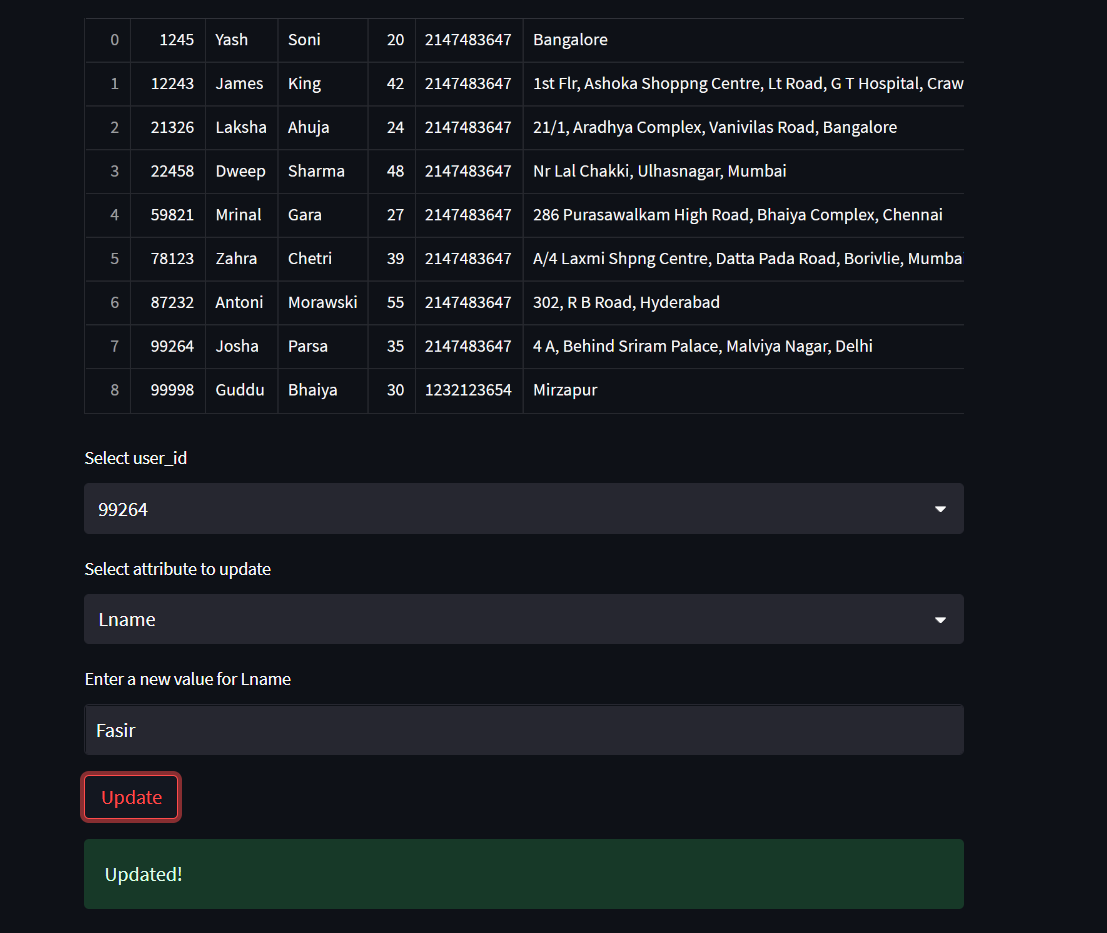
****

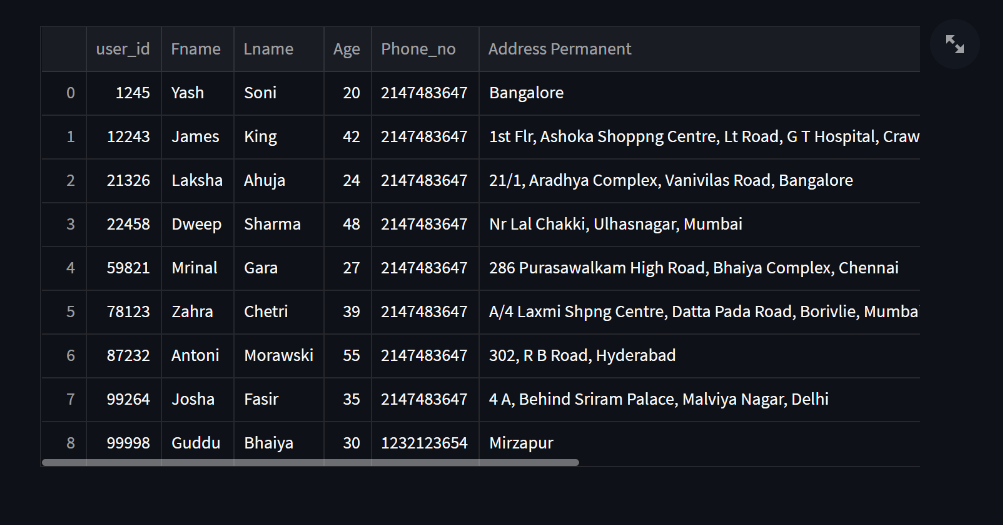
**DELETE –**

****

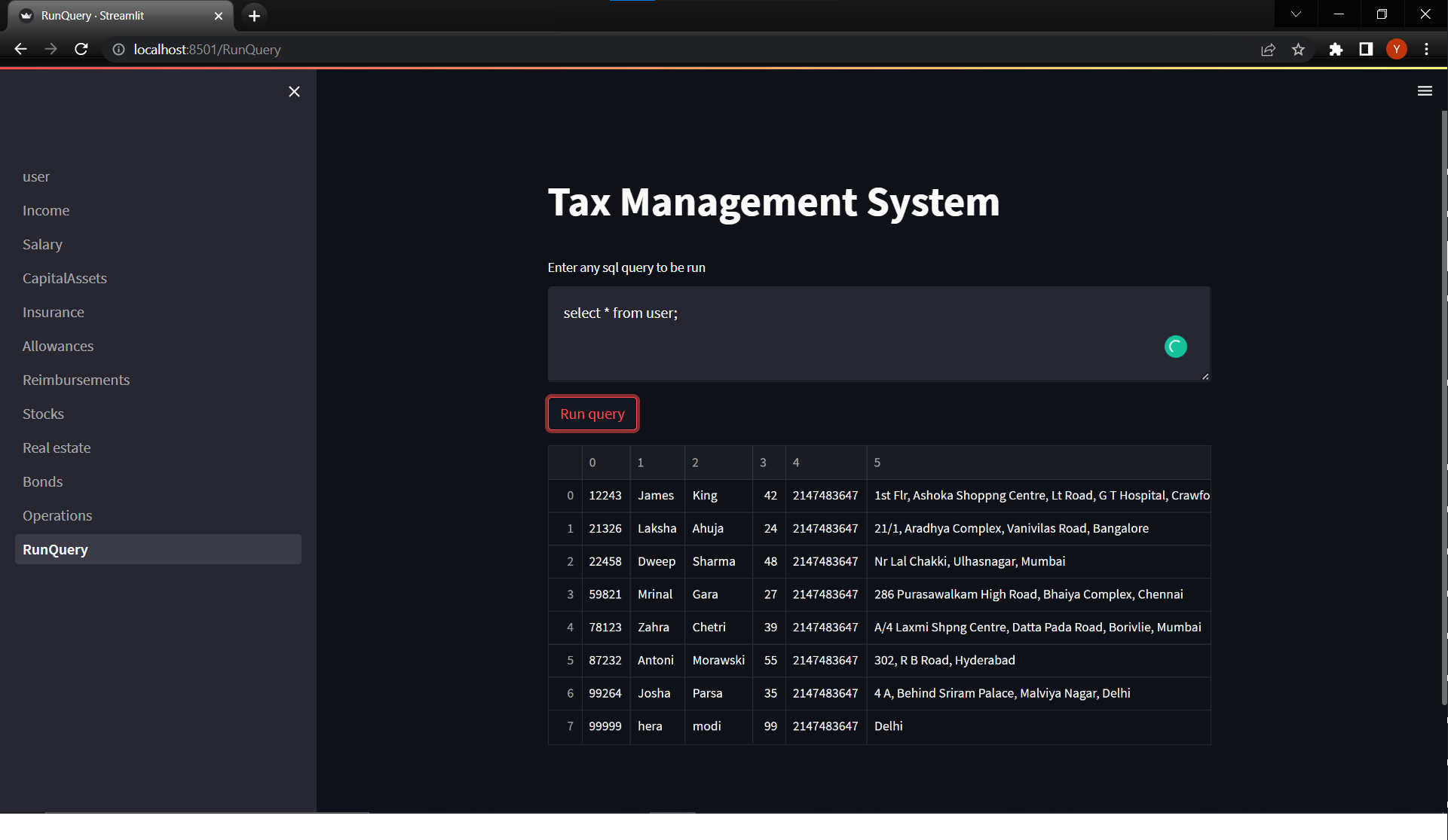
****

**UPDATE –**

****

****

**Run Queries -**



**Modifications During Presentation**

DELIMITER $$

CREATE TRIGGER total\_tax

BEFORE INSERT

ON income FOR EACH ROW

BEGIN

    DECLARE err\_msg VARCHAR(255);

    SET err\_msg = ('Sell Date is before purchase date');

    SET new.total\_tax = 0;

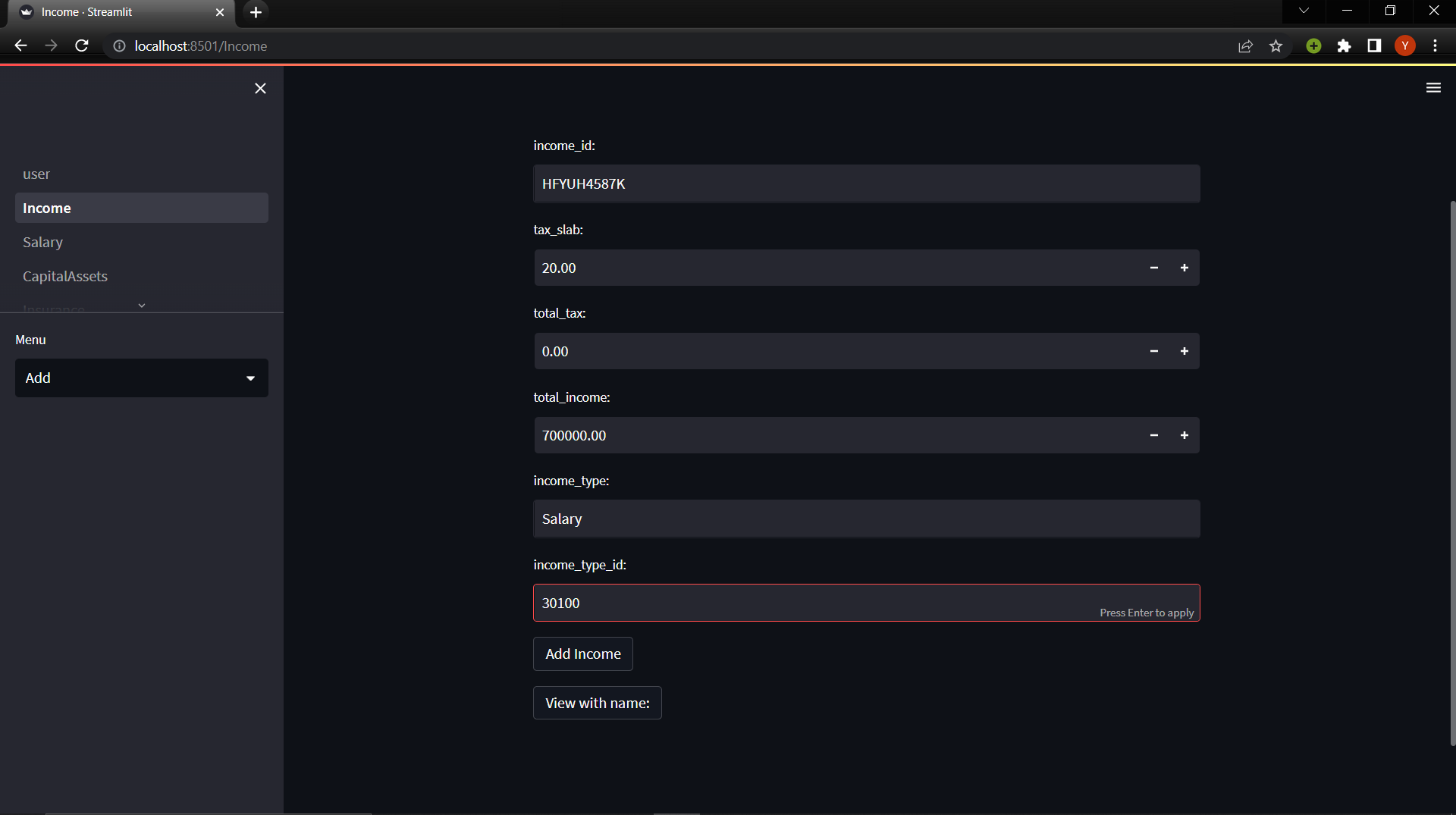
    IF new.total\_income > 500000 THEN

    SET new.total\_tax = 20 \* new.total\_income/100;

    END IF;

END $$

DELIMITER ;

****

****