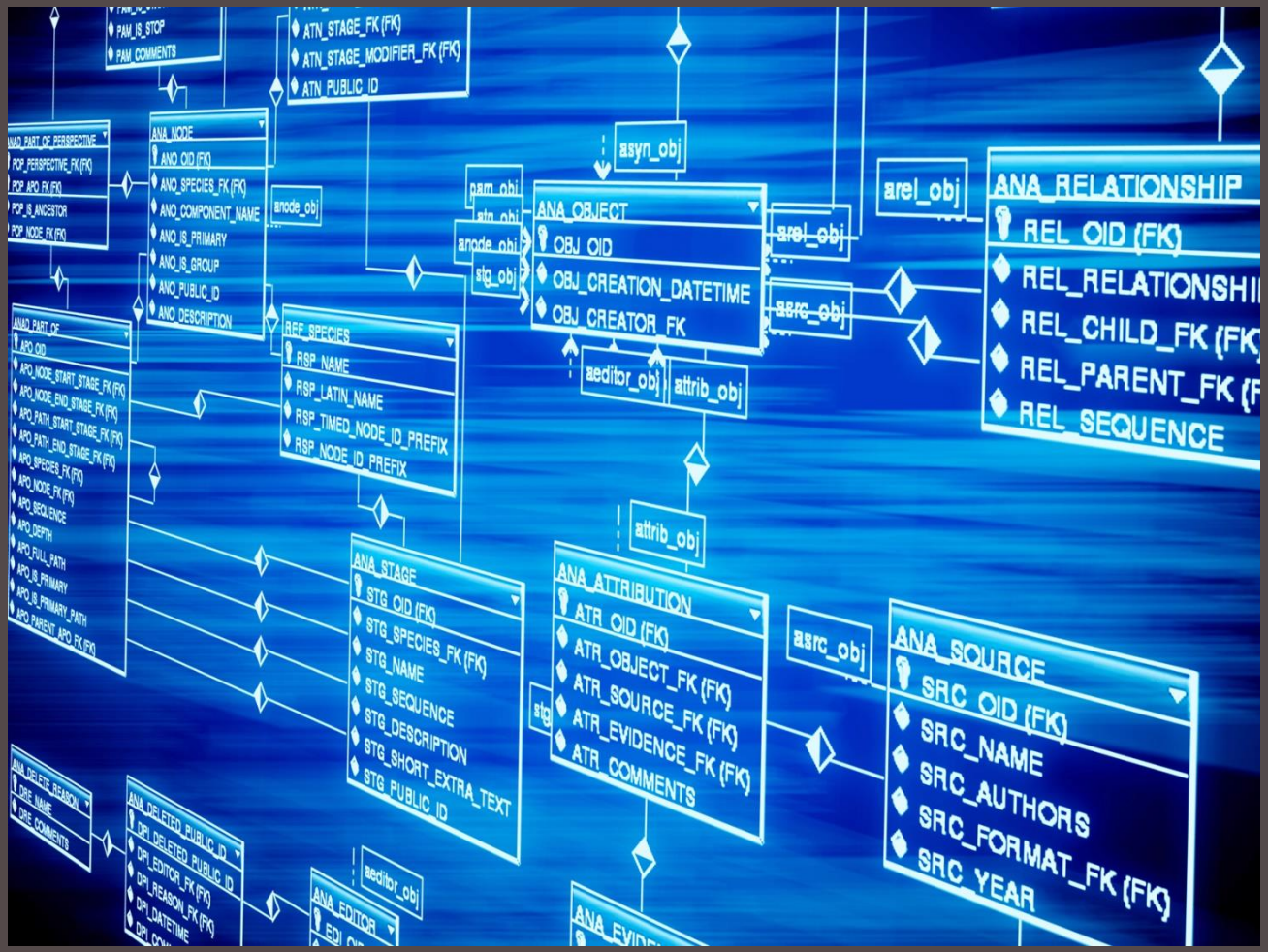
# DATABASE MANAGEMENT AND DATABASE DESIGN

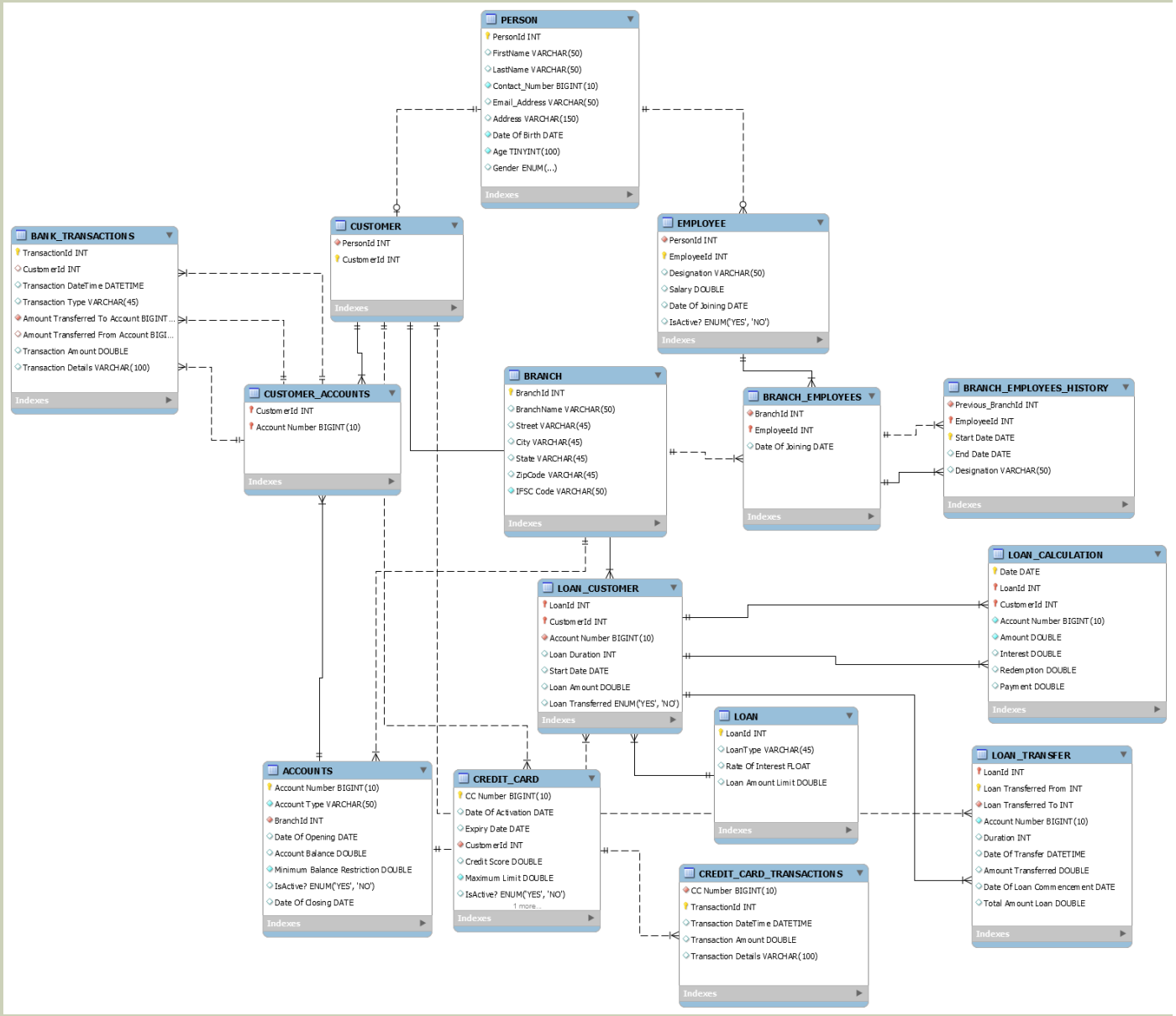



SELECT @firstName = **‘VAIBHAVI’,**
@lastName = **‘KAMANI’**,
@NUID = **001825058**
FROM Students
WHERE Section = **06**
AND Project_Topic = **‘Banking Database System’**;

# PROPOSAL

- A database design is an important part of any system.
- The purpose of the banking system is to develop a banking database design that provides banks with the facility to organize various information related to employees, customers, credit cards, loans, etc.
- This database shall help banks to maintain their information at one place and provide an overview of the bank operations.
- The database can be used to perform the statistical analysis like number of transactions done per week or month or year per branch, total profit generated by each branch, total number of accounts particular customer has, types of loans that granted more to the customers etc.
- This kind of analysis is important for bank so as to ensure smooth and efficient functioning.

- The users of the applications will be Branch Manager, Bank Teller, Bank Clerk, Loan Manager, Human Resource, Bank Manager.
- Bank Manager will be having access to view the overall database across the branches.
- Branch Manager will be having access to view only the information related to his/her branch.
- Bank Teller will be having access to update the information of the customer, perform various transactions, view the overall reports generated for the clients.
- Bank Clerk will be having access to just view the data related to accounts and provide information when queried by customer.
- Loan Manager will be having access to grant loans to the customers as well as monitor the loan related information such as deciding loan rates, calculating interest for customers etc.
- Human Resource will be having access for hiring the employees to work in the bank.


PERSON
PersonId INT
FirstName VARCHAR(50)
LastName VARCHAR(50)
Contact_Number BIGINT(10)
Email_Address VARCHAR(50)
Address VARCHAR(150)
Date Of Birth DATE
Age TINYINT(100)
Gender ENUM(...)
Indexes
CUSTOMER
PersonId INT
CustomerId INT
Indexes
EMPLOYEE
PersonId INT
EmployeeId INT
Designation VARCHAR(50)
Salary DOUBLE
Date Of Joining DATE
IsActive? ENUM('YES', 'NO')
Indexes
BANK_TRANSACTIONS
TransactionId INT
CustomerId INT
Transaction DateTime DATETIME
Transaction Type VARCHAR(45)
Amount Transferred To Account BIGINT ...
Amount Transferred From Account BIGI...
Transaction Amount DOUBLE
Transaction Details VARCHAR(100)
Indexes
CUSTOMER_ACCOUNTS
CustomerId INT
Account Number BIGINT(10)
Indexes
BRANCH
BranchId INT
BranchName VARCHAR(50)
Street VARCHAR(45)
City VARCHAR(45)
State VARCHAR(45)
ZipCode VARCHAR(45)
IFSC Code VARCHAR(50)
Indexes
BRANCH_EMPLOYEES
BranchId INT
EmployeeId INT
Date Of Joining DATE
Indexes
BRANCH_EMPLOYEES_HISTORY
Previous_BranchId INT
EmployeeId INT
Start Date DATE
End Date DATE
Designation VARCHAR(50)
Indexes
LOAN_CALCULATION
Date DATE
LoanId INT
CustomerId INT
Account Number BIGINT(10)
Amount DOUBLE
Interest DOUBLE
Redemption DOUBLE
Payment DOUBLE
Indexes
LOAN_CUSTOMER
LoanId INT
CustomerId INT
Account Number BIGINT(10)
Loan Duration INT
Start Date DATE
Loan Amount DOUBLE
Loan Transferred ENUM('YES', 'NO')
Indexes
LOAN
LoanId INT
LoanType VARCHAR(45)
Rate Of Interest FLOAT
Loan Amount Limit DOUBLE
Indexes
LOAN_TRANSFER
LoanId INT
Loan Transferred From INT
Loan Transferred To INT
Account Number BIGINT(10)
Duration INT
Date Of Transfer DATETIME
Amount Transferred DOUBLE
Date Of Loan Commencement DATE
Total Amount Loan DOUBLE
Indexes
ACCOUNTS
Account Number BIGINT(10)
Account Type VARCHAR(50)
BranchId INT
Date Of Opening DATE
Account Balance DOUBLE
Minimum Balance Restriction DOUBLE
IsActive? ENUM('YES', 'NO')
Date Of Closing DATE
Indexes
CREDIT_CARD
CC Number BIGINT(10)
Date Of Activation DATE
Expiry Date DATE
CustomerId INT
Credit Score DOUBLE
Maximum Limit DOUBLE
IsActive? ENUM('YES', 'NO')
1 more...
Indexes
CREDIT_CARD_TRANSACTIONS
CC Number BIGINT(10)
TransactionId INT
Transaction DateTime DATETIME
Transaction Amount DOUBLE
Transaction Details VARCHAR(100)
Indexes


DATABASE

# IMPLEMENTATIONS:

- TRIGGERS
- STORED PROCEDURES
- UDF
- VIEW
- TRANSACTION
- JOINS
- SUBQUERY
- VARIOUS FUNCTIONS
- NORMALIZATION
- INHERITANCE (Person IS A Customer, Person IS A Employee)
- USERS & PRIVILEGES
- BACKUP

# PROCEDURES:

1. beginTransaction

- The procedure “beginTransaction” is used to withdraw and deposit money to a particular account number.
- The Stored Procedure calls Transaction within it and updates the amount in the respective accounts as well inserts the vales in the table.
- It checks the various conditions such as don’t withdraw the amount if amount is greater than minimum account balance.

```
DROP PROCEDURE IF EXISTS beginTransaction;
DELIMITER $$
CREATE PROCEDURE beginTransaction(IN amount double,IN transactionType VARCHAR(20),IN accNum BIGINT(10))
BEGIN
    IF transactionType = 'Withdraw' THEN
        SET @minAmt = (SELECT `Minimum Balance Restriction`
                       FROM Accounts
                       WHERE `Account Number` = accNum);
        SET @balRem = (SELECT `Account Balance`
                       FROM Accounts
                       WHERE `Account Number` = accNum);
        SET @transactDetails = 'Amount withdrawed.';

        IF @balRem - amount > @minAmt THEN
            START TRANSACTION;
                SET @amtRemain = @balRem - amount;
                UPDATE Accounts
                SET `Account Balance` = @amtRemain
                WHERE `Account Number` = accNum ;

                SET @custId = (SELECT CustomerId FROM CUSTOMER_ACCOUNTS WHERE `Account Number` = accNum);

                INSERT INTO BANK_TRANSACTIONS (`Transaction DateTime`,`Transaction Type`,`Amount Transferred To Ac
                VALUES
                (NOW(),transactionType,accNum,@custId,amount,@transactDetails);
            COMMIT;

            SELECT 'Transaction Completed Successfully.';
        ELSE
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT ='The remaining amount is less than the minimum balance in the account!';
        END IF;
    END IF;
```

```
    IF transactionType = 'Deposit' THEN
        SET @balRem = (SELECT `Account Balance`
                       FROM Accounts
                       WHERE `Account Number` = accNum);
        SET @transactDetails = 'Amount deposited.';

        START TRANSACTION;
                SET @amtRemain = @balRem + amount;
                SET @custId = (SELECT CustomerId FROM CUSTOMER_ACCOUNTS WHERE `Account Number` = accNum);

                UPDATE Accounts
                SET `Account Balance` = @amtRemain
                WHERE `Account Number` = accNum ;

                INSERT INTO BANK_TRANSACTIONS(`CustomerId`,`Transaction DateTime`,`Transaction Type`,`Amount Transferred To
                VALUES
                (@custId,NOW(),transactionType,accNum,amount,@transactDetails);
        COMMIT;

        SELECT 'Transaction Completed Successfully.';
    END IF;
END
$$
```

```sql
call beginTransaction(1000,'Withdraw',1234569872);
call beginTransaction(10000,'Deposit',1234569872);
call beginTransaction(12000,'Deposit',4558223682);
call beginTransaction(20000,'Deposit',4558223682);
call beginTransaction(4500,'Withdraw',4558223682);

SELECT * FROM Bank_Transactions;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| TransactionId | CustomerId | Transaction DateTime | Transaction Type | Amount Transferred To Account | Amount Transferred From Account | Transaction Amount | Transaction Details |
|---|---|---|---|---|---|---|---|
| 1 | 13 | 2017-12-13 05:27:30 | Withdraw | 1234569872 | NULL | 1000 | Amount withdrawed. |
| 2 | 13 | 2017-12-13 05:27:30 | Deposit | 1234569872 | NULL | 10000 | Amount deposited. |
| 3 | 1 | 2017-12-13 05:27:30 | Deposit | 4558223682 | NULL | 12000 | Amount deposited. |
| 4 | 1 | 2017-12-13 05:27:31 | Deposit | 4558223682 | NULL | 20000 | Amount deposited. |
| 5 | 1 | 2017-12-13 05:27:31 | Withdraw | 4558223682 | NULL | 4500 | Amount withdrawed. |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Result 1 | Result 2 | Result 3 | Result 4 | Result 5 | Bank_Transactions 6 | Apply

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1630 | 05:27:30 | call beginTransaction(1000,'Withdraw',1234569872) | 1 row(s) returned |
| 1631 | 05:27:30 | call beginTransaction(10000,'Deposit',1234569872) | 1 row(s) returned |
| 1632 | 05:27:30 | call beginTransaction(12000,'Deposit',4558223682) | 1 row(s) returned |
| 1633 | 05:27:31 | call beginTransaction(20000,'Deposit',4558223682) | 1 row(s) returned |
| 1634 | 05:27:31 | call beginTransaction(4500,'Withdraw',4558223682) | 1 row(s) returned |
| 1635 | 05:27:31 | SELECT * FROM Bank_Transactions LIMIT 0, 1000 | 5 row(s) returned |

## 2. transferInterAccAmount

- The procedure “transferInterAccAmount” is used to transfer amount from one account to another and update values in the respective accounts.
- The Stored Procedure calls Transaction within it and updates the amount in the respective accounts as well inserts the vales in the transaction table.
- It checks the various conditions such as don’t transfer the amount if amount is greater than minimum account balance.

```mysql
DROP PROCEDURE IF EXISTS transferInterAccAmount;
DELIMITER $$
CREATE PROCEDURE transferInterAccAmount(IN amount INT, IN accFrom BIGINT(10),IN accTo BIGINT(10))
BEGIN
    SET @minAmt = (SELECT `Minimum Balance Restriction`
                   FROM Accounts
                   WHERE `Account Number` = accFrom);
    SET @balRemAccFrom = (SELECT `Account Balance`
                   FROM Accounts
                   WHERE `Account Number` = accFrom);
    SET @balRemAccTo = (SELECT `Account Balance`
                   FROM Accounts
                   WHERE `Account Number` = accTo);
    SET @transactDetails = 'Amount transferred.';

    IF @balRemAccFrom - amount > @minAmt THEN
      START TRANSACTION;
          SET @amtRemain = @balRemAccFrom - amount;
          SET @amtForAccTo = @balRemAccTo + amount;
          SET @custId = (SELECT CustomerId FROM CUSTOMER_ACCOUNTS WHERE `Account Number` = accFrom);

          UPDATE Accounts
          SET `Account Balance` = @amtRemain
          WHERE `Account Number` = accFrom ;

          UPDATE Accounts
          SET `Account Balance` = @amtForAccTo
          WHERE `Account Number` = accTo ;

          INSERT INTO BANK_TRANSACTIONS(`CustomerId`,`Transaction DateTime`,`Transaction Type`,`Amount Transferred To Acc
          VALUES
          (@custId,NOW(),'Amount Transfer',accTo,accFrom,amount,@transactDetails);
        COMMIT;
        SELECT 'Transaction Completed Successfully.';
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT ='The remaining amount is less than the minimum balance in the account!';
    END IF;
END
$$
```

```sql
call transferInterAccAmount(1000,1234569872,4558223682);

SELECT * FROM Bank_Transactions;
```

Result Grid

| TransactionId | CustomerId | Transaction DateTime | Transaction Type | Amount Transferred To Account | Amount Transferred From Account | Transaction Amount | Transaction Details |
|---|---|---|---|---|---|---|---|
| 1 | 13 | 2017-12-13 05:27:30 | Withdraw | 1234569872 | NULL | 1000 | Amount withdrawed. |
| 2 | 13 | 2017-12-13 05:27:30 | Deposit | 1234569872 | NULL | 10000 | Amount deposited. |
| 3 | 1 | 2017-12-13 05:27:30 | Deposit | 4558223682 | NULL | 12000 | Amount deposited. |
| 4 | 1 | 2017-12-13 05:27:31 | Deposit | 4558223682 | NULL | 20000 | Amount deposited. |
| 5 | 1 | 2017-12-13 05:27:31 | Withdraw | 4558223682 | NULL | 4500 | Amount withdrawed. |
| 6 | 13 | 2017-12-13 05:35:44 | Amount Transfer | 4558223682 | 1234569872 | 1000 | Amount transferred. |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Result 7 | Bank_Transactions 8

## 3. creditcardTransaction

- The procedure “creditcardTransaction” is used to insert the values of the credit card transactions in the credit card transaction table.
- It validates the maximum limit of the credit card and allows the insertion in the table if the transaction amount doesn’t exceed the maximum available limit.


```mysql
DROP PROCEDURE IF EXISTS creditcardTransaction;
DELIMITER $$
CREATE PROCEDURE creditcardTransaction(IN amount INT, IN ccNum BIGINT(10))
BEGIN
    SET @amt = (SELECT `Maximum Limit`
                FROM CREDIT_CARD
                WHERE `CC Number` = ccNum);

    IF(amount < @amt) THEN
        SET @transactDetails ='Transaction Completed Successfully.';
        INSERT INTO credit_card_transactions
        (`CC Number`, `Transaction DateTime`,`Transaction Amount`,`Transaction Details`)
        VALUES
        (ccNum,NOW(),amount,@transactDetails);
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT ='The amount is greater than credit card limit offered.';
    END IF;
END
$$
```

```mysql
call creditcardTransaction(1000,12547993282);
call creditcardTransaction(1500,12547993282);
call creditcardTransaction(500,45789148268);
call creditcardTransaction(2000,745821479248);
call creditcardTransaction(3000,748591255558);
call creditcardTransaction(4000,748526588852);

Select * from credit_card_transactions;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| CC Number | TransactionId | Transaction DateTime | Transaction Amount | Transaction Details |
|---|---|---|---|---|
| 12547993282 | 1 | 2017-12-13 05:43:04 | 1000 | Transaction Completed Successfully. |
| 12547993282 | 2 | 2017-12-13 05:43:04 | 1500 | Transaction Completed Successfully. |
| 45789148268 | 3 | 2017-12-13 05:43:05 | 500 | Transaction Completed Successfully. |
| 745821479248 | 4 | 2017-12-13 05:43:05 | 2000 | Transaction Completed Successfully. |
| 748591255558 | 5 | 2017-12-13 05:43:05 | 3000 | Transaction Completed Successfully. |
| 748526588852 | 6 | 2017-12-13 05:43:06 | 4000 | Transaction Completed Successfully. |
| NULL | NULL | NULL | NULL | NULL |

credit_card_transactions 9

## 4. calculate_Loan

- This stored procedure is one of the main feature since it calculates the loan amount for customer when provided with loanId, customerId.
- It performs various calculations for validating the amount, calculating interest and the value is inserted in the loan_calculation table.


```
DROP PROCEDURE IF EXISTS calculate_loan;
DELIMITER $$
CREATE PROCEDURE calculate_loan(IN loan_Id INT,IN custId INT)
BEGIN
    SET @loanAmount = (SELECT `Loan Amount`
                       FROM Loan_Customer
                       WHERE LoanId = loan_Id AND CustomerId = custId);
    SET @startDate = (SELECT `Start Date`
                      FROM Loan_Customer
                      WHERE LoanId = loan_Id AND CustomerId = custId);
    SET @rateOfInterest = (SELECT `Rate Of Interest`
                           FROM loan
                           WHERE LoanId = loan_Id);
    SET @duration = (SELECT `Loan Duration`
                     FROM loan_customer
                     WHERE LoanId = loan_Id AND CustomerId = custId);
    SET @payment = (SELECT `Minimum Balance Restriction`
                    FROM Accounts
                    WHERE `Account Number` = (SELECT ac.`Account Number`
                                              FROM customer_accounts AS ca
                                              JOIN Accounts AS ac ON ca.`Account Number` = ac.`Account Number`
                                              WHERE ca.CustomerId = custId AND ac.`Account Type` = 'LoanAcc'));
    SET @accNum = (SELECT ac.`Account Number`
                   FROM customer_accounts AS ca
                   JOIN Accounts AS ac ON ca.`Account Number` = ac.`Account Number`
                   WHERE ca.CustomerId = custId AND ac.`Account Type` = 'LoanAcc');
    SET @mon = @startDate;

    SET @totalamount = @loanAmount;
```

```
    WHILE @totalamount > 0 DO
        SET @interest = (SELECT calculateInterest(@totalamount,@rateOfInterest,@duration));
        SET @redemption = @payment - @interest;

        IF(@totalamount > @redemption) THEN
            SET @totalamount = @totalamount - @redemption;
        ELSE
            SET @redemption = @totalamount;
            SET @totalAmount = 0;
        END IF;
        SET @mon = (SELECT ADDDATE(@mon, INTERVAL '1' MONTH));

        INSERT INTO Loan_Calculation
        (`Date`,LoanId,CustomerId,`Account Number`,Amount,Interest,Redemption,Payment)
        VALUES
        (@mon,loan_Id,custId,@accNum,ROUND(@totalamount,2),ROUND(@interest,2),ROUND(@redemption,2),@payment);
    END WHILE;
END
$$
```

```
call calculate_loan(4,7);
call calculate_loan(2,6);
call calculate_loan(1,2);
call calculate_loan(2,3);
call calculate_loan(4,4);
call calculate_loan(1,5);

select * from loan_calculation
order by loanid,customerid,`date`;
```

sult Grid | Filter Rows: | Edit: | Export/Import: | Wrap C

| Date | LoanId | CustomerId | Account Number | Amount | Interest | Redemption | Payment |
|---|---|---|---|---|---|---|---|
| 2016-02-01 | 1 | 2 | 5467874253 | 65729.17 | 729.17 | 4270.83 | 5000 |
| 2016-03-01 | 1 | 2 | 5467874253 | 61413.85 | 684.68 | 4315.32 | 5000 |
| 2016-04-01 | 1 | 2 | 5467874253 | 57053.57 | 639.73 | 4360.27 | 5000 |
| 2016-05-01 | 1 | 2 | 5467874253 | 52647.88 | 594.31 | 4405.69 | 5000 |
| 2016-06-01 | 1 | 2 | 5467874253 | 48196.3 | 548.42 | 4451.58 | 5000 |
| 2016-07-01 | 1 | 2 | 5467874253 | 43698.34 | 502.04 | 4497.96 | 5000 |
| 2016-08-01 | 1 | 2 | 5467874253 | 39153.53 | 455.19 | 4544.81 | 5000 |
| 2016-09-01 | 1 | 2 | 5467874253 | 34561.38 | 407.85 | 4592.15 | 5000 |
| 2016-10-01 | 1 | 2 | 5467874253 | 29921.4 | 360.01 | 4639.99 | 5000 |
| 2016-11-01 | 1 | 2 | 5467874253 | 25233.08 | 311.68 | 4688.32 | 5000 |
| 2016-12-01 | 1 | 2 | 5467874253 | 20495.92 | 262.84 | 4737.16 | 5000 |
| 2017-01-01 | 1 | 2 | 5467874253 | 15709.42 | 213.5 | 4786.5 | 5000 |
| 2017-02-01 | 1 | 2 | 5467874253 | 10873.06 | 163.64 | 4836.36 | 5000 |

calculation 10

## 5. bank_Statement

- This procedure is used to get the bank statement of the customer by providing customerid and time period between the transactions as input.
- It uses the concat_Name function to get the customer's first and last name concatenated along with the space.

```sql
DROP FUNCTION IF EXISTS concat_Name;
CREATE FUNCTION concat_Name (firstName VARCHAR(50), lastName VARCHAR(50))
RETURNS VARCHAR(100)
RETURN CONCAT_WS(' ',firstName, lastName);

-- to get the bank statement of the customer between particular date
DROP PROCEDURE IF EXISTS bank_Statement;
DELIMITER $$
CREATE PROCEDURE bank_Statement(IN startDate DATE, IN endDate DATE, IN custId int)
BEGIN
SELECT cust.CustomerId, concat_Name(per.FirstName, per.LastName) AS 'FULL Name', ca.`Account Number`,
        bt.`Transaction DateTime`, bt.`Transaction Amount`, bt.`Transaction Type`, bt.`Amount Transferred To Accour
FROM Customer AS cust
    INNER JOIN  Person AS per ON per.PersonId = cust.PersonId
    INNER JOIN Customer_Accounts AS ca ON ca.CustomerId = cust.CustomerId
    LEFT JOIN Bank_Transactions AS bt ON bt.CustomerId = cust.CustomerId
WHERE cust.CustomerId = custId AND (DATE(bt.`Transaction DateTime`) BETWEEN startdate AND endDate);
END
$$
```

```
call bank_statement ('2017-11-15','2017-12-15',13);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| CustomerId | FULL Name | Account Number | Transaction DateTime | Transaction Amount | Transaction Type | Amount Transferred To Account |
|---|---|---|---|---|---|---|
| 13 | Rahanik Vora | 1234569872 | 2017-12-13 14:46:14 | 1000 | Withdraw | 1234569872 |
| 13 | Rahanik Vora | 1234569872 | 2017-12-13 14:46:18 | 10000 | Deposit | 1234569872 |
| 13 | Rahanik Vora | 1234569872 | 2017-12-13 14:46:29 | 1000 | Amount Transfer | 4558223682 |

## TRIGGERS:

1. check_loanAmount

- It is triggered before insertion on loan_Customer that checks whether the row inserted has loan amount that does not exceed the maximum loan that can be given to the person.

```
DROP TRIGGER IF EXISTS check_loanAmount;
DELIMITER $$
    CREATE TRIGGER check_loanAmount
    BEFORE INSERT ON Loan_Customer
        FOR EACH ROW
            BEGIN
                SET @amount = (SELECT `Loan Amount Limit` FROM Loan WHERE LoanId = new.LoanId);
                IF(new.`Loan Amount` > @amount) THEN
                    BEGIN
                        SIGNAL SQLSTATE '45000'
                        SET MESSAGE_TEXT = 'The Amount limit exceeds the loan amount that can be provided.';
                    END;
                END IF;
            END
$$
```

```
insert into loan_customer values
(1,6,4517896325,12,'2016-01-01',900000,'NO');
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 06:16:08 | insert into loan_customer values (1,6,4517896325,12,'2016-01-01',900000,'NO') | Error Code: 1644. The Amount limit exceeds the loan amount that can be provided. |

## 2. inactiveEmp

- If the employee is made inactive delete it from branch employee and it add it in branch employee history.

```
DROP TRIGGER IF EXISTS inactiveEmp; -- If the employee is made inactive delete it from branch employee
DELIMITER $$
CREATE TRIGGER inactiveEmp
BEFORE UPDATE ON Employee
FOR EACH ROW
BEGIN
    IF new.`IsActive?` = 'YES' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You cannot make an inactive employee active.';
    ELSE
        IF (new.`IsActive?` != OLD.`IsActive?` && new.`IsActive?` = 'NO')THEN
            DELETE
            FROM Branch_Employees
            WHERE EmployeeID IN (SELECT EmployeeId
                                 FROM Employee
                                 WHERE new.`IsActive?` != old.`IsActive?` && new.`IsActive?` = 'NO'
                                 && EmployeeId = OLD.EmployeeId);
        END IF;
    END IF;
END
$$
```

```
update employee
set `isActive?` = 'NO'
where employeeid= 15

update employee
set `IsActive?` = 'YES'
where employeeid= 15
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 06:16:08 | insert into loan_customer values (1,6,4517896325,12,'2016-01-01',900000,'NO') | Error Code: 1644. The Amount limit exceeds the loan amount that can be |
| 2 | 06:20:43 | DROP TRIGGER IF EXISTS inactiveEmp | 0 row(s) affected, 1 warning(s): 1360 Trigger does not exist |
| 3 | 06:20:43 | CREATE TRIGGER inactiveEmp BEFORE UPDATE ON Employee FOR EACH ROW BEGIN IF ne... | 0 row(s) affected |
| 4 | 10:51:24 | select * from employees LIMIT 0, 1000 | Error Code: 1146. Table 'bankdb2.employees' doesn't exist |
| 5 | 10:51:31 | select * from employee LIMIT 0, 1000 | 29 row(s) returned |
| 6 | 10:52:43 | update employee set `isActive?` = 'NO' where employeeid= 15 | 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 |
| 7 | 10:54:41 | update employee set `IsActive?` = 'YES' where employeeid= 15 | Error Code: 1644. You cannot make an inactive employee active. |

3. branchUpdate

- If the employee changes from one branch to another then the entry of the old branch where he used to work and the other details are entered in branch_employee_history table which has all the history of the previous employees and branch.
- If the date of joining is not updated then it gives error and Branch_Employee table is not updated.

```
CREATE TRIGGER branchUpdate
BEFORE UPDATE ON BRANCH_EMPLOYEES
FOR EACH ROW
BEGIN
    IF(New.`Date Of Joining` != OLD.`Date Of Joining`) THEN
    SET @designation = (SELECT Designation
                        FROM Employee
                        WHERE EmployeeId = old.EmployeeId);
    INSERT INTO BRANCH_EMPLOYEES_HISTORY
    SET Previous_BranchId = OLD.BranchId,
        EmployeeId = OLD.EmployeeId,
        `Start Date` = OLD.`Date Of Joining`,
        `End Date` = curdate(),
        Designation = @designation;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Please update the date of Joining.';
    END IF;
END
$$
```

```
update branch_employees
set branchId = 2
where employeeid=1
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 11:16:18 | update branch_employees set branchId = 2 where employeeid=1 | Error Code: 1644. Please update the date of Joining. |

## 4. branchDelete

- If employee is deleted from particular branch in branch_Employee table then she/he should be inserted into branch_employee_history table.

```
DROP TRIGGER IF EXISTS branchDelete;
DELIMITER $$
CREATE TRIGGER branchDelete
BEFORE DELETE ON BRANCH_EMPLOYEES
FOR EACH ROW
BEGIN
    SET @designation = (SELECT Designation
                        FROM Employee
                        WHERE EmployeeId = old.EmployeeId);
    INSERT INTO BRANCH_EMPLOYEES_HISTORY
    SET Previous_BranchId = OLD.BranchId,
        EmployeeId = OLD.EmployeeId,
        `Start Date` = OLD.`Date Of Joining`,
        `End Date` = curdate(),
        Designation = @designation;
END
$$
```

```sql
delete from branch_employees
where employeeid = 16

select * from branch_employees_history
```

Result Grid | Filter Rows: | Edit: | Export/

| Previous_BranchId | EmployeeId | Start Date | End Date | Designation |
|---|---|---|---|---|
| 5 | 14 | 2006-02-28 | 2017-12-13 | Branch Manager |
| 3 | 15 | 2010-08-16 | 2017-12-13 | Bank Teller |
| 4 | 16 | 2011-04-25 | 2017-12-13 | Bank Teller |
| NULL | NULL | NULL | NULL | NULL |

branch_employees_history 19

Output

Action Output

| # | Time | Action |
|---|---|---|
| 1 | 11:28:51 | delete from branch_employees where employeeid = 16 |
| 2 | 11:28:55 | select * from branch_employees_history LIMIT 0, 1000 |

## 5. inactiveAccount

- While updating accounts you cannot set the flag of inactive accounts to active.
- Once the account is closed the person cannot re-open it, he/she has to open a new account.

```
DROP TRIGGER IF EXISTS inactiveAccount;
DELIMITER $$
CREATE TRIGGER inactiveAccount
BEFORE UPDATE ON Accounts
FOR EACH ROW
BEGIN
    IF (new.`IsActive?` = 'YES' && OLD.`IsActive?` = 'NO') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You cannot activate closed account.';
    ELSE
        IF (new.`IsActive?` != OLD.`IsActive?` && new.`IsActive?` = 'NO')THEN
                SET new.`Date Of Closing` = CURDATE();
                DELETE
                FROM CUSTOMER_ACCOUNTS
                WHERE `Account Number` IN (SELECT `Account Number`
                                           FROM Accounts
                                           WHERE new.`IsActive?` != old.`IsActive?` && new.`IsActive?` = 'NO'
                                           && `Account Number` = OLD.`Account Number`);
            END IF;
    END IF;
END
$$
```

```sql
select * from accounts
where `isactive?`='NO';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| Account Number | Account Type | BranchId | Date Of Opening | Account Balance | Minimum Balance Restriction | IsActive? | Date Of Closing |
|---|---|---|---|---|---|---|---|
| 5478963112 | Checking | 3 | 2014-10-15 | 85000 | 1600 | NO | 2017-12-13 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

```sql
update accounts
set `isactive?` = 'YES'
where `account number` = 5478963112;
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 11:39:10 | update accounts set `isactive?` = 'YES' where `account number` = 5478963112 | Error Code: 1644. You cannot activate closed account. |

## 6. check_loanAmount

- While inserting row in the loan customer table i.e. while granting loan to customer the amount is checked against the maximum amount of loan that can be given.

```
DROP TRIGGER IF EXISTS check_loanAmount;
DELIMITER $$
CREATE TRIGGER check_loanAmount
BEFORE INSERT ON LOAN_CUSTOMER
FOR EACH ROW
BEGIN
    SET @amtLimit = (SELECT `Loan Amount Limit`
                     FROM Loan
                     WHERE new.LoanId = LoanId);
    IF (new.`Loan Amount` > @amtLimit) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The loan amount exceed given loan limit.';
    END IF;
END
$$
```

```sql
select * from loan;
```

Result Grid | Filter Rows: | Edit:

| LoanId | LoanType | Rate Of Interest | Loan Amount Limit |
|---|---|---|---|
| 1 | Secured Education | 12.5 | 500000 |
| 2 | Secured Housing | 10.5 | 1000000 |
| 3 | Secured Vehicles | 12.75 | 300000 |
| 4 | Secured Personal | 11.25 | 700000 |
| 5 | Unsecured Education | 13.5 | 400000 |
| 6 | Unsecured Housing | 11.5 | 900000 |
| 7 | Unsecured Vehicles | 13.25 | 200000 |
| 8 | Unsecured Mortgage | 12.25 | 500000 |
| NULL | NULL | NULL | NULL |

```sql
insert into loan_customer
(LoanId,CustomerId,`Account Number`,`Loan Duration`,`Start Date`,`Loan Amount`,`Loan Transferred`)
values
(1,6,7458585547,12,'2017-12-12',600000,'NO');
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 11:47:05 | select * from loan LIMIT 0, 1000 | 8 row(s) returned |
| 2 | 11:53:04 | select * from loan_customer LIMIT 0, 1000 | 6 row(s) returned |
| 3 | 11:55:10 | insert into loan_customer (LoanId,CustomerId,`Account Number`,`Loan Duration`,`Start Date`,`Loan... | Error Code: 1644. The loan amount exceed given loan limit. |

## 7. check_OnupdateloanAmount

- While updating the existing record for loan amount it checks that the loan amount does not exceed the maximum amount that can be provided for a particular loan type.

```
DROP TRIGGER IF EXISTS check_OnupdateloanAmount;
DELIMITER $$
CREATE TRIGGER check_OnupdateloanAmount
BEFORE UPDATE ON LOAN_CUSTOMER
FOR EACH ROW
BEGIN
    SET @amtLimit = (SELECT `Loan Amount Limit`
                     FROM Loan
                     WHERE new.LoanId = LoanId);
    IF (new.`Loan Amount` > @amtLimit) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The loan amount exceed given loan limit.';
    END IF;
END
$$
```

```
select * from loan_customer
```

Result Grid

| LoanId | CustomerId | Account Number | Loan Duration | Start Date | Loan Amount | Loan Transferred |
|---|---|---|---|---|---|---|
| 1 | 2 | 5467874253 | 12 | 2016-01-01 | 70000 | NO |
| 1 | 5 | 4517896325 | 12 | 2017-06-06 | 55000 | NO |
| 2 | 3 | 7458585547 | 18 | 2016-01-01 | 80000 | NO |
| 2 | 6 | 7458585547 | 12 | 2017-01-01 | 50000 | NO |
| 4 | 4 | 7489544933 | 12 | 2016-01-01 | 55000 | NO |
| 4 | 7 | 7489544933 | 24 | 2016-09-12 | 60000 | NO |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

```
update loan_customer
set `Loan Amount` = 7000000
where loanId = 1 and customerId = 2;
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 12:02:20 | update loan_customer set `Loan Amount` = 7000000 where loanId = 1 and customerId = 2 | Error Code: 1644. The loan amount exceed given loan limit. |

## 8. loanTransfer

- The trigger is called when the loan_customer table is updated with new customerId i.e. when the loan amount is transferred to different person the details are added to loan_transfer table.
- It performs the check before updating the loan transfer such as the person to whom loan is transferred does not have the same loan with him/her already.
- If the person does not have loan the values are inserted into the loan transfer table.

```
DROP TRIGGER IF EXISTS loanTransfer;
DELIMITER $$
CREATE TRIGGER  loanTransfer
BEFORE UPDATE ON LOAN_CUSTOMER
FOR EACH ROW
BEGIN
    IF(new.CustomerId != OLD.CustomerId) THEN
        SET @checkLoanExistsWithCustomer = (SELECT 'True'
                                            FROM Loan_Customer
                                            WHERE OLD.CustomerId = new.CustomerId AND OLD.LoanId = new.LoanId);
        SET @custId = OLD.CustomerId;
        SET @loan_Id = OLD.LoanId;

        IF (@checkLoanExistsWithCustomer) THEN
           SIGNAL SQLSTATE '45000'
           SET MESSAGE_TEXT = 'The Customer already has the same loan so cannot be transferred.';
        ELSE
           SET @startdate = OLD.`Start Date`;
           SET @actualdate = (SELECT CURDATE());
           SET @duration = OLD.`Loan Duration`;
           SET @monNum = (SELECT TIMESTAMPDIFF(MONTH,@startdate,@actualdate));
           SET @duartionDiff = @duration - @monNum;
           SET @getMon = (SELECT ADDDATE(@startdate, INTERVAL @monNum MONTH));
           SET @getAmount = (SELECT Amount
                             FROM Loan_Calculation as lc
                             WHERE lc.`Date` = @getMon AND (lc.CustomerId = @custId AND lc.LoanId = @loan_Id));

           SET NEW.`Loan Transferred` =  'YES';
           SET new.`Start date` = @actualdate;
           SET new.`Loan Duration` = @duartionDiff;
           SET new.`Loan Amount` = @getAmount;

          INSERT INTO LOAN_TRANSFER
          (LoanId,`Loan Transferred From`,`Loan Transferred To`,`Account Number`,Duration,`Date Of Transfer`,`Amount Tra
          VALUES
          (OLD.LoanId, OLD.CustomerId, NEW.CustomerId,OLD.`Account Number`,OLD.`Loan Duration`,@actualdate,@getAmount,OLI
       END IF;
    END IF;
END
$$
```

```
3 •  select * from loan_customer
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

| LoanId | CustomerId | Account Number | Loan Duration | Start Date | Loan Amount | Loan Transferred |
|---|---|---|---|---|---|---|
| 1 | 2 | 5467874253 | 12 | 2016-01-01 | 70000 | NO |
| 1 | 5 | 4517896325 | 12 | 2017-06-06 | 55000 | NO |
| 2 | 3 | 7458585547 | 18 | 2016-01-01 | 80000 | NO |
| 2 | 6 | 7458585547 | 12 | 2017-01-01 | 50000 | NO |
| 4 | 4 | 7489544933 | 12 | 2016-01-01 | 55000 | NO |
| 4 | 7 | 7489544933 | 24 | 2016-09-12 | 60000 | NO |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

```sql
update loan_customer
set customerId = 7
where loanId = 1 and customerId = 2;

select * from loan_customer
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| LoanId | CustomerId | Account Number | Loan Duration | Start Date | Loan Amount | Loan Transferred |
|---|---|---|---|---|---|---|
| 1 | 5 | 4517896325 | 12 | 2017-06-06 | 55000 | NO |
| 1 | 7 | 5467874253 | 12 | 2016-01-01 | 70000 | NO |
| 2 | 3 | 7458585547 | 18 | 2016-01-01 | 80000 | NO |
| 2 | 6 | 7458585547 | 12 | 2017-01-01 | 50000 | NO |
| 4 | 4 | 7489544933 | 12 | 2016-01-01 | 55000 | NO |
| 4 | 7 | 7489544933 | 24 | 2016-09-12 | 60000 | NO |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

loan_customer 32 ×  Apply

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 12:17:52 | select * from loan_customer LIMIT 0, 1000 | 6 row(s) returned |
| 2 | 12:19:26 | update loan_customer set customerId = 5 where loanId = 1 and customerId = 7 | 0 row(s) affected Rows matched: 0 Changed: 0 Warnings: 0 |
| 3 | 12:19:32 | select * from loan_customer LIMIT 0, 1000 | 6 row(s) returned |
| 4 | 12:21:29 | update loan_customer set customerId = 7 where loanId = 1 and customerId = 2 | 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 |
| 5 | 12:21:39 | select * from loan_customer LIMIT 0, 1000 | 6 row(s) returned |

# FUNCTIONS

1. concat_Name(firstName VARCHAR(50), lastName VARCHAR(50))

- This function concats the first and last name of person and is used in lot of queries and stored procedures.

```
DROP FUNCTION IF EXISTS concat_Name;
CREATE FUNCTION concat_Name (firstName VARCHAR(50), lastName VARCHAR(50))
RETURNS VARCHAR(100)
RETURN CONCAT_WS(' ',firstName, lastName);
```

```
select concat_Name('John','Jose') AS 'Full Name';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Full Name |
|---|
| John Jose |

## 2. calculateInterest(loanAmt Double, roi Double, duration INT)

- This function calculates the interest and returns the result.
- It is called from calculate_loan stored procedure.

```
DROP FUNCTION IF EXISTS calculateInterest;
DELIMITER $$
CREATE FUNCTION calculateInterest(loanAmt Double, roi Double, duration INT)
RETURNS DOUBLE
BEGIN
    RETURN  (loanAmt*(roi/100)*(duration/12))/12;
END
$$
```

```
select calculateInterest(20000,10.5,12) AS 'Interest';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Interest |
|---|
| 175 |

# VIEWS

1. To view the total number of weekly transactions per branch.

```sql
-- To get total number of transactions per branch
DROP VIEW IF EXISTS view_Weekly_Transactions_Per_Branch;
CREATE VIEW view_Weekly_Transactions_Per_Branch AS
    SELECT ac.BranchId, COUNT(bt.TransactionId) AS 'Total Transactions', SUM(bt.`Transaction Amount`) AS 'Total Amount'
    FROM Bank_Transactions AS bt JOIN Accounts AS ac ON ac.`Account Number` = bt.`Amount transferred To Account`
    WHERE (DATE(`Transaction DateTime`) <= CURDATE() AND DATE(`Transaction DateTime`) >= SUBDATE(CURDATE(), INTERVAL '7' DAY))
    GROUP BY ac.BranchId;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| BranchId | Total Transactions | Total Amount |
|---|---|---|
| 1 | 2 | 11000 |
| 3 | 4 | 37500 |

2. To view the total accounts opened per branch quarterly.

```sql
-- To get total accounts opened by bank per branch quaterly
DROP VIEW IF EXISTS view_Quaterly_Accounts_Opened;
CREATE VIEW view_Quaterly_Accounts_Opened AS
    SELECT BranchId, COUNT(`Account Type`)
    FROM Accounts
    WHERE `IsActive?` = 'YES' AND
        `Date Of Opening` <= CURDATE() AND  `Date Of Opening` >= SUBDATE(CURDATE(), INTERVAL '3' MONTH);

SELECT * FROM view_Quaterly_Accounts_Opened;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| BranchId | COUNT(`Account Type`) |
|---|---|
| 2 | 2 |

# 3. To calculate total profit made by the bank on loan interest per branch.

```
-- To calculate total profit made by bank based on loan interest per branch
DROP VIEW IF EXISTS view_Bank_Profit_Per_Branch_Annually;
CREATE VIEW view_Bank_Profit_Per_Branch_Annually AS
    SELECT ac.BranchId AS BranchId,l.LoanType, ROUND(SUM(lc.Interest),2) AS 'Total Interest'
    FROM loan_calculation AS lc
        INNER JOIN accounts AS ac ON ac.`Account Number` = lc.`Account Number`
        LEFT JOIN loan AS l ON l.loanid = lc.loanid
    WHERE lc.`Date` <= CURDATE() AND lc.`Date` >=  SUBDATE(CURDATE(), INTERVAL '1' YEAR)
    GROUP BY ac.BranchId,l.loanType WITH ROLLUP;

SELECT * FROM view_Bank_Profit_Per_Branch_Annually;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| BranchId | LoanType | Total Interest |
|---|---|---|
| 1 | Secured Education | 3211.39 |
| 1 | Secured Housing | 1806.02 |
| 1 | Secured Personal | 11678.70 |
| 1 | NULL | 16696.11 |
| 2 | Secured Education | 563.68 |
| 2 | Secured Housing | 2558.14 |
| 2 | Secured Personal | 3211.79 |
| 2 | NULL | 6333.61 |
| NULL | NULL | 23029.72 |

# 4. To get the total number of loans granted to customers based on loan type annually. i.e. to get the most popular loan among customers.


```sql
-- To get total loan type count by customers per branch per year
DROP VIEW IF EXISTS view_TotalCustomers_PerBranch_Annually;
CREATE VIEW view_TotalCustomers_PerBranch_Annually AS
    SELECT ac.BranchId, l.LoanType ,COUNT(lc.CustomerId) AS 'Total Customers', SUM(lc.`Loan Amount`) AS 'Total Loar
    FROM loan_Customer AS lc
        INNER JOIN loan AS l ON l.loanId = lc.LoanId
        LEFT JOIN Accounts AS ac ON lc.`Account Number` = ac.`Account Number`
    WHERE CURDATE() >= SUBDATE(CURDATE(), INTERVAL'1' YEAR)
    GROUP BY ac.BranchId, l.LoanType;

 SELECT * FROM view_TotalCustomers_PerBranch_Annually;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

| BranchId | LoanType | Total Customers | Total Loan Amount |
|---|---|---|---|
| 1 | Secured Education | 1 | 55000 |
| 1 | Secured Personal | 2 | 115000 |
| 2 | Secured Education | 1 | 70000 |
| 2 | Secured Housing | 2 | 130000 |

Result Grid

Form Editor

# 5. To get details of all the employees who are also the customers of the same bank.

```sql
-- TO get the list of employees who are also the customers of the same bank
DROP VIEW IF EXISTS view_Employees_Customers;
CREATE VIEW view_Employees_Customers AS
    SELECT emp.EmployeeId,concat_Name(per.FirstName,per.LastName) AS 'Full Name', emp.Designation,
            ac.BranchId, ac.`Account Number`,ac.`Account Type`
    FROM Employee AS emp
        INNER JOIN Person AS per ON per.PersonId = emp.PersonId
        INNER JOIN Customer AS cust ON cust.PersonId = per.PersonId
        LEFT JOIN Customer_Accounts AS ca ON ca.CustomerId = cust.CustomerId
        LEFT JOIN Accounts AS ac ON ac.`Account Number` = ca.`Account Number`
    WHERE emp.`IsActive?` = 'YES';

SELECT * FROM view_Employees_Customers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| EmployeeId | Full Name | Designation | BranchId | Account Number | Account Type |
|---|---|---|---|---|---|
| 1 | Amar Desai | Bank Teller | 2 | 8547963326 | Checking |
| 15 | Paresh Soni | Bank Teller | 1 | 8574963214 | Checking |
| 16 | Rahanik Vora | Bank Teller | 1 | 1234569872 | Checking |
| 17 | Krishna Gandhi | Bank Clerk | 2 | 4587215665 | Checking |
| 18 | Kavita Shah | Bank Clerk | 1 | 5485214755 | Checking |
| 19 | Reshma Mulla | Bank Teller | 2 | 5478992582 | Checking |

# 6. To get the details of the customers with more than one account.

```sql
-- To get the list of customers having more than on account in bank branch
DROP VIEW IF EXISTS view_CustomersWithMoreThan_One_Account;
CREATE VIEW view_CustomersWithMoreThan_One_Account AS
    SELECT cust.CustomerId,concat_Name(per.FirstName, per.LastName) AS 'Full Name', ac.BranchId,
           ca.`Account Number`,ac.`Account Type`
    FROM Customer AS cust
        INNER JOIN Person AS per ON per.PersonId = cust.PersonId
        INNER JOIN Customer_Accounts AS ca ON ca.CustomerId = cust.CustomerId
        LEFT JOIN Accounts AS ac ON ac.`Account Number` = ca.`Account Number`
    WHERE ca.CustomerId IN (SELECT CustomerId
                            FROM Customer_Accounts
                            GROUP BY CustomerId
                            HAVING COUNT(CustomerId)>1)
    ORDER BY cust.CustomerId;

SELECT * FROM view_CustomersWithMoreThan_One_Account;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| CustomerId | Full Name | BranchId | Account Number | Account Type |
|---|---|---|---|---|
| 1 | Manish Kamani | 1 | 1236547895 | Checking |
| 1 | Manish Kamani | 3 | 4558223682 | Savings |
| 2 | Parshva Shah | 2 | 3654485225 | Checking |
| 2 | Parshva Shah | 2 | 5467874253 | LoanAcc |
| 2 | Parshva Shah | 1 | 7459631258 | Savings |

# 7. To get the total interest the customer pays on the loan.

```
-- To let the amount of interest the customer pays on loan for particular amount and period
DROP VIEW IF EXISTS view_TotalInterest_Customer_Paid;
CREATE VIEW view_TotalInterest_Customer_Paid AS
    SELECT cust.CustomerId, concat_Name(per.FirstName, per.LastName) AS 'Full Name',
           l.loanId,l.LoanType, lc.`Loan Amount`, lc.`Loan Duration`, ROUND(SUM(lcal.Interest),2) AS 'Total Intere
    FROM customer AS cust
        INNER JOIN loan_customer AS lc ON lc.CustomerId = cust.CustomerId
        INNER JOIN Person AS per ON per.PersonId = cust.PersonId
        LEFT JOIN Loan AS l ON l.LoanId = lc.LoanId
        LEFT JOIN loan_calculation AS lcal ON lcal.LoanId = lc.LoanId and lc.CustomerId = lcal.CustomerId
    GROUP By cust.CustomerId, 'Full Name', l.loanId, l.LoanType, lc.`Loan Amount`, lc.`Loan Duration`;

SELECT * FROM view_TotalInterest_Customer_Paid;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

| CustomerId | Full Name | loanId | LoanType | Loan Amount | Loan Duration | Total Interest |
|---|---|---|---|---|---|---|
| 2 | Parshva Shah | 1 | Secured Education | 70000 | 12 | 6059.60 |
| 3 | Henah Montev | 2 | Secured Housing | 80000 | 18 | 10389.35 |
| 4 | Khushali Mehta | 4 | Secured Personal | 55000 | 12 | 8904.16 |
| 5 | Harsh Jain | 1 | Secured Education | 55000 | 12 | 10142.25 |
| 6 | Varsha Kulkarni | 2 | Secured Housing | 50000 | 12 | 2558.14 |

Result Grid

# 8. To get the credit card and customer details having more than 1 credit card.

```sql
-- To get the details of the customer and credit card having more than 1 credit card
DROP VIEW IF EXISTS view_CustomerWithMoreThan_OneCC;
CREATE VIEW view_CustomerWithMoreThan_OneCC AS
    SELECT cc.CustomerId,concat_Name(per.FirstName,per.LastName) AS 'Full Name', cc.`CC Number`,
           cc.`Date of Activation`, cc.`Expiry Date`, cc.`Credit Score`, cc.`Maximum Limit`
    FROM credit_card AS cc
         INNER JOIN Customer AS cust ON cust.CustomerId = cc.CustomerId
         LEFT JOIN Person  AS per ON per.PersonId = cust.PersonId
    WHERE cc.CustomerId IN (SELECT CustomerId
                            FROM credit_card
                            GROUP BY CustomerId
                            HAVING COUNT(CustomerId) > 1)
    ORDER BY cc.CustomerId;

SELECT * FROM view_CustomerWithMoreThan_OneCC;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

| CustomerId | Full Name | CC Number | Date of Activation | Expiry Date | Credit Score | Maximum Limit |
|---|---|---|---|---|---|---|
| 5 | Harsh Jain | 458757812821 | 2017-04-15 | 2028-09-14 | 600 | 175000 |
| 5 | Harsh Jain | 587496828525 | 2017-07-15 | 2029-09-14 | 600 | 2000000 |
| 7 | Parth Shah | 745821479248 | 2017-09-03 | 2027-02-23 | 650 | 100000 |
| 7 | Parth Shah | 785214785448 | 2017-03-03 | 2025-02-23 | 650 | 500000 |

# QUERIES

1. To get the count of the employees who joined in the current year seggregated by branch.

```sql
-- To get the count of the employees joined this year
SELECT be.BranchId,COUNT(be.EmployeeId) AS 'Total Employees Joined',
        YEAR(SUBDATE(CURDATE(),INTERVAL '1' YEAR)) AS 'Last Year' ,YEAR(CURDATE()) AS 'Current Year'
FROM Branch_Employees AS be
    INNER JOIN Employee AS emp ON emp.EmployeeId = be.EmployeeId
WHERE (YEAR(be.`Date Of Joining`) BETWEEN YEAR(SUBDATE(CURDATE(),INTERVAL '1' YEAR)) AND YEAR(CURDATE()))
    AND emp.`IsActive?`= 'YES'
GROUP BY BranchId;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| BranchId | Total Employees Joined | Last Year | Current Year |
|---|---|---|---|
| 2 | 1 | 2016 | 2017 |
| 5 | 1 | 2016 | 2017 |

2. To get the total count of credit card transactions and total amount of the transactions per week.

```sql
-- To get the total number of credit card transactions occuring per week
SELECT COUNT(transactionid) AS 'Total Transactions', SUM(`transaction Amount`) AS 'Total Amount'
FROM credit_card_transactions
WHERE (DATE(`Transaction DateTime`) <= CURDATE()
    AND  DATE(`Transaction DateTime`) >= SUBDATE(CURDATE(), INTERVAL '7' DAY));
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Total Transactions | Total Amount |
|---|---|
| 6 | 12000 |

## 3. To get the accounts closed by the bank quarterly per branch.

```sql
SELECT * FROM view_Quaterly_Accounts_Opened;

-- To get total accounts closed by bank per branch quaterly
SELECT BranchId, COUNT(`Account Type`) AS 'Total Accounts Closed'
FROM Accounts
WHERE `IsActive?` = 'NO' AND
    `Date Of Closing` <= CURDATE() AND  `Date Of Opening` >= SUBDATE(CURDATE(), INTERVAL '3' MONTH);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| BranchId | Total Accounts Closed |
|---|---|
| NULL | 0 |

## 4. To get the details of the credit cards sold in the past 1 month.

```sql
-- To get the details of the credit cards sold in the past 1 month
SELECT `CC Number`,CustomerId, `Maximum Limit`
FROM Credit_Card
WHERE `isActive?` = 'YES' AND
    (`Date Of Activation` <=  CURDATE() AND `Date Of Activation` >= SUBDATE(CURDATE(),INTERVAL '1' MONTH));
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| CC Number | CustomerId | Maximum Limit |
|---|---|---|
| 12547993282 | 2 | 50000 |
| 45789148268 | 3 | 50000 |
| 748591255558 | 10 | 150000 |
| NULL | NULL | NULL |

# 5. To get the details of the credit card that expires in the current month and year.

```
-- To get the credit card details that expires in the current month
SELECT `CC Number`,`Expiry Date`,CustomerId, `Maximum Limit`
FROM credit_card
WHERE `isActive?` = 'YES' AND
        MONTH(`Expiry Date`) = MONTH(CURDATE()) AND YEAR(`Expiry Date`) = YEAR(CURDATE());
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| CC Number | Expiry Date | CustomerId | Maximum Limit |
|---|---|---|---|
| 12365478924 | 2017-12-26 | 1 | 100000 |
| NULL | NULL | NULL | NULL |

# USERS & PRIVILEGES

- The users are created for roles of Bank Manager, Bank Teller, Loan Manager, Bank Clerk, Human Resource.
- Bank Manager is given access to entire database.
- Bank Clerk is given access to  select specific customer and their accounts related information as well as transactions done by the customer.
- Bank teller is given access to perform CRUD operations on tables Bank_Transactions, Customer_Accounts, Accounts as well as view, update the information of Customers and few procedures to perform transactions .
- Loan Manager is given access to all the loan, customer related tables and stored procedures as shown in the screenshot.

- Human Resource has access to perform CRUD opertaions on the Person, Employee tables.

```sql
CREATE USER 'Manager'@'localhost'
IDENTIFIED BY 'man';

CREATE USER 'Clerk'@'localhost'
IDENTIFIED BY 'clerk';

CREATE USER 'Teller'@'localhost'
IDENTIFIED BY 'teller';

CREATE USER 'LoanManager'@'localhost'
IDENTIFIED BY 'lman';

CREATE USER 'HR'@'localhost'
IDENTIFIED BY 'HR';
```

```sql
GRANT ALL ON bankdb.* TO 'Manager'@'localhost' WITH GRANT OPTION;

GRANT SELECT ON bankdb.Bank_Transactions
TO 'Clerk'@'localhost';

GRANT SELECT ON bankdb.Accounts
TO 'Clerk'@'localhost';

GRANT SELECT ON bankdb.Customer_Accounts
TO 'Clerk'@'localhost';

GRANT SELECT(PersonId,FirstName,LastName,`Date Of Birth`,Gender) ON bankdb.Person
TO 'Clerk'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Bank_Transactions
TO 'Teller'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Accounts
TO 'Teller'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Customer_Accounts
TO 'Teller'@'localhost';

GRANT SELECT, UPDATE ON bankdb.Customer
TO 'Teller'@'localhost';

GRANT SELECT, UPDATE  ON bankdb.Person
TO 'Teller'@'localhost';

GRANT EXECUTE ON PROCEDURE bankdb.bank_Statement
TO 'Teller'@'localhost';
```

```sql
GRANT EXECUTE ON PROCEDURE bankdb.beginTransaction
TO 'Teller'@'localhost';

GRANT EXECUTE ON PROCEDURE bankdb.transferInterAccAmount
TO 'Teller'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan
TO 'LoanManager'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan_Customer
TO 'LoanManager'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan_Calculation
TO 'LoanManager'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan_Transfer
TO 'LoanManager'@'localhost';

GRANT SELECT, UPDATE ON bankdb.Customer
TO 'LoanManager'@'localhost';

GRANT SELECT ON bankdb.Customer_Accounts
TO 'LoanManager'@'localhost';

GRANT SELECT  ON bankdb.Accounts
TO 'LoanManager'@'localhost';

GRANT EXECUTE ON PROCEDURE bankdb.calculate_loan
TO 'LoanManager'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Person
TO 'HR'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Employee
TO 'HR'@'localhost';
```

| | | | |
|---|---|---|---|
| 15 | 15:57:21 | CREATE USER 'Manager'@'localhost' IDENTIFIED BY 'man' | 0 row(s) affected |
| 16 | 15:57:39 | CREATE USER 'Clerk'@'localhost' IDENTIFIED BY 'clerk' | 0 row(s) affected |
| 17 | 15:58:17 | GRANT ALL ON bankdb.* TO 'Manager'@'localhost' WITH GRANT OPTION | 0 row(s) affected |
| 18 | 15:58:35 | GRANT SELECT ON bankdb.Bank_Transactions TO 'Clerk'@'localhost' | 0 row(s) affected |
| 19 | 16:00:18 | GRANT SELECT(PersonId,FirstName,LastName,`Date Of Birth`,Gender) ON bankdb.Person ... | 0 row(s) affected |
| 20 | 16:01:41 | CREATE USER 'Teller'@'localhost' IDENTIFIED BY 'teller' | 0 row(s) affected |
| 21 | 16:09:27 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Bank_Transactions TO 'Teller'... | 0 row(s) affected |
| 22 | 16:09:27 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Accounts TO 'Teller'@'localhost' | 0 row(s) affected |
| 23 | 16:09:28 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Customer_Accounts TO 'Teller'... | 0 row(s) affected |
| 24 | 16:09:28 | GRANT SELECT, UPDATE ON bankdb.Customer TO 'Teller'@'localhost' | 0 row(s) affected |
| 25 | 16:09:28 | GRANT SELECT, UPDATE ON bankdb.Person TO 'Teller'@'localhost' | 0 row(s) affected |
| 26 | 16:11:44 | GRANT EXECUTE ON PROCEDURE bankdb.bank_Statement TO 'Teller'@'localhost' | 0 row(s) affected |
| 27 | 16:11:44 | GRANT EXECUTE ON PROCEDURE bankdb.beginTransaction TO 'Teller'@'localhost' | 0 row(s) affected |
| 28 | 16:11:44 | GRANT EXECUTE ON PROCEDURE bankdb.transferInterAccAmount TO 'Teller'@'localhost' | 0 row(s) affected |
| 29 | 16:21:22 | select * from employee LIMIT 0, 1000 | 29 row(s) returned |
| 30 | 16:41:50 | CREATE USER 'LoanManager'@'localhost' IDENTIFIED BY 'man' | 0 row(s) affected |
| 31 | 16:41:50 | CREATE USER 'HR'@'localhost' IDENTIFIED BY 'HR' | 0 row(s) affected |
| 32 | 16:47:17 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan TO 'LoanManager'@'loc... | 0 row(s) affected |
| 33 | 16:47:17 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan_Customer TO 'LoanMan... | 0 row(s) affected |
| 34 | 16:47:17 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan_Calculation TO 'LoanMa... | 0 row(s) affected |
| 35 | 16:47:17 | GRANT SELECT, UPDATE, DELETE, INSERT ON bankdb.Loan_Transfer TO 'LoanMana... | 0 row(s) affected |

- The screenshot for the clerk user is attached on the next page and the other users also work in the similar way.

- The Clerk is given access to retrieve specific rows from the Person table.
- The Clerk user gets the following error when trying to access all the columns of Person table.

```
1 • SELECT * from Person;
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 16:53:20 | SELECT * from Person LIMIT 0, 1000 | Error Code: 1142. SELECT command denied to user 'Clerk'@'localhost' for table 'person' |

```
1 • SELECT FirstName,LastName, `Date Of Birth`, Gender
2   from Person;
```

Result Grid

| FirstName | LastName | Date Of Birth | Gender |
|---|---|---|---|
| Manish | Kamani | 1962-12-05 | Male |
| Parshva | Shah | 1993-06-05 | Female |
| Henah | Montev | 1982-09-15 | Female |
| Khushali | Mehta | 1953-08-18 | Female |
| Harsh | Jain | 1961-05-05 | Male |
| Varsha | Kulkarni | 1984-02-24 | Female |
| Parth | Shah | 1982-11-25 | Male |
| Kaushal | Dedhia | 1977-05-15 | Male |
| Viren | Gala | 1982-07-11 | Male |

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 16:53:20 | SELECT * from Person LIMIT 0, 1000 | Error Code: 1142. SELECT command denied to user 'Clerk'@'localhost' for table 'person' |
| 2 | 16:56:18 | SELECT FisrtName,LastName, `Date Of Birth`, Gender from Person LIMIT 0, 1000 | Error Code: 1143. SELECT command denied to user 'Clerk'@'localhost' for column 'FisrtName' i... |
| 3 | 16:56:26 | SELECT FirstName,LastName, `Date Of Birth`, Gender from Person LIMIT 0, 1000 | 40 row(s) returned |

# BACKUP

- Backup of the dump is taken every night at 9:28pm through Task Scheduler which automatically calls the BAT file to generate the backup at a particular path.
- The bat file is attached in the zip folder.