Course Project

Final project report

Loan collection management

Database Systems

Sumanth Patel Pogaku

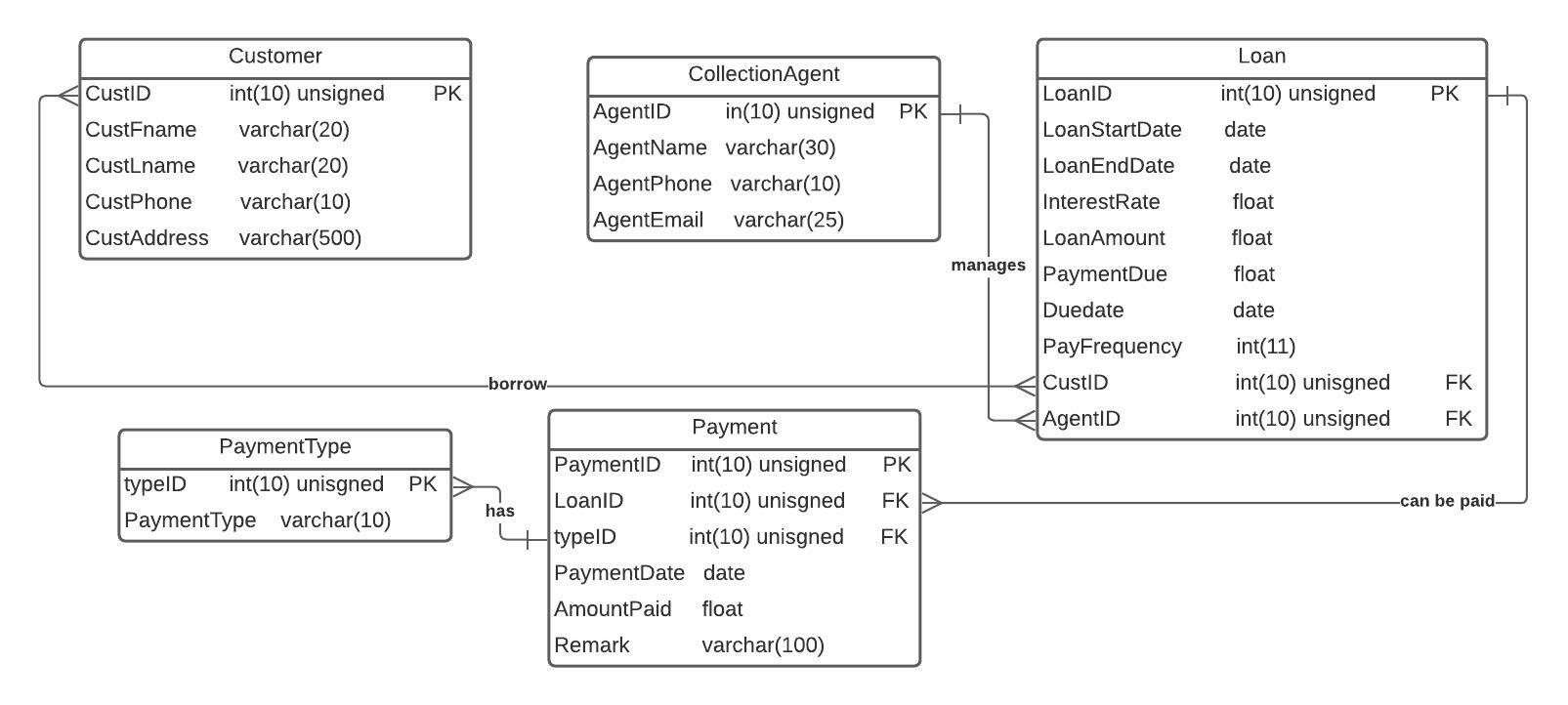
**Initial Proposal**

A customer loan management system can be used to monitor the loans' performance in a simple and effective way. It facilitates tracking of loans and checking whether payment was received on time or whether there was any unpaid interest. The finance company offers loans to the eligible customer with interest and track their payments. Agents keep track of loan transactions and deal with the processes involved in approving loans and collecting money from borrowers. My company’s agent role is to collect the unpaid debts and verify debts are repaid on a specific date and time. I have designed a database that stores data about customers, Collection\_Agents, loans, payments, and payment types. The main purpose of my ADF finance application is to grant loans to the customer and protect my business from payment delays or late payments and to manage in such a way that it will reduce the risk of financial failure. Its purpose is to help track the primary information of customers who have borrowed loans as well as that of the agents who lend the money.

In particular, the database assists in verifying the type of payment they made and the complete loan details such as the date on which the agreement was made and the exact date on which the loan was paid back and the amount of interest due on the payment and the due date, including how much has been paid so far. The data of the customers plays a vital role in tracking the specific customers who borrowed loans and in connecting with them over time. Enter the information of my collection agent who handles payments and loans respectively. My task is to track the loans we have granted with how much interest and check the due date, the date of completion, and the frequency by which we paid the interest, so I need all the loan records in detail. It is necessary for me to know the details of and the date of the payment made for each loan. Customer data is obtained directly from customers who approach me for loans, and the details are stored in a secure way for future reference. To increase the business of my company, my employees and I use the data gathered for business growth by contacting customers over time with offers and discounts offered by our company in order to grow our business and maximize our future growth.

This database files are located at <https://github.com/sumanthpogaku2021/CourseProject>

**Entity relationship diagram**

****

**Relational Database Design Process**

In the database, information about the different entities is stored, along with associations among them. The Customer table, the Collection Agent table, the Loan table, the Payment table, and the Payment Type table are their attributes can be found in a database.

There are five attributes in the customer table: CustID, CustFname, CustLname, CustPhone and CustAddress.

The Collection\_Agent table consists of four attributes: AgentID, AgentName, AgentPhone, and AgentEmail.

The Loan table includes the attributes such as LoanID, LoanStartDate, LoanEndDate, InterestRate, LoanAmount, PaymentDue, Duedate, PayFrequency, CustID, and AgentID.

A Payment table contains six attributes, such as PaymentID, LoanID, typeID, PaymentDate, AmountPaid, and Remark and the Payment Type includes the typeID and PaymentType.

**Customer table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Description** | **Data Type** | **Constraint** |
| CustID | Customer ID | int(10) | Primary key and associates other entities with the customer table. |
| CustFname | First Name of customer | varchar(20) | Not null |
| CustLname | Last name of customer | varchar(20) | Not null |
| CustPhone | Contact of customer | varchar(10) | Not null |
| CustAddress | Address of customer | varchar(500) | Not null |

**CollectionAgent table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Description** | **Data Type** | **Constraint** |
| AgentID | Agent ID | Int(10) | Primary key associates with the CollectioAgent table. |
| AgentName | Name of Agent | Varchar(30) | Not null |
| AgentPhone | Contact of Agent | Varchar(10) | Not null |
| AgentEmail | Email address Agent | Varchar(25) | Reference to foreign keys from the customer table and null. |

**Payment table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Description** | **Data Type** | **Constraint** |
| typeID | Type ID | Int(10) | Primary key associates with payment type table. |
| PaymentType | Payment type | Varchar(10) | Null |

**Payment:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Description** | **Data Type** | **Constraint** |
| PaymentID | Payment ID | Int(10) | Primary key associates with payment table and not null |
| LoanID | Loan ID | Int(10) | Not Null |
| typeID | Type ID | Int(10) | Not Null |
| PaymentDate | Date of Payment | date | Not Null |
| AmountPaid | Total Amount | float | Not Null |
| Remark | Remarks | Varchar(100) | Null |

**Loan:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Description** | **Data Type** | **Constraint** |
| LoanID | Loan ID | Int(10) | Primary key associates with the loan table. |
| LoanStartDate | Start date of Loan | date | Not null |
| LoanEndDate | End date of Loan | date | Not null |
| InterestRate | Amount of Interest | float | Not null |
| LoanAmount | The amount of loan | float | Not null |
| PaymentDue | Payment due | float | Not null |
| Duedate | Due date | date | Null |
| PayFrequency | Frequency of payment | Int(11) | Null |
| CustID | Customer ID | Int(10) | Reference to foreign keys from the customer table. |
| AgentID | Agent ID | Int(10) | Reference to foreign keys from the Agent table. |

**Relationships**

One customer can have more than one loan and also one loan can be granted to more customers, and their association is many to many.

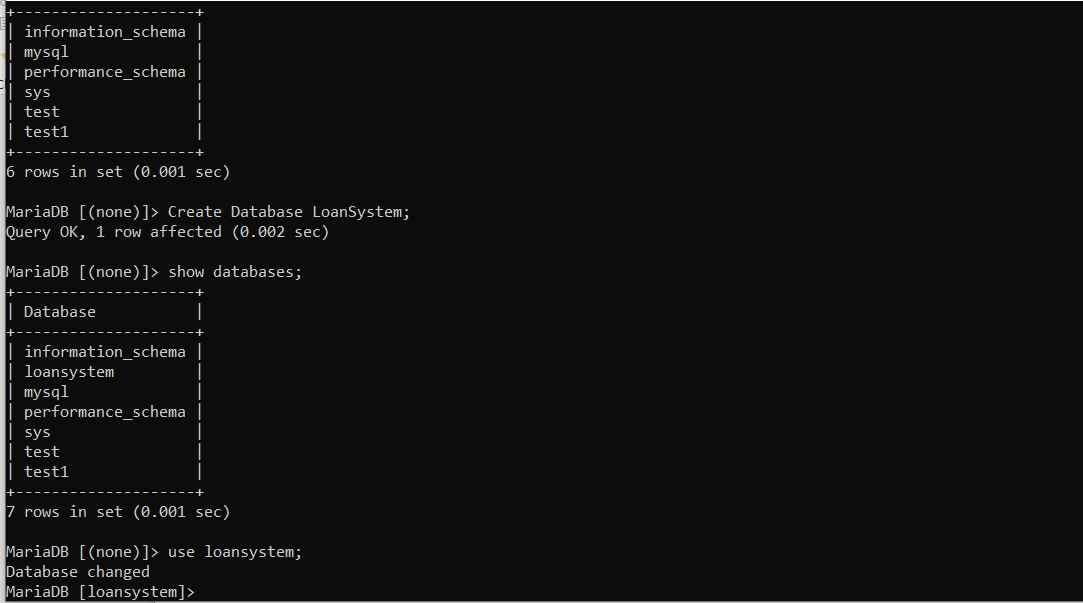
There are different modes of payment to be made and only one payment is made at a time, so it's a one-to-many relationship.

One collection agent may manage several loans from several customers and one loan from many customers with one-to-many associations.

When a loan can be repaid multiple times and a payment is associated with one loan at a time, the relationship is one-to-many.

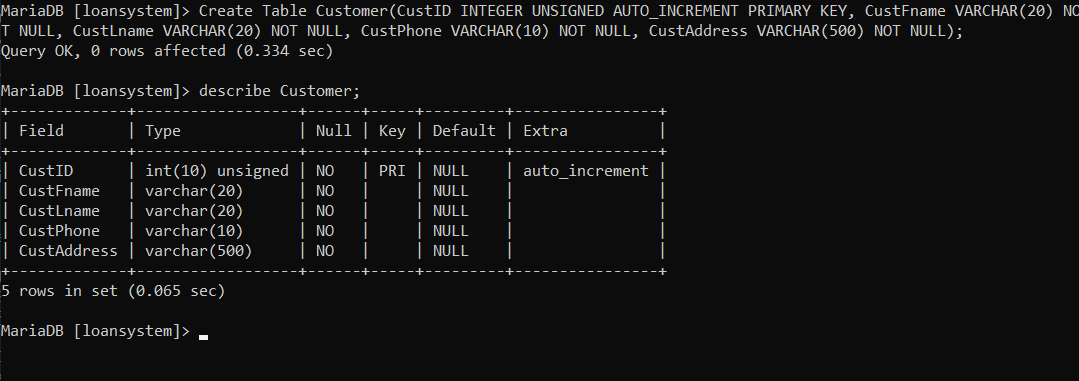
**Data Definition Language Scripts**

Based on the database requirements identified and explained in the above section, it is created using MariaDB. Commands used to create the databases, tables and the relations are given here

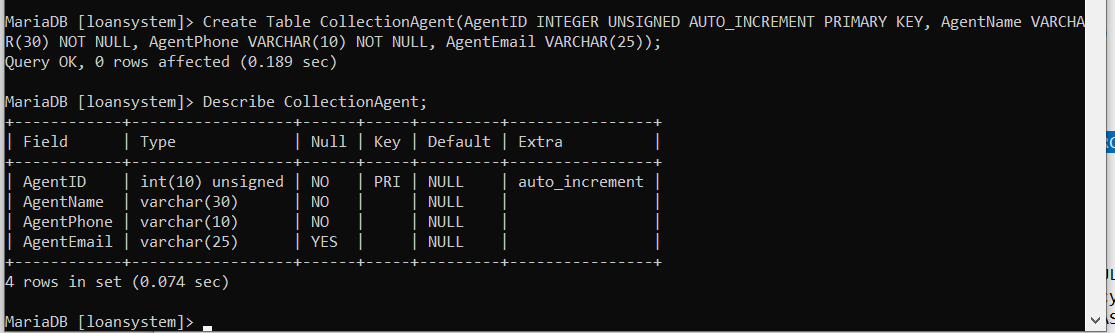


Now the required database i.e. Loansystem is created and it is used to create the tables as show below.

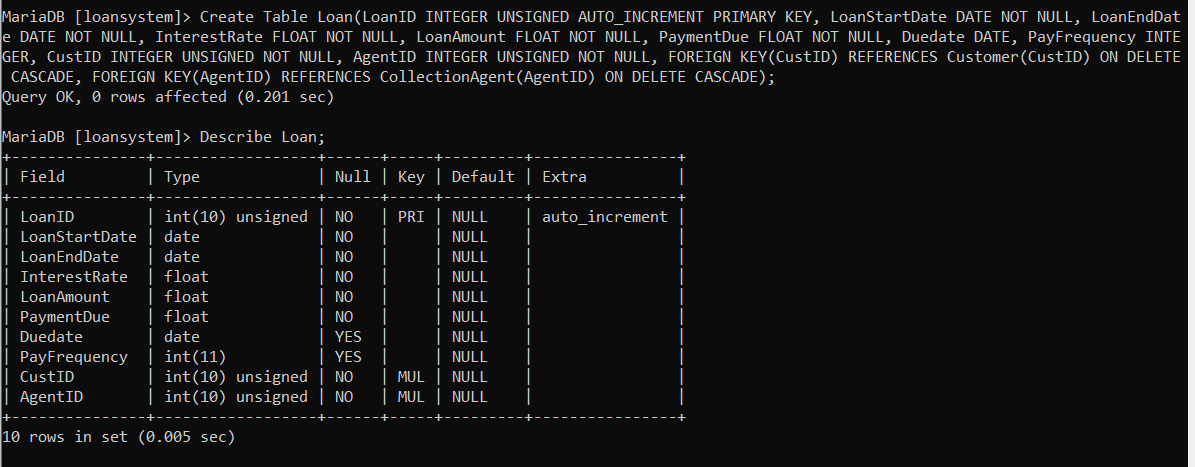
**Customer table**



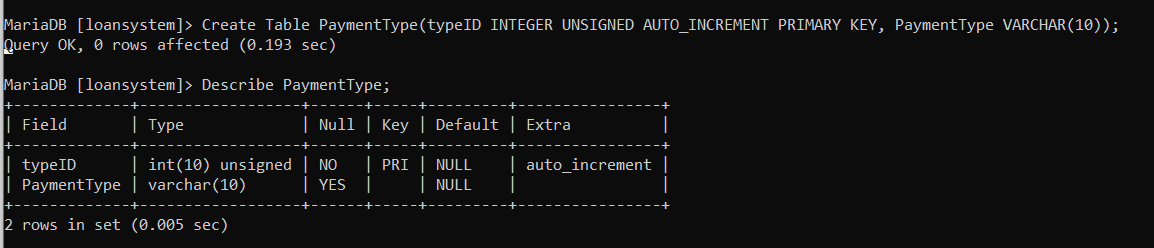
**CollectionAgent table**



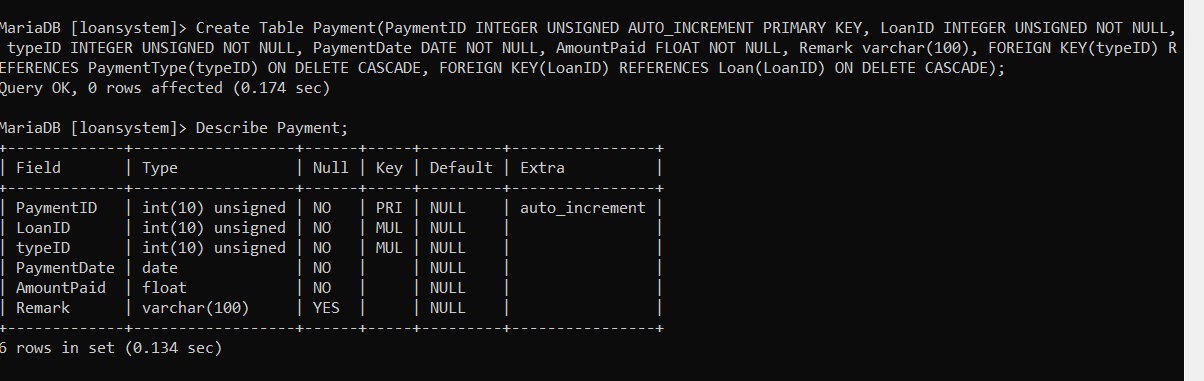
**Loan table**



**PaymentType table**



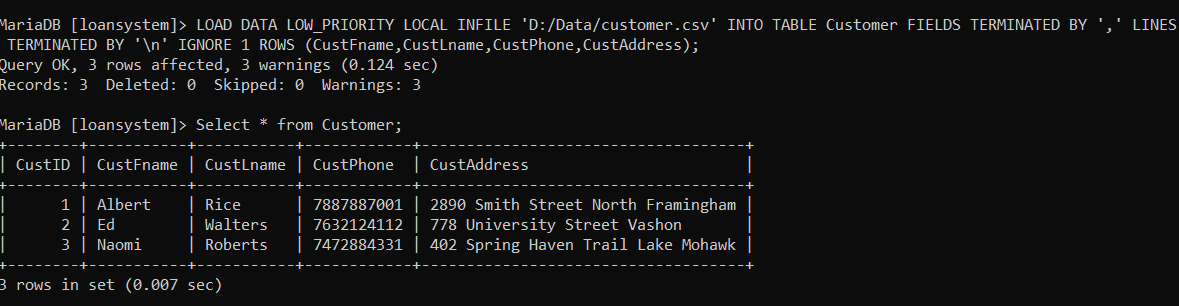
**Payment Table**

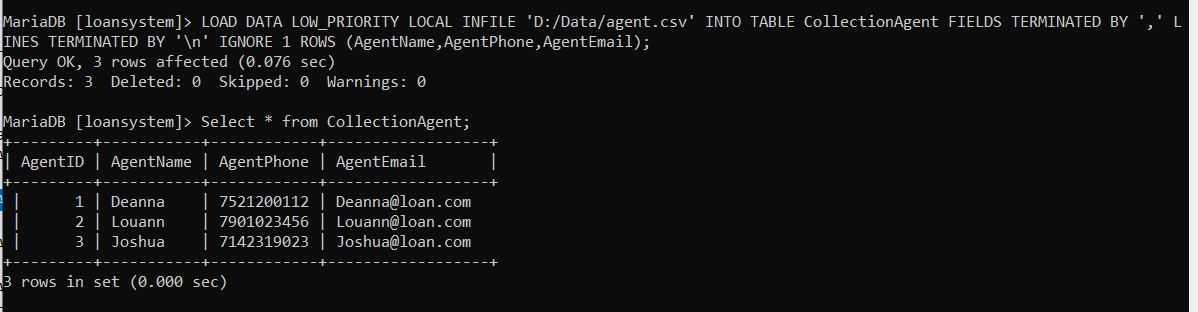


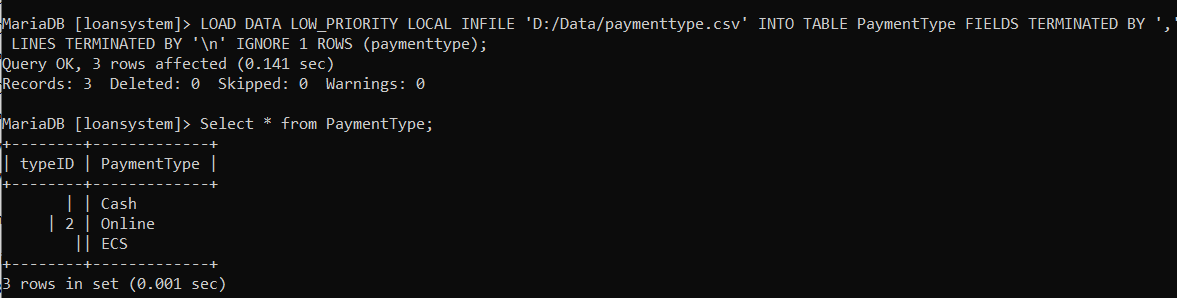
**Data Manipulation Language Scripts**

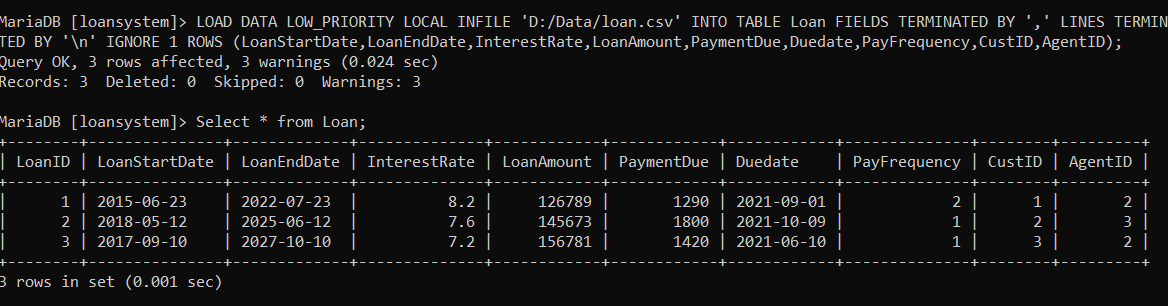
Once the required tables as mentioned are created, few records for each table are inserted from CSV files as uploaded with this project and the SQL commands used are given here

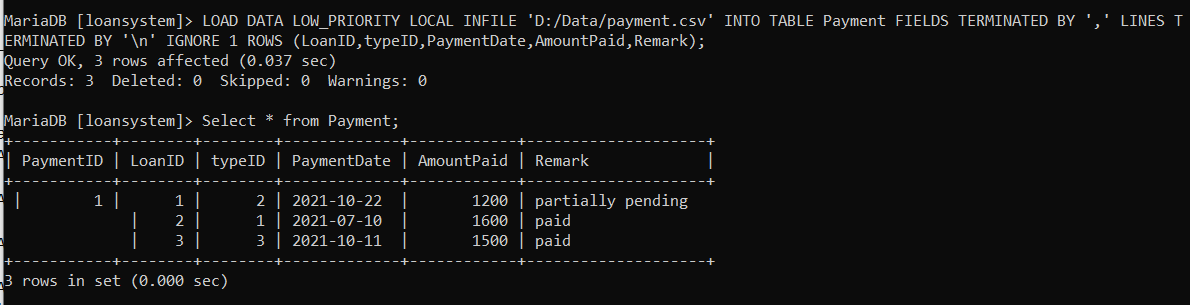
**Insert Statements**



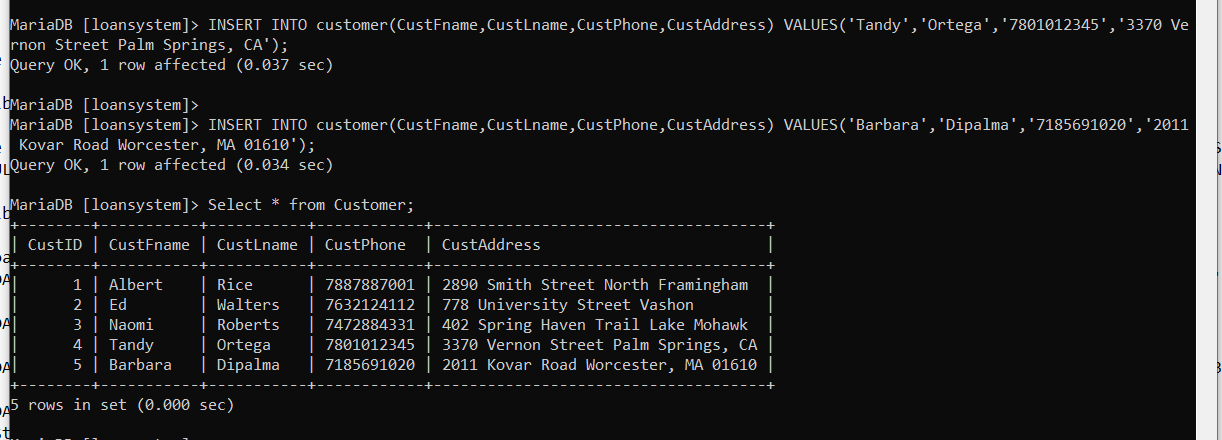


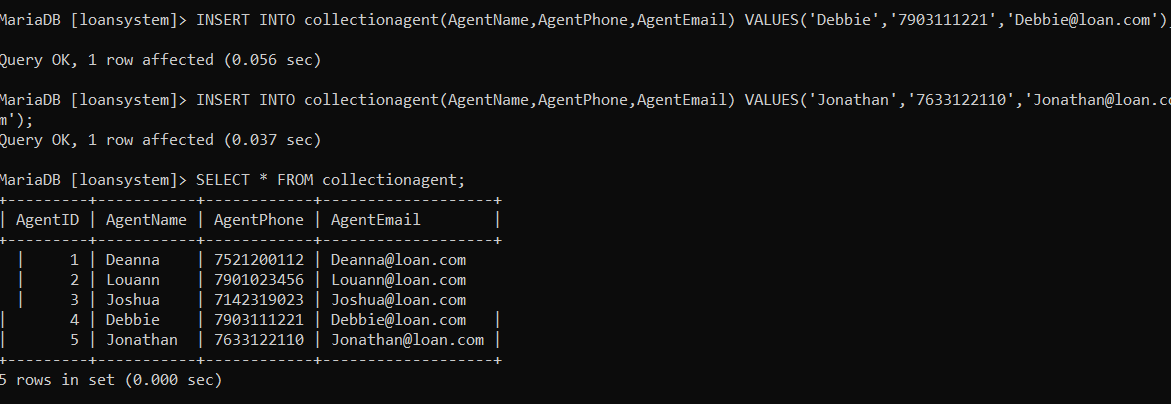


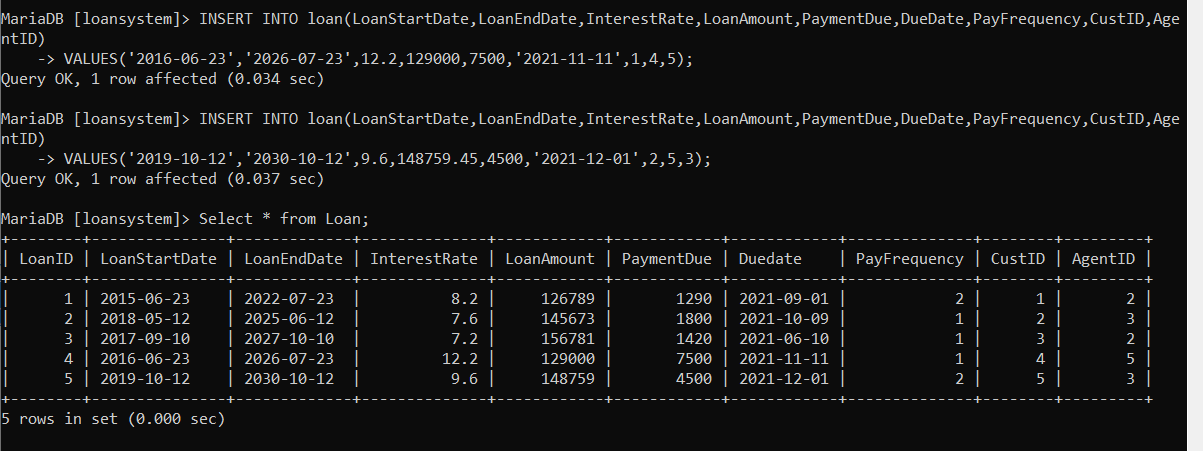


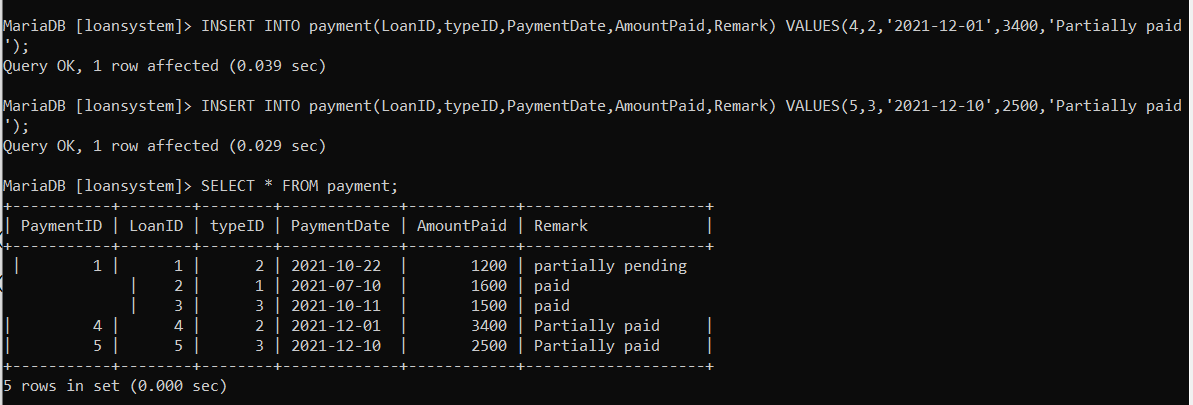


Now the data is inserted into the tables using normal SQL insert into statements as shown



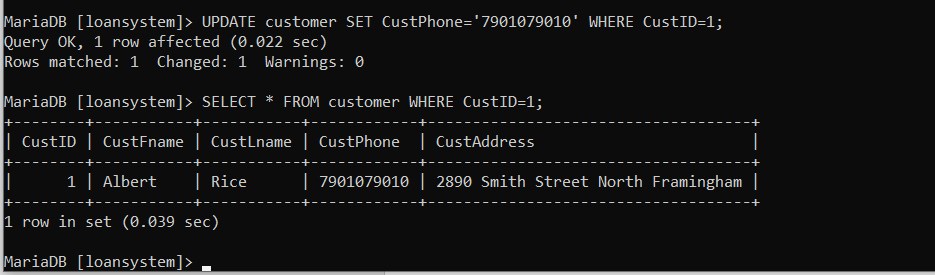




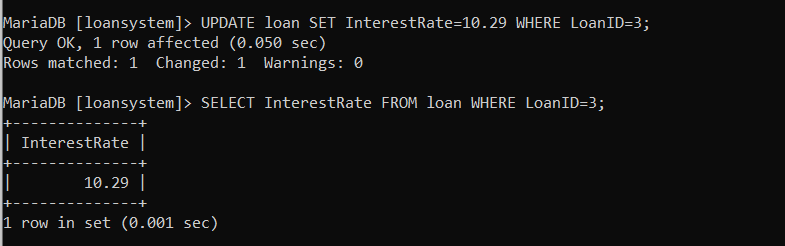


**Update Statements**

Customer phone number is updated for the customer id 1 and the corresponding output is shown below

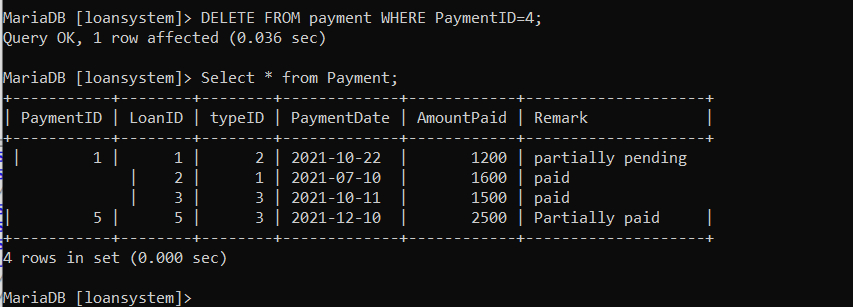


Interest rate of the loan is update for the loan id 3 and the corresponding output is shown below



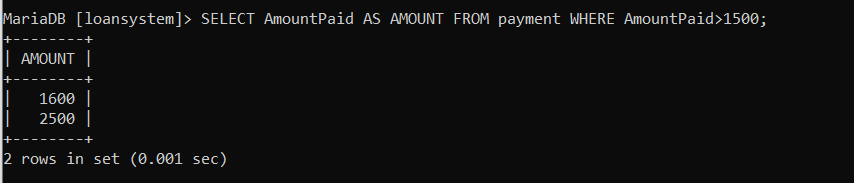
**Delete Statement**

Payment record is deleted for the payment id =4 and the corresponding output is shown below



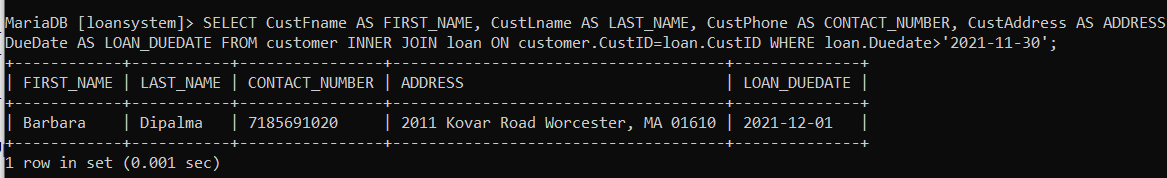
**Select Statements**

Amount paid from the payment table is selected for the payments where the amount > 1500 as shown below

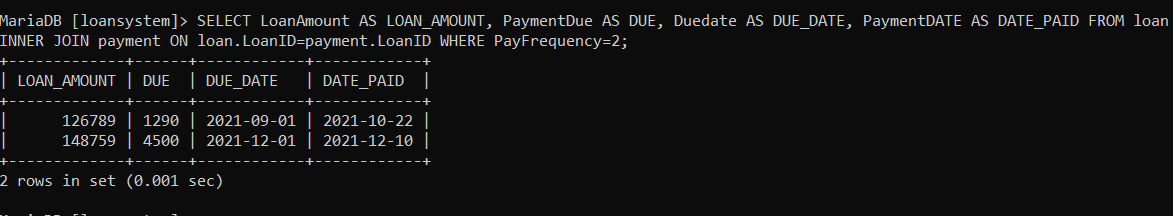


**Join Statements**

Details of the customer who has loan due date greater than 2021-11-30 is selected using the join statement as shown below

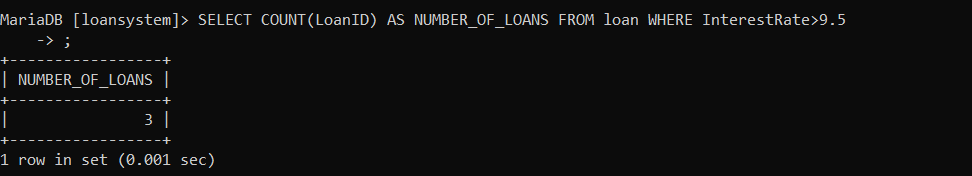


Loan details with payment frequency 2 are selected using join and shown below

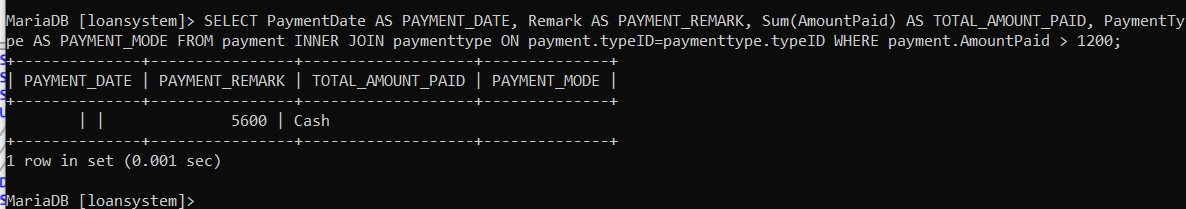


**Summary statements**

Number of loans with interest rate greater than 9.5 are selected using COUNT function as shown below



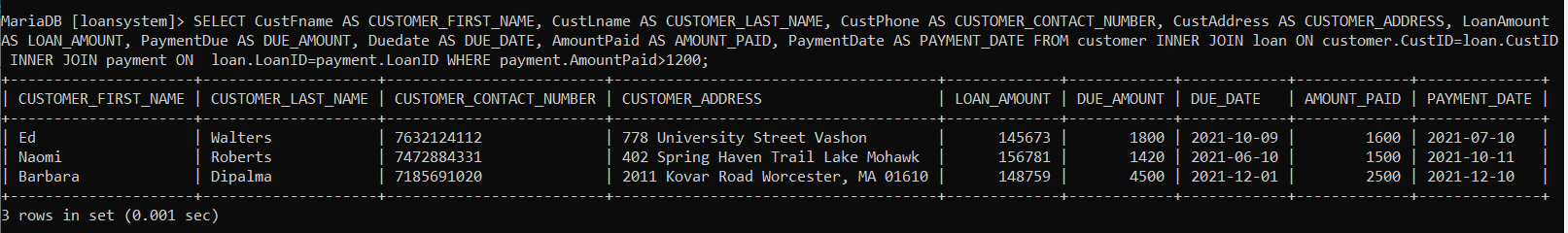
Sum of the payment amount paid using different payment types are selected using the SUM function for the payment amount > 1200 is shown below



**Multi-table Query**

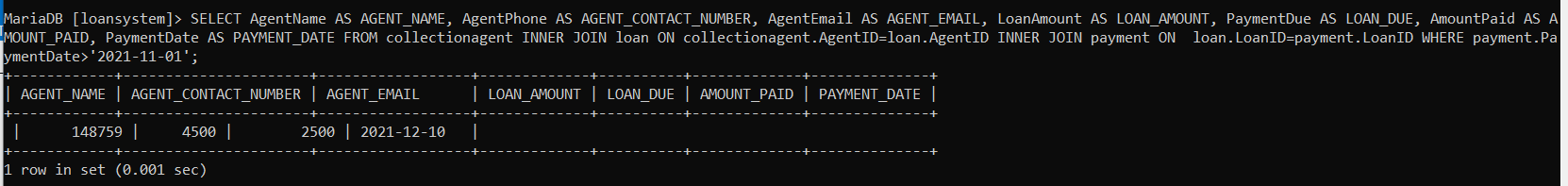
Customer loan and payment details where payment amount > 1200 is selected form the customer, loan and payment tables using inner join statements and shown below

SELECT CustFname AS CUSTOMER\_FIRST\_NAME, CustLname AS CUSTOMER\_LAST\_NAME, CustPhone AS CUSTOMER\_CONTACT\_NUMBER, CustAddress AS CUSTOMER\_ADDRESS, LoanAmount AS LOAN\_AMOUNT, PaymentDue AS DUE\_AMOUNT, Duedate AS DUE\_DATE, AmountPaid AS AMOUNT\_PAID, PaymentDate AS PAYMENT\_DATE FROM customer INNER JOIN loan ON customer.CustID=loan.CustID INNER JOIN payment ON loan.LoanID=payment.LoanID WHERE payment.AmountPaid>1200;



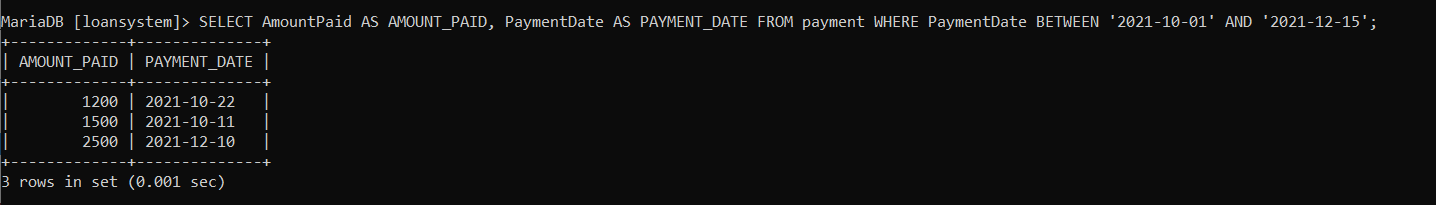
Agent details who collected the loan payments after 2021-11-01 are selected using the Collection agent table, loan table and payment table using the inner joins as shown below

SELECT AgentName AS AGENT\_NAME, AgentPhone AS AGENT\_CONTACT\_NUMBER, AgentEmail AS AGENT\_EMAIL, LoanAmount AS LOAN\_AMOUNT, PaymentDue AS LOAN\_DUE, AmountPaid AS AMOUNT\_PAID, PaymentDate AS PAYMENT\_DATE FROM collectionagent INNER JOIN loan ON collectionagent.AgentID=loan.AgentID INNER JOIN payment ON loan.LoanID=payment.LoanID WHERE payment.PaymentDate>'2021-11-01';



**Between statement**

Amount paid between the dates 2021-10-01 and 2021-12-15 are selected using the BETWEEN statement and shown below



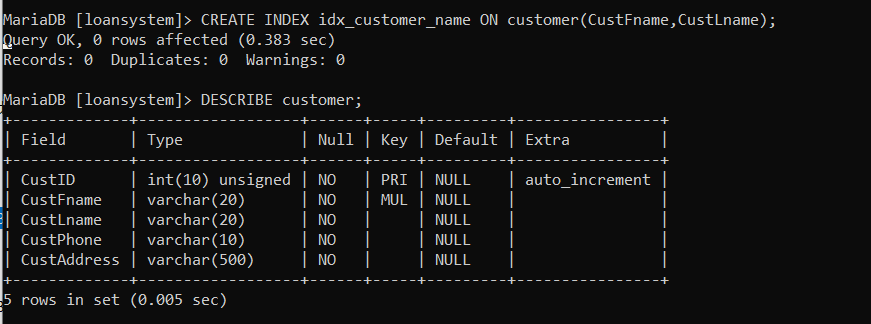
**Indexes**

Three indexes are created for the LoanSystem database to improve the performance of the search function and also select the data in an efficient manner and the corresponding queries are shown below

1st index is created on the Customer table, where the customer first name and last name are considered for this index. Further, the query performance can be optimized when selecting the customer names and using the Group by clause

CREATE INDEX idx\_customer\_name ON customer(CustFname,CustLname);

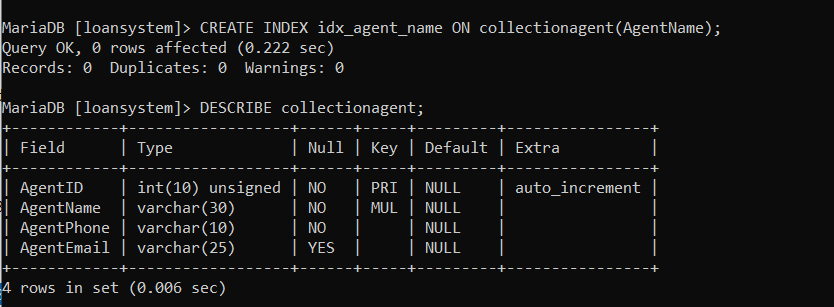
DESCRIBE customer;



Similarly, the second index is created on the Collection agent table on the Agent name as shown below

CREATE INDEX idx\_agent\_name ON collectionagent(AgentName);

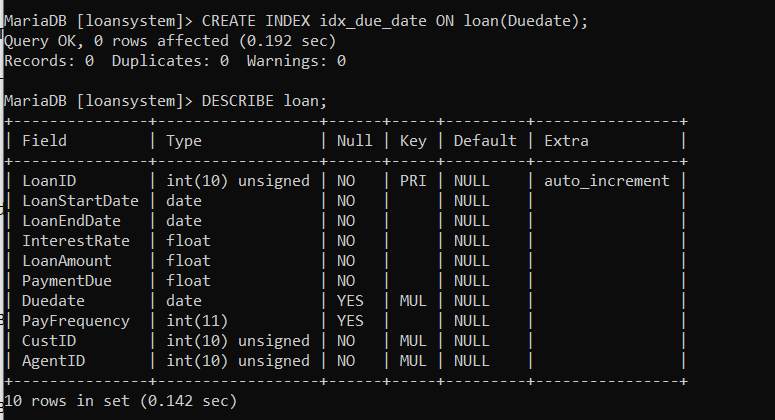
DESCRIBE collectionagent;



Third index is created on the Loan table with an index on the Due date as shown below

CREATE INDEX idx\_due\_date ON loan(Duedate);

DESCRIBE loan;



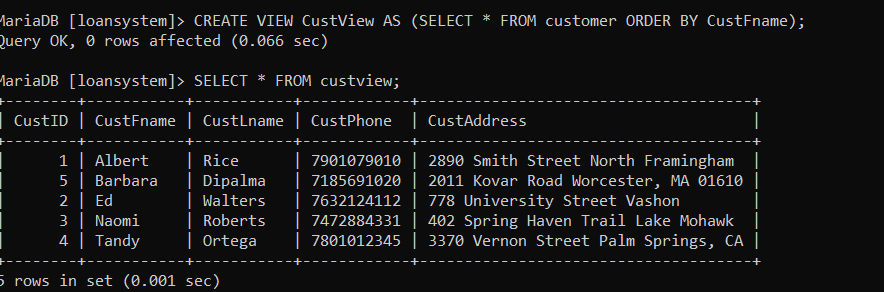
**Views**

Two views are created on this database and they are explained here.

First view is created to view the customer names order by Customer first name as shown below

CREATE VIEW CustView AS (SELECT \* FROM customer ORDER BY CustFname);

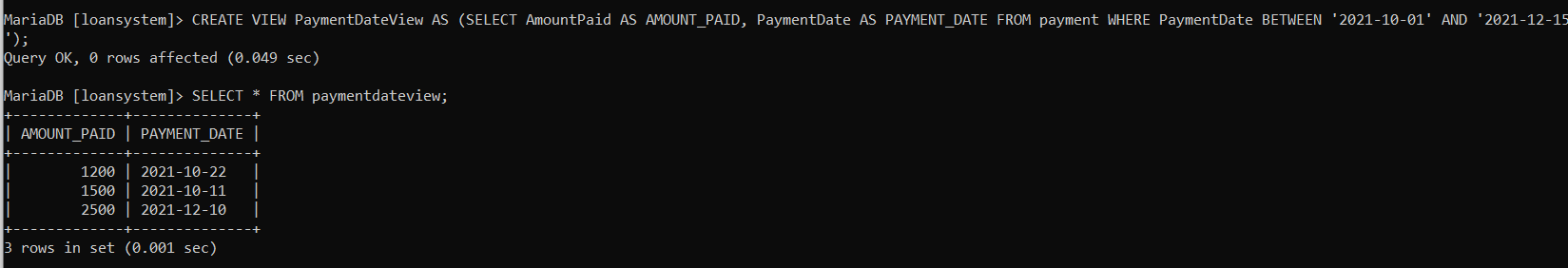
SELECT \* FROM custview;



Second view is created to view the payment date between the range 2021-10-01 and 2021-12-15 and shown below

CREATE VIEW PaymentDateView AS (SELECT AmountPaid AS AMOUNT\_PAID, PaymentDate AS PAYMENT\_DATE FROM payment WHERE PaymentDate BETWEEN '2021-10-01' AND '2021-12-15');

SELECT \* FROM paymentdateview;



Accessing these views will enable the users to view the data as populated with the query without rewriting the SQL statements

**Triggers**

A trigger is added on this database and it will be activated whenever a new payment record is inserted into the payment table. Condition used for this trigger event is that, if the payment date is greater than the current date the payment remark will be set as Not paid and it is given below

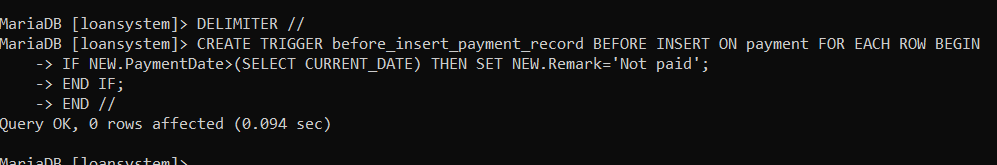
DELIMITER //

CREATE TRIGGER before\_insert\_payment\_record BEFORE INSERT ON payment FOR EACH ROW BEGIN

IF NEW.PaymentDate>(SELECT CURRENT\_DATE) THEN SET NEW.Remark='Not paid';

END IF;

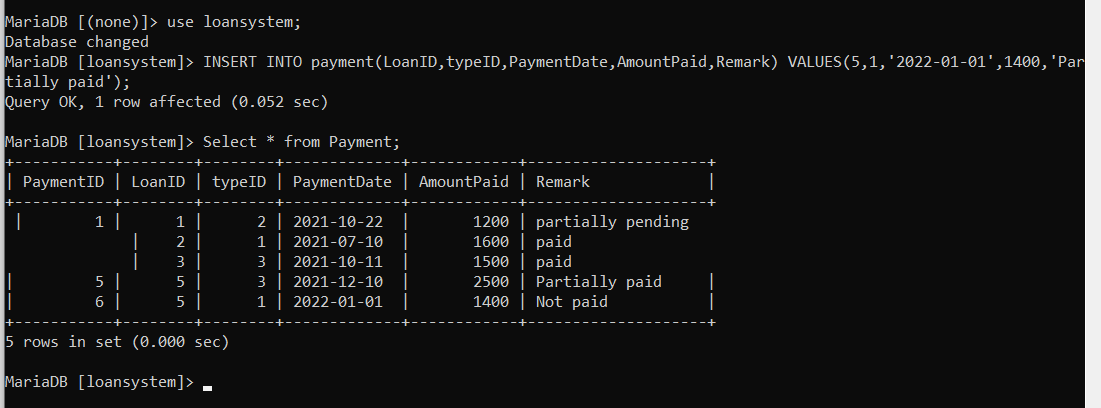
END //



Now a new payment record is inserted with payment date as 2022-01-01, which is greater than the current system date and the trigger will invoke automatically and the remarks will be changed as per the trigger set

INSERT INTO payment(LoanID,typeID,PaymentDate,AmountPaid,Remark) VALUES(5,1,'2022-01-01',1400,'Partially paid');

SELECT \* FROM payment;



Here, the remark of the new record is automatically set to not paid due to the trigger event created

**Transactions**

**Importance of transactions to ensuring ACID behavior**

Transactions play an important role in maintaining integrity on the database. Required integrity is achieved using the ACID behavior i.e., Atomicity, Consistency, Isolation and Durability ("IBM Docs", 2018).

* Atomicity ensures that all the changes on the database are done via a single transaction
* Consistency ensures that the data is always in consistent state when the transaction starts and ends
* Isolation ensures intermediate states of a transaction are not visible to the other transactions
* Durability ensures that the transaction completes successfully and the persistent data changes are not undone during the case of system failures.

A Consistency transaction is applied on this database, where amount paid from the payment table is selected using a select statement and also insert statement is used to insert values into the payment table while this transaction is being run as shown below. Commit and Rollback statements are used to ensure the data is inserted and the corresponding insert transaction is roll backed

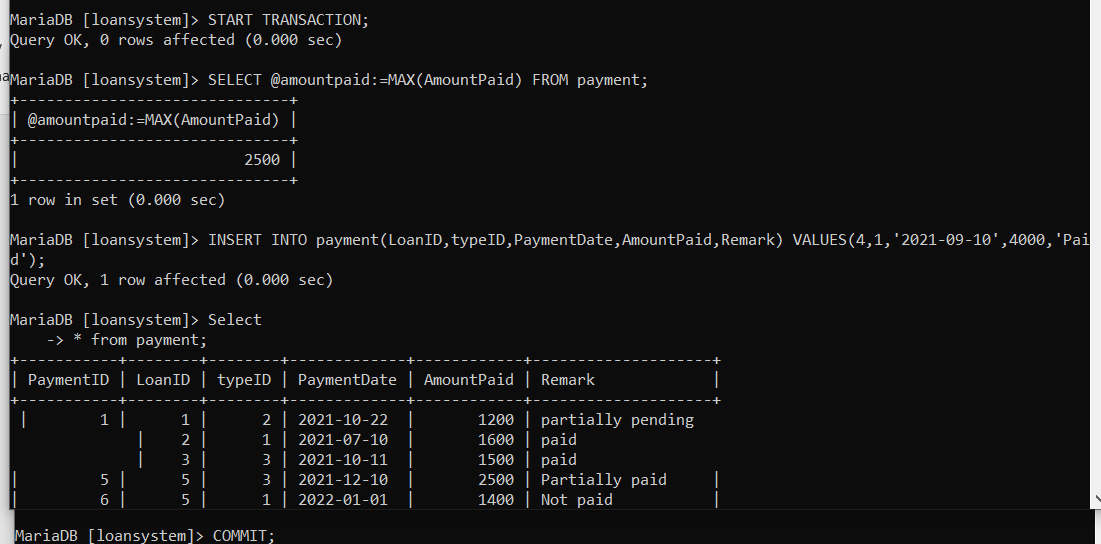
START TRANSACTION;

SELECT @amountpaid:=MAX(AmountPaid) FROM payment;

INSERT INTO payment(LoanID,typeID,PaymentDate,AmountPaid,Remark) VALUES(4,1,'2021-09-10',4000,'Paid');

COMMIT;

ROLLBACK;



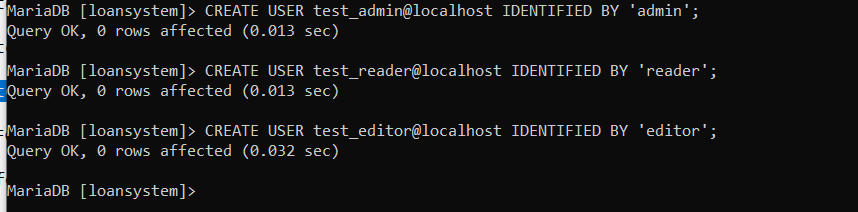
**Security**

To demonstrate the users and their previleges, three users are created on this database and they are test\_admin@localhost, test\_reader@localhost and test\_editor@localhost using the following statements

CREATE USER test\_admin@localhost IDENTIFIED BY 'admin';

CREATE USER test\_reader@localhost IDENTIFIED BY 'reader';

CREATE USER test\_editor@localhost IDENTIFIED BY 'editor';

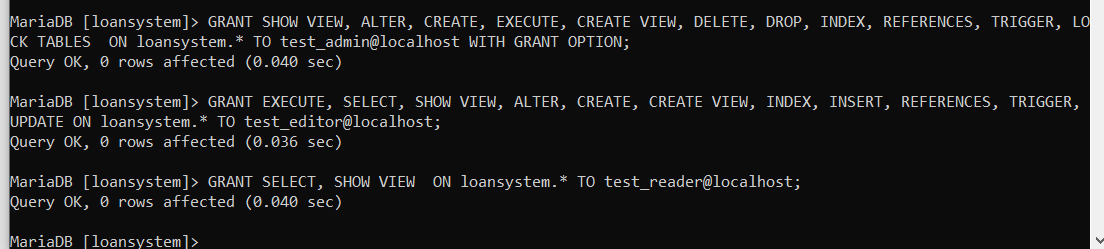


Following grants are given for each of these users

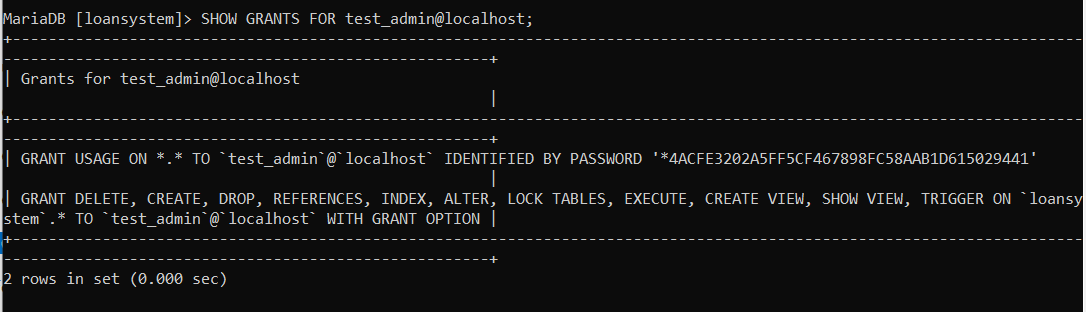
GRANT SHOW VIEW, ALTER, CREATE, EXECUTE, CREATE VIEW, DELETE, DROP, INDEX, REFERENCES, TRIGGER, LOCK TABLES ON loansystem.\* TO test\_admin@localhost WITH GRANT OPTION;

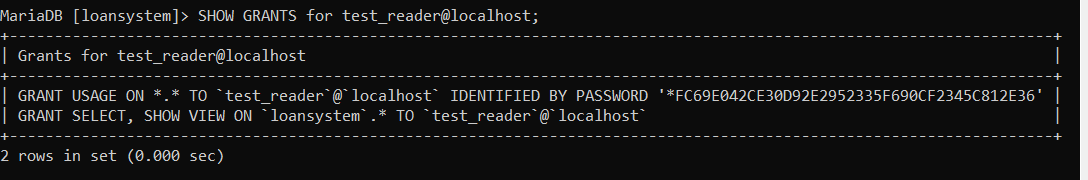
GRANT EXECUTE, SELECT, SHOW VIEW, ALTER, CREATE, CREATE VIEW, INDEX, INSERT, REFERENCES, TRIGGER, UPDATE ON loansystem.\* TO test\_editor@localhost;

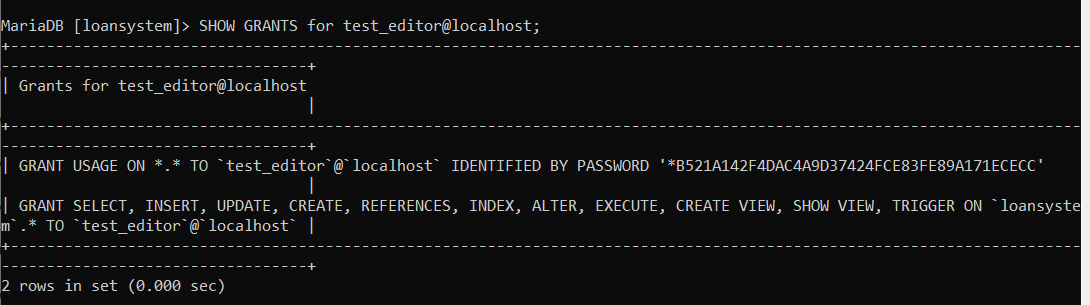
GRANT SELECT, SHOW VIEW ON loansystem.\* TO test\_reader@localhost;



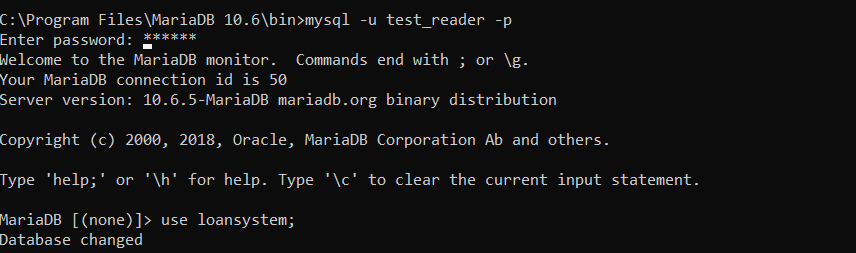
Grants given to these users are viewed using the following SQL commands and the outputs are shown below



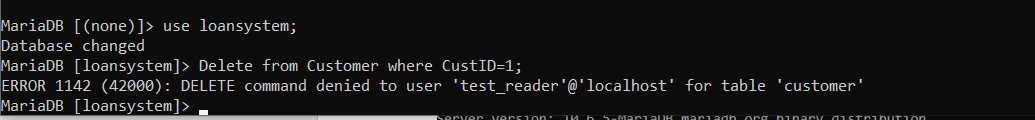




To test the grants, root user is switched to test\_reader as shown below



Now this used had tried to delete a record from customer and it has shown the error as the test\_reader doesn’t have access to delete the records

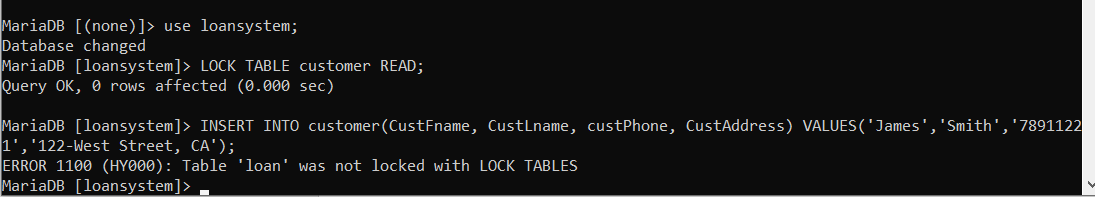


**Locking**

READ Lock is applied on the Customer table, such that only read operations can be performed on this table and writing or inserting a new record is inserted and the process is given below

LOCK TABLE customer READ;

INSERT INTO customer(CustFname, CustLname, custPhone, CustAddress) VALUES('James','Smith','78911221','122-West Street, CA');

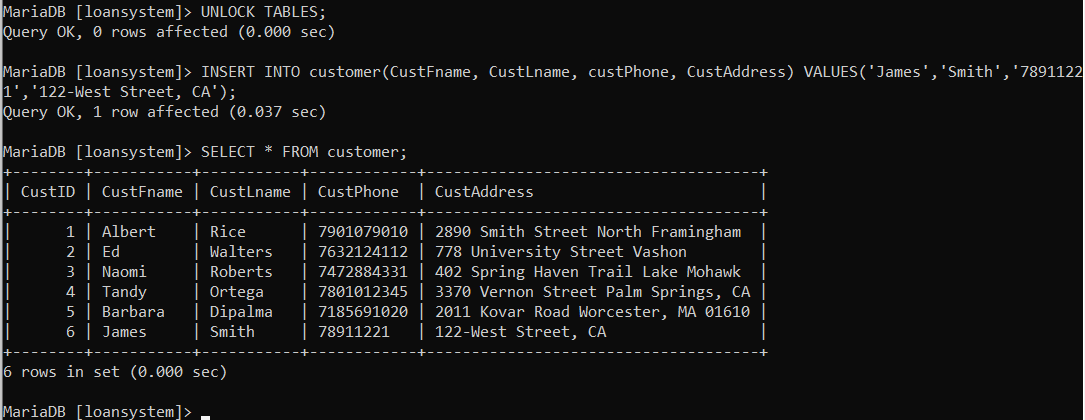


Now unlock operation is applied to check this feature

UNLOCK TABLES;

INSERT INTO customer(CustFname, CustLname, custPhone, CustAddress) VALUES('James','Smith','78911221','122-West Street, CA');

SELECT \* FROM customer;



**Backup**

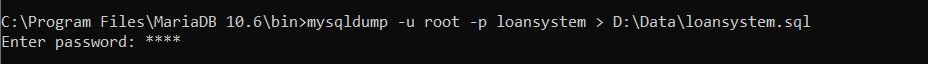
Database backup and restore features are supported with Maria DB, where the entire database can be created as physical database dump or a SQL file. Backup commands can be automated by using trigger. Required condition on this trigger event can be set, such that, backup of the database occurs at the 12:00 AM every day or as per the schedule mention in the Trigger code

Backup commands used on this database are shown below

mysqldump -u root -p loansystem > D:\Data\loansystem.dump



mysqldump -u root -p loansystem > D:\Data\loansystem.sql



**Python Programming**

import mariadb

import sys

try:

cm\_connection = mariadb.connect(

user="root",

password="root",

host="127.0.0.1",

port=5701,

database="loansystem")

except mariadb.Error as e:

print(f"Error connecting to MariaDB Platform: {e}")

sys.exit(1)

else:

mycursor=cm\_connection.cursor()

Q1="SELECT \* FROM payment WHERE AmountPaid>1500;"

mycursor.execute(Q1)

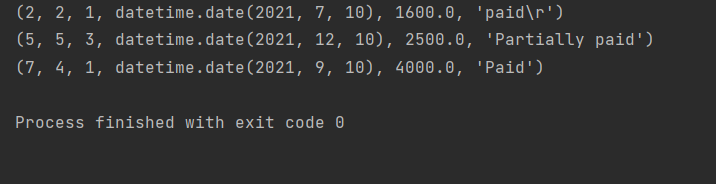
results=mycursor.fetchall()

for i in results:

print(i)

cm\_connection.close()

Output is shown



**Suggested Future Work**

The MariaDB is a relational database to maintain the features and structure of MySQL. It uses SQL interface in order to access the data. This model uses client server architecture for accessing the services provided by the server. The SQL statements are useful to access and retrieve the data effectively. It has high performance compared to the SQL. It provides high security features to protect the user’s data without getting any harm. The system performs faster due to its small code. Despite all the initial promises, MariaDB is no longer completely compatible with MySQL. Due to its incompatibility with SQL, it cannot support all of its features. MariaDB lacks a few features available only in the MySQL Enterprise Edition. After MariaDB, there was no way to switch back to MySQL. It needs to have better debugging procedures and functions to minimize the errors and increase the efficiency (Raman et al., 2021).

To decrease the issues that cause in the system by improving the stability. Multiple imports of the data to the database would be more useful and time-saving. Uploading large data sets and importing them into a database requires a CSV file, as the system does not activate automatically by default in the latest version. To increase the system's performance and to increase its capability to work, the compatibility of the system must be improved. The leverage cloud services help in storing the data and accessing it in an effective order. In order to prevent hackers from altering the data, it provides the protocols related to security. It offers the scalability functions to increase the growth and to enhance the performance time of operations. When working on a database using the leverage cloud service, security, scalability, and flexibility are provided as an advantage (Kaushik et al., 2017). It improves the system performance by providing multiple features. In a flexible manner, it is possible to combine different formats of data such as structured and unstructured.

It is possible to update the schema dynamically and adapt to the changing requirements. The NoSQL database is designed for high performance. It fails to provide the standardization in order to define the rules and roles of NoSQL. It doesn't focus on consistency and will accept rows without error, whereas relational databases ensure that no duplicate rows are stored in the database. The NoSQL database is document-based, whereas MySQL relies on a tabular structure. The NoSQL can be easily scalable, whereas the MySQL is tough to scale the big data. The MySQL is lesser in flexibility compared to the NoSQL. It is the dynamic schema that makes NOSQL so advantageous, whereas MySQL schemas are rigid by nature (Jose & Abraham, 2020).

**References**

*IBM Docs*. Ibm.com. (2018). Retrieved 17 December 2021, from https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions.

Jose, B., & Abraham, S. (2020). Performance analysis of NoSQL and relational databases with MongoDB and MySQL. *Materials Today: Proceedings*, *24*, 2036-2043.

Kaushik, A., Kumar, D., & Chaturvedi, D. (2017). Analysis of Database Services for Cloud. *International Journal Of Engineering And Computer Science*.

Raman, A., Sharma, D., & Phaujdar, A. (2021). *MariaDB vs MySQL: A Comprehensive Analysis*. Learn | Hevo. Retrieved 17 December 2021, from <https://hevodata.com/learn/mariadb-vs-mysql/>.