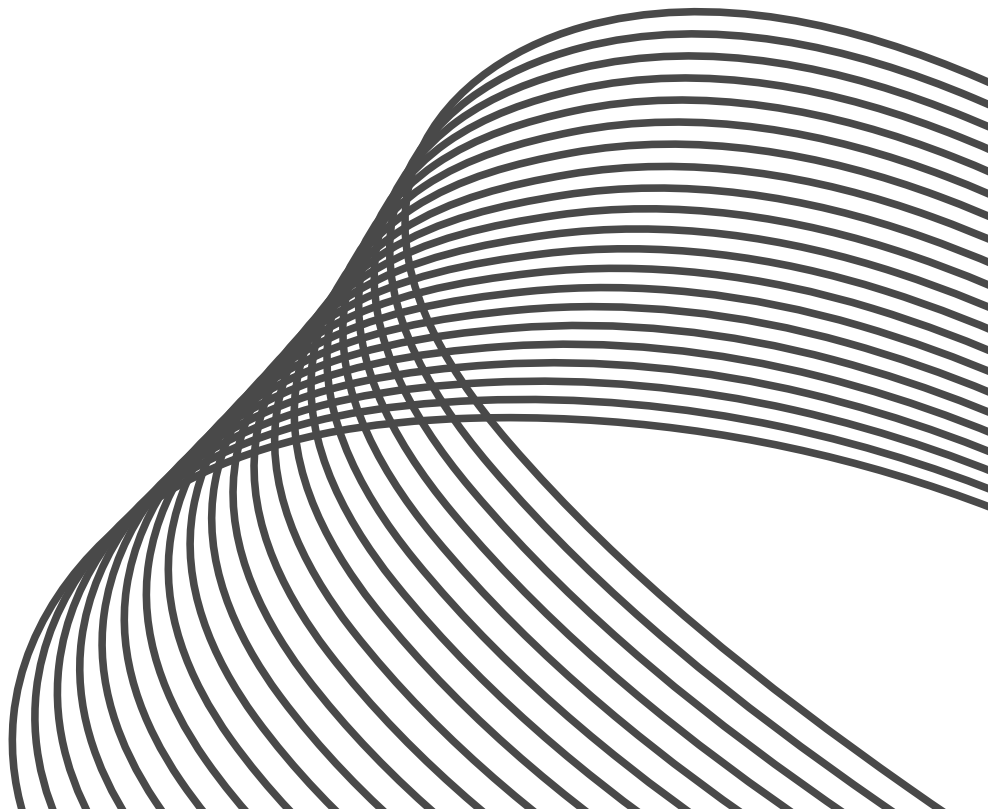
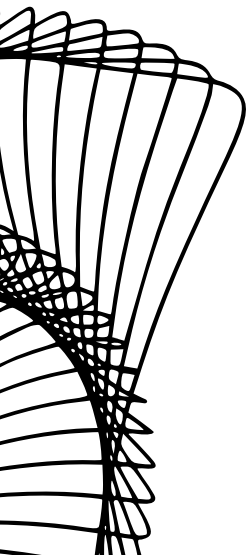


DBMS PROJECT

# BANK MANAGEMENT SYSTEM



# A PROJECT REPORT ON THE BANK MANAGEMENT SYSTEM



**COURSE: DATABASE MANAGEMENT SYSTEM (CSE1401)**

**SUBMITTED BY**

Name	Seat No	PRN
DHRUVCHANDRA DIPAKKUMAR BHATT	453003	8021076494
AAYUSH ASHISHBHAI DALAL	453010	8021057905
JAY AMRISH FANSE	453015	8021079917
BHAUMIK PIYUSH LODHIA	453026	8021058753
KARM DIPAKKUMAR SONI	453072	8021007953

# INDEX

Content	Page No.
Description	1
Assumptions	3
Entity Relationship Diagram	6
Tables	8
Description Of Normalization	12
DDLs	21
PROCEDURES	36
FUNCTIONS	59
TRIGGERS	64



# DESCRIPTION

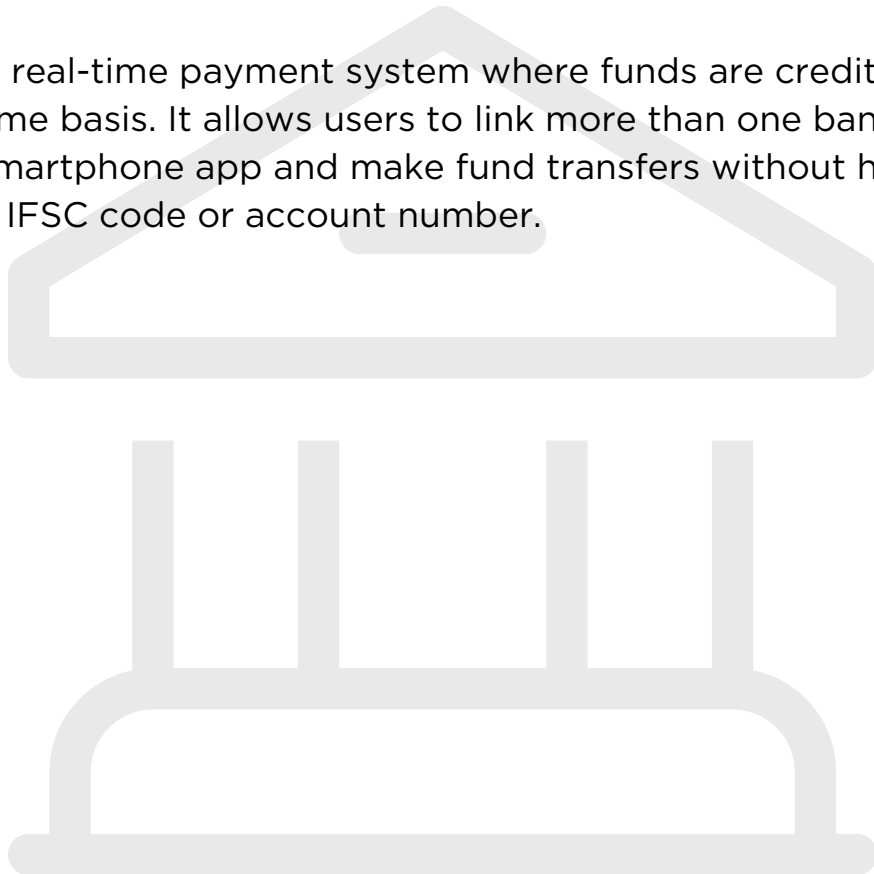
- **The BANK MANAGEMENT SYSTEM** is a software tool that models and manages financial activities connected with customer, banking activities, also connected with implementation of management functions in banking.
- The main aim of the project is provide customer with user account to carry various financial activities, model the functioning of a bank and to store all the data with minimum redundancy.
- This project maintains in detail the following functions of a bank
  - BRANCH, DEPARTMENT, EMPLOYEES, CUSTOMER
- This segment deals with all branches of a bank along with details of departments that a particular branch contains. The records of employees working in particular department of a branch and records of customer is also maintained. The customer is one who has one or more accounts in bank.
- **ACCOUNT AND TRANSACTION**
  - This section maintains the records of accounts that are opened in a bank. Customers are provided with various types of accounts such as savings, current, salary, PF. The transactions on accounts are also maintained efficiently and securely. Various types of payment methods available with customers are :- debit card, credit card, UPI, cheque.
- **FD, LOAN, AND LOCKER**
  - Customers can create FD account. FDs with variables intervals and variable interests are available. Further, details of loans borrowed by customer along with loan types are stored in database. The process of EMI deduction, pre- matured FD fine are also handled efficiently. Locker facility is one of the ancillary services provided by the Bank to its customers. Lockers Branches are equipped with high security features and specially built strong rooms. The details of available lockers, occupied lockers, along with operator i.e. a person who is in charge of a locker are stored.

- **DEBIT CARD, CREDIT CARD AND CHEQUE :-**

- A Debit Card is a plastic currency or plastic money. The banks issue a Debit Card and it is linked with the account. All transactions made through Debit Card are reflected in bank account statement. A credit card is a type of credit facility, provided by banks that allow customers to borrow funds within a pre-approved credit limit. The details of debit and credit card holders are maintained in the project's database. A cheque is a financial document that orders a bank to pay a particular amount of money from a person's account to another individual's or company's account in whose name the cheque has been made or issued. The details of issued cheque book, pending cheques, cheque bounce scenario ,etc. are handled with utmost efficiency and security.

- **UPI**

- This is a real-time payment system where funds are credited instantly on a real-time basis. It allows users to link more than one bank account in a single smartphone app and make fund transfers without having to provide IFSC code or account number.





# ASSUMPTIONS

- Every branch of the bank has a distinguishable IFSC code(datatype:varchar2) with pattern:
- 4 digit - bank name + 4 digit - branch no, branch no will be in order of opening of particular branch.
- All the departments present in all branches of the bank are listed in the Department table and every department is associated with department number.
- Branch may or may not be managed by at most 1 manager at particular instance compulsorily working in that branch only, similarly department of any branch may be headed by any employee working in that branch but 1 employee can head 1 or more department in branch in which he/she is working.
- Any customer registered in a bank must have a minimum 1 account in bank of type Savings, Current or salary.
- In case of a joint account there must be a main account holder and rate of interest depends upon type of account and major account holder.
- In case a minor(having age<18 years) is an account holder then it must be a joint account with at least 1 elder as joint account holder.
- Interest on a balance of account will be calculated on the basis of balance in account on a particular day and total interest of a month will be transferred in account at end of month as a transaction from bank to account.
- Maintenance charges and price on issuing a new checkbook will be deducted from the account directly after 1 year of opening the account, also as a transaction from account to bank.
- Penalty on failure of maintaining minimum balance from account will be deducted from account whenever balance overcomes minimum balance depending upon type of an account as transaction.
- PF account will be issued on salary account depending upon choice of customer.
- Account holders can issue a number of FDs on 1 account.



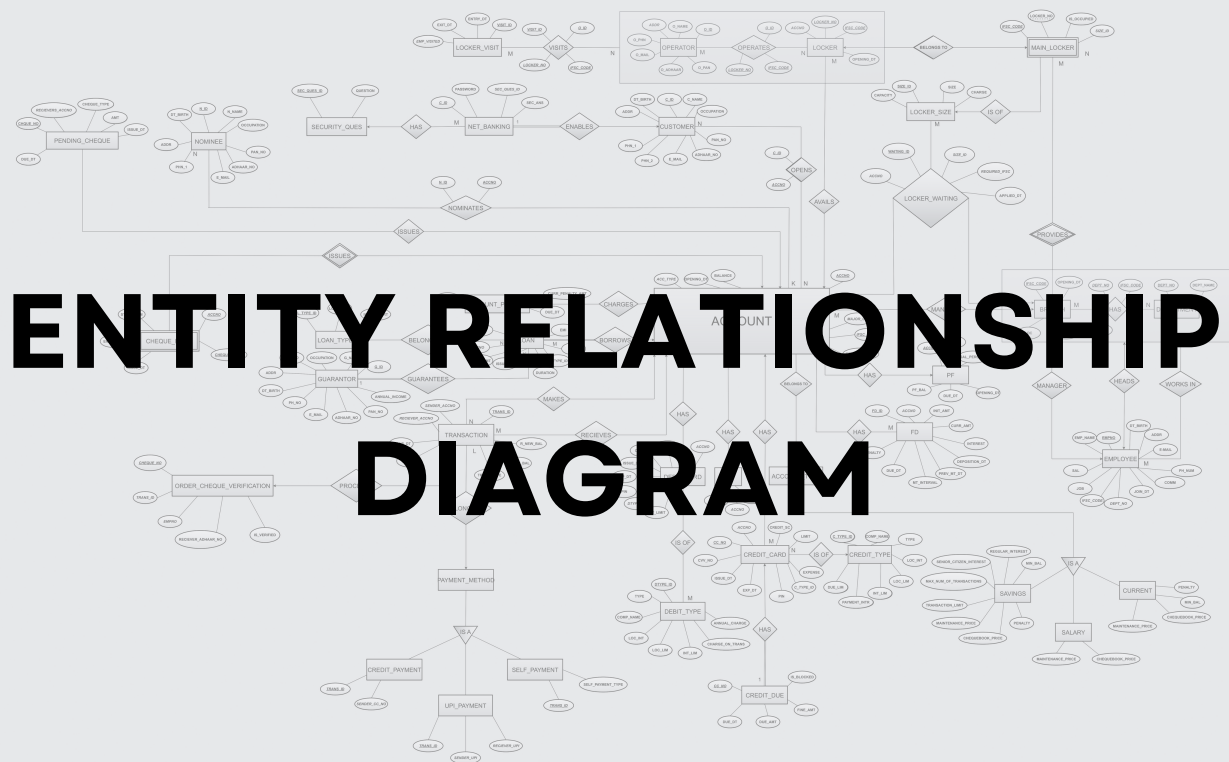
- Interest on FD will be deposited in FD depending upon maturity interval prescribed.
- If someone wants a loan from the bank then the person must have an account in the bank and such person can issue a number of loans and of types prescribed, Interest depends upon loan type.
- Every loan holder must have 1 and only 1 guarantor guaranteeing loan and the guarantor may or may not have an account in the bank and he/she can guarantee 1 or more loans.
- Number of lockers of different types in the number of branches can be owned by an account holder.
- One Operator can operate a number of lockers and One locker can be operated by a number of operators (many to many) not compulsorily be account holders.
- Lockers will be issued for an account holder on the basis of date applied(First come first serve).
- Only 1 debit card can be issued for 1 account and via debit card only account to account transfer is possible.
- Transaction of amount>limit of debit card will not be performed.
- Credit card will be inserted in credit\_due table if transaction of amt>limit of credit card will be performed or bill is not paid in time limit ,If any credit card is satisfying conditions of credit\_due then it will be blocked and can not be used for transaction before paying fine and bill.
- Via Credit card only credit card to account transfer is possible.
- Via UPI only UPI to UPI transfer is possible.
- All services payments between bank and customer(b2c) and Third party transactions(c2c) (Including cheques and self transfer also) will be stored in the transaction table.
- In one day only  $10^{30}$  maximum transactions are possible.
- Transaction id (datatype:number(38)) will be as of following pattered :  
ddmmyyyy+Transaction of that particular day
- Transaction will only be performed when the sender is having  
balance>amount of transaction in account.
- Cheque can be either order, barrer or account payee. In case of account payee cheque only receiver account no is needed.



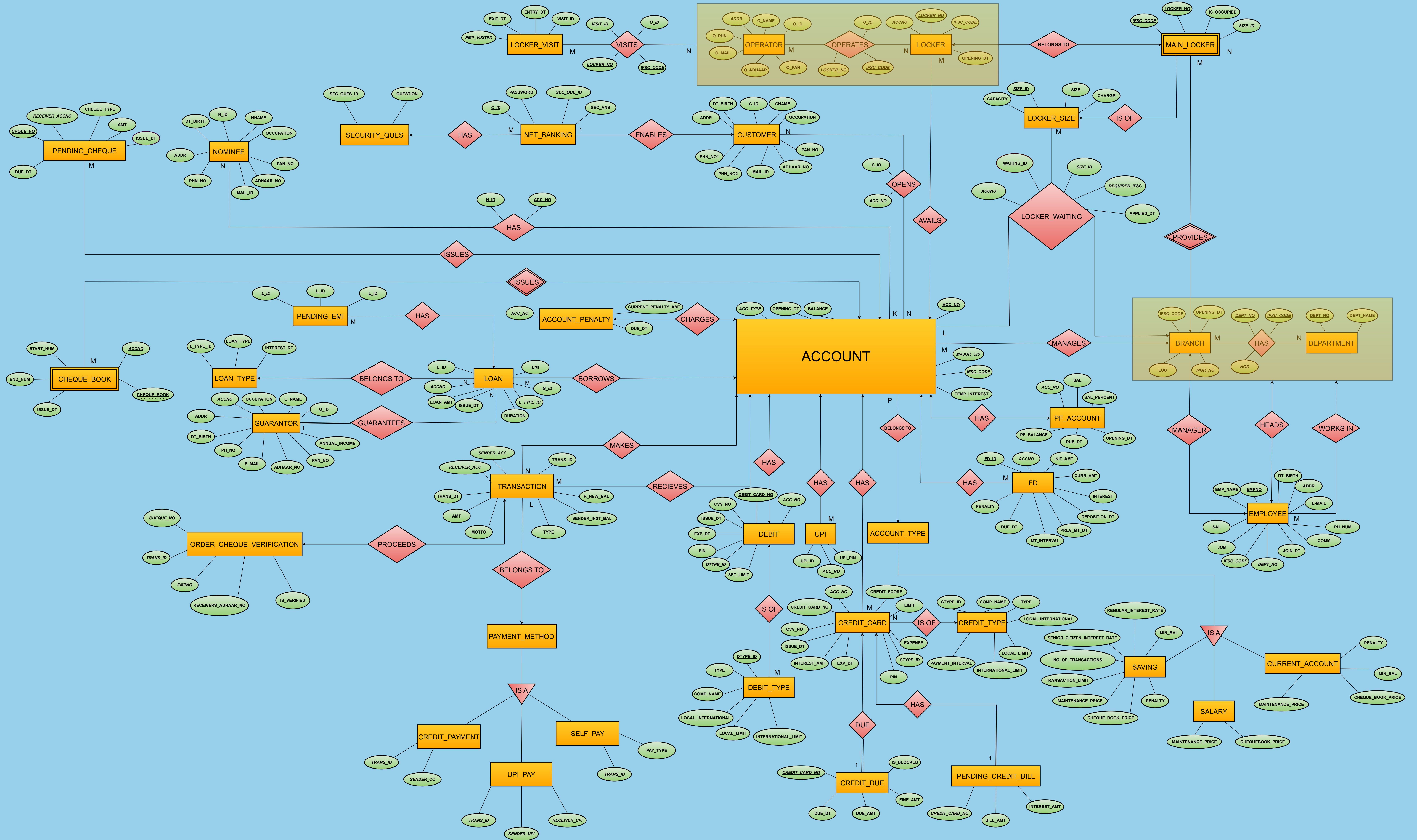
- Records of Pending Cheques will be deleted on successful transaction or also on unsuccessful transaction and be inserted in transaction if payment is successful.
- In case of order cheques receivers general information will be inserted in Order Cheque Verification table.







# E-R DIAGRAM



## ENTITY

## WEAK ENTITY

## RELATION

## WEAK RELATION

## ATTRIBUTE

**PRIMARY KEY** => **UNDERLINE**

**FOREIGN KEY**    => *ITALICS*

**DISCRIMINATOR => DOTTED LINE**





# Tables

- **CUSTOMER**(C\_ID, CNAME, DT\_BIRTH, ADDR, PHN\_NO1, PHN\_NO2, MAIL\_ID, ADHAAR\_NO, PAN\_NO, OCCUPATION)
- **BRANCH**(IFSC\_CODE, OPENING\_DT, LOC, MGR)
- **DEPARTMENT**(DEPTNO, DNAME)
- **EMPLOYEE**(EMPNO, EMP\_NAME, SAL, COMM, JOB, ADDRESS, BIRTH\_DT, JOINING\_DT, PHN\_NO, MAIL\_ID, IFSC\_CODE, DEPTNO)
- **ACCOUNT**(ACC\_NO, BALANCE, OPENING\_DT, ACC\_TYPE, TEMP\_INTEREST, MAJOR\_CID, IFSC\_CODE)
- **ACCOUNT\_PENALTY**(ACC\_NO, DUE\_DT, CURRENT\_PENALTY\_AMT)
- **NOMINEE**(N\_ID, NNAME, DT\_BIRTH, ADDR, PHN\_NO, MAIL\_ID, ADHAAR\_NO, PAN\_NO, OCCUPATION)
- **SAVING**(MIN\_BAL, REGULAR\_INTEREST, SENIOR\_SITIZEN\_INTEREST\_RATE, NO\_OF\_TRANSACTION, TRANSACTION\_LIMIT, PENALTY, CHEQUE\_BOOK\_PRICE, MAINTENANCE\_PRICE)
- **CURRENT\_ACCOUNT**(MIN\_BAL, PENALTY, CHEQUE\_BOOK\_PRICE, MAINTENANCE\_PRICE)
- **SALARY**(CHEQUEBOOK\_PRICE, MAINTENANCE\_PRICE)
- **PF\_ACCOUNT**(ACC\_NO, SAL, SAL\_PERCENT, OPENING\_DT, DUE\_DT, PF\_BALANCE)



- **FD**(**FD\_ID**, INIT\_AMT, CURR\_AMT, INTEREST, DEPOSITION\_DT, PREV\_MT\_DT, MT\_INTERVAL, DUE\_DT, PENALTY, **ACC\_NO**)
- **LOAN**(**L\_ID**, LOAN\_AMT, ISSUE\_DT, DURATION, EMI, L\_TYPE\_ID, G\_ID, **ACC\_NO**)
- **LOAN\_TYPE**(**L\_TYPE\_ID**, LOAN\_TYPE, INTEREST\_RT)
- **GUARANTOR**(**G\_ID**, G\_NAME, DT\_BIRTH, ADDR, PHN\_NO, E\_MAIL, OCCUPATION, ANNUAL\_INCOME, ADHAAR\_NO, PAN\_NO, **ACC\_NO**)
- **PENDING\_EMI**(**L\_ID**, PENDING\_DT, AMT)
- **LOCKER**(**IFSC\_CODE**, **LOCKER\_NO**, OPENING\_DT, ACCNO)
- **OPERATOR**(**O\_ID**, O\_NAME, ADDR, O\_PHN, O\_MAIL, O\_ADHAAR, O\_PAN)
- **LOCKER\_VISIT**(**VISIT\_ID**, ENTRY\_DT, EXIT\_DT, EMP\_VISITED)
- **MAIN\_LOCKER**(**IFSC\_CODE**, **LOCKER\_NO**, IS\_OCCUPIED, SIZE\_ID)
- **LOCKER\_SIZE**(**SIZE\_ID**, SIZE, CHARGE, CAPACITY)
- **DEBIT**(**DEBIT\_CARD\_NO**, CVV\_NO, PIN, SET\_LIMIT, ISSUE\_DT, EXP\_DT, **ACC\_NO**, DTYPE\_ID)
- **DEBIT\_TYPE**(**DTYPE\_ID**, COMP\_NAME, TYPE, LOCAL\_INTERNATIONAL, LOCAL\_LIMIT, INTERNATIONAL\_LIMIT)

- **CREDIT**(CREDIT\_CARD\_NO, CVV\_NO, PIN, LIMIT, CREDIT\_SCORE, EXPENSE, INTEREST\_AMT, ISSUE\_DT, EXP\_DT, **ACC\_NO**, CTYPE\_ID)
- **CREDIT\_TYPE**(CTYPE\_ID, COMP\_NAME, TYPE, LOCAL\_INTERNATIONAL, LOCAL\_LIMIT, INTERNATIONAL\_LIMIT, PAYMENT\_INTERVAL)
- **CREDIT\_DUE**(CREDIT\_CARD\_NO, DUE\_DT, DUE\_AMT, FINE\_AMT, IS\_BLOCKED)
- **PENDING\_CREDIT\_BILL**(CREDIT\_CARD\_NO, BILL\_AMT, INTEREST\_AMT)
- **UPI**(UPI\_ID, UPI\_PIN, **ACC\_NO**)
- **TRANSACTION**(TRANS\_ID, TRANS\_DT, AMT, TYPE, SENDER\_INSTANT\_BAL, RECEIVER\_INSTANT\_BAL, MOTTO, SENDER\_ACC, RECEIVER\_ACC)
- **CREDIT\_PAY**(TRANS\_ID, SENDER\_CC)
- **UPI\_PAY**(TRANS\_ID, SENDER\_UPI, RECEIVER\_UPI)
- **SELF\_PAY**(TRANS\_ID, PAY\_TYPE)
- **PENDING\_CHEQUE**(CHEQUE\_NO, CHEQUE\_TYPE, AMT, ISSUE\_DT, DUE\_DT, RECEIVER\_ACCNO)



- ORDER\_CHEQUE\_VERIFICATION(CHEQUE\_NO, RECEIVERS\_ADHAAR\_NO, IS\_VERIFIED, EMPNO, TRANS\_ID)
- CHEQUE\_BOOK(ACC\_NO, CHEQUEBOOK\_NO, START\_NUM, END\_NUM, ISSUE\_DT)
- NET\_BANKING(C\_ID, PASSWORD, SEC\_ANS, SEC\_QUE\_ID)
- SECURITY\_QUES(SEC\_QUES\_ID, QUESTION)
- BRANCH\_HAS\_DEPT(IFSC\_CODE, DEPT\_NO, HOD)
- ACCOUNT\_HAS\_NOMINEE(N\_ID, ACC\_NO)
- CUSTOMER\_OPENS\_ACCOUNT(C\_ID, ACC\_NO)
- LOC\_OPR(O\_ID, IFSC\_CODE, LOCKER\_NO)
- OPR\_VISITS\_LOC(O\_ID, IFSC\_CODE, LOCKER\_NO, V\_ID)
- LOCKER\_WAITING(WAITING\_ID, APPLIED\_DT, ACCNO, SIZE\_ID, REQUIRED\_IFSC)



# NORMALIZATION

- **CUSTOMER :**

- Every record in the customer table is atomic and uniquely identified by the primary key, therefore the table is in 1NF.
- Since there is a single primary key, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **BRANCH :**

- Every record is atomic, so it satisfies 1NF.
- Since there is single prime attribute, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, thus, transitivity is not present and the table is in 3NF.

- **DEPARTMENT :**

- Every record is atomic, so it satisfies 1NF.
- Since there is a single prime attribute, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **BRANCH\_HAS\_DEPARTMENT (RELATION) :**

- Every record is atomic and uniquely identified by prime attributes, thus it is in 1NF.
- Since, HOD is fully functionally dependent on both the prime attributes, it is in 2NF.
- As there is only one non-prime attribute, transitivity will not occur and it is 3NF.

- **EMPLOYEE :**

- Every record in EMPLOYEE table is atomic and uniquely identified by the primary key, therefore the table is in 1NF.
- Since, there is single primary key, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **ACCOUNT :**

- Since every record is atomic, it is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- As there is no transitivity in the columns, it is in 3NF.

- **ACCOUNT\_PENALTY :**

- Since every record is atomic, it is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- As there is no transitivity in the columns, it is in 3NF.

- **ACCOUNT\_HAS\_NOMINEE (RELATION):**

- Since every record is atomic, it is in 1NF.
- As there are only two attributes and both of them are prime attributes, each record is fully functionally dependent, thus it is in 2NF.
- Since there are no non-prime attributes, it also satisfies the condition for 3NF.



- **NOMINEE :**

- Every record in NOMINEE table is atomic and uniquely identified by the primary key, therefore the table is in 1NF.
- Since there is a single primary key, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **CUTOMER\_OPENS\_ACCOUNT (RELATION) :**

- Since every record is atomic, it is in 1NF.
- As there are only two attributes and both of them are prime attributes, each record is fully functionally dependent, thus it is in 2NF.
- Since there are no non-prime attributes, it also satisfies the condition for 3NF.

- **PF\_ACCOUNT :**

- Since every record is atomic and uniquely identified by the primary key, it is in 1NF.
- Since there is only one prime attribute, no need to check for 2NF.
- As no non-prime attribute depends on other non-prime attribute, transitivity is not present and hence it is in 3NF.

**• FD :**

- Since every record is atomic and uniquely identified by the primary key, it is in 1NF.
- Since there is only one prime attribute, no need to check for 2NF.
- As no non-prime attribute depends on other non-prime attribute, transitivity is not present and hence it is in 3NF.

**• LOAN :**

- Since every record is atomic and uniquely identified by the primary key, it is in 1NF.
- Since there is only one prime attribute, no need to check for 2NF.
- As no non-prime attribute depends on other non-prime attribute, transitivity is not present and hence it is in 3NF.

**• LOAN\_TYPE :**

- Since every record is atomic and uniquely identified by the primary key, it is in 1NF.
- Since there is only one prime attribute, no need to check for 2NF.
- As no non-prime attribute depends on other non-prime attribute, transitivity is not present and hence it is in 3NF.

**• GUARANTOR :**

- Every record in GUARANTOR table is atomic and uniquely identified by the primary key, therefore the table is in 1NF.
- Since there is a single primary key, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **LOCKER :**

- Since every record is atomic and uniquely identified by the primary key, it is in 1NF.
- As all the non-prime attributes are fully functionally dependent on both the prime attributes, it is in 2NF.
- As there is no transitivity present, the table is in 3NF.

- **OPERATOR :**

- Every record in OPERATOR table is atomic and uniquely identified by the primary key, therefore the table is in 1NF.
- Since there is a single primary key, we don't need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **OPERATOR\_OPERATES\_LOCKER (RELATION) :**

- As every record is atomic, the table is in 1NF.
- Since the record fully depends on all the three prime attributes, it satisfies 2NF.
- As there are no non-prime attributes, no need to check for 3NF.

- **OPERATOR\_VISITS\_LOCKER (RELATION) :**

- As every record is atomic, the table is in 1NF.
- Since the record fully depends on all the three prime attributes, it satisfies 2NF.
- As there are no non-prime attributes, no need to check for 3NF.



- **LOCKER\_VISIT :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, there is no transitivity, the table is in 3NF.

- **MAIN\_LOCKER :**

- As every record is atomic, the table is in 1NF.
- As all the non-prime attributes are fully functionally dependent on both the prime attributes, the table is in 2NF.
- As transitivity is not present, the table is in 3NF too.

- **LOCKER\_SIZE :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **LOCKER\_WAITING (RELATION) :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, there is no transitivity, the table is in 3NF.

- **DEBIT :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, there is no transitivity, the table is in 3NF.

- **DEBIT\_TYPE :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **CREDIT :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **CREDIT\_TYPE :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **CREDIT\_DUE :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **PENDING\_CREDIT\_BILL :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.



- **UPI :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **TRANSACTION :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **CREDIT\_PAYMENT :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- As there is only one non-prime attribute, no need to check for 3NF, it is already in 3NF.

- **UPI\_PAYMENT :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **SELF\_PAYMENT :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- As there is only one non-prime attribute, no need to check for 3NF, it is already in 3NF.

- **PENDING\_CHEQUE :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **ORDER\_CHEQUE\_VERIFICATION :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **CHEQUE\_BOOK :**

- As every record is atomic, the table is in 1NF.
- As all the non-prime attributes are fully functionally dependent on both the prime attributes, the table is in 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **NET\_BANKING :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- Since, no non-prime attribute depends on any other non-prime attribute, the table is in 3NF.

- **SECURITY\_QUES :**

- As every record is atomic, the table is in 1NF.
- As there is only one prime attribute, no need to check for 2NF.
- As there is only one non-prime attribute, no need to check for 3NF, it is already in 3NF.



## DDLs

- CREATE TABLE BRANCH(  
IFSC\_CODE VARCHAR2(8) CONSTRAINT PK\_IFSC PRIMARY KEY,  
OPENING\_DT DATE CONSTRAINT NN\_OPEN\_DT NOT NULL,  
LOC VARCHAR2(50) CONSTRAINT NN\_LOC NOT NULL,  
MGR\_NO NUMBER(4) CONSTRAINT FK\_MGR REFERENCES  
EMPLOYEE(EMPNO));
- CREATE TABLE DEPARTMENT (  
DEPT\_NO NUMBER(2) CONSTRAINT PK\_DEPTNO PRIMARY KEY,  
DEPT\_NAME CONSTRAINT NN\_DNAME NOT NULL);
- CREATE TABLE BRANCHHASDEPT(  
IFSC\_CODE VARCHAR2(8) CONSTRAINT FK\_IFSC REFERENCES  
BRANCH(IFSC\_CODE),  
DEPT\_NO NUMBER(2) CONSTRAINT FK\_DEPTNO REFERENCES  
DEPT(DEPTNO),  
HOD NUMBER(4) CONSTRAINT FK\_HOD REFERENCES  
EMPLOYEE(EMPNO),  
CONSTRAINT PK\_IFSC\_DEPTNO PRIMARY  
KEY(IFSC\_CODE,DEPTNO));
- CREATE TABLE EMPLOYEE(  
EMPNO NUMBER(4) CONSTRAINT PK\_EMPNO PRIMARY KEY,  
EMP\_NAME VARCHAR2(30) CONSTRAINT NN\_ENAME NOT  
NULL,  
SAL NUMBER(9,2),  
COMM NUMBER(5,2),  
JOB VARCHAR2(30),



```
ADDR VARCHAR2(100) CONSTRAINT NN_ADDRESS NOT
NULL,
DT_BIRTH DATE CONSTRAINT NN_BIRTH_DT NOT NULL,
JOIN_DT DATE CONSTRAINT NN_JOIN_DT NOT NULL,
PH_NUM NUMBER(10) CONSTRAINT NN_PHN NOT NULL,
E_MAIL VARCHAR2(30) CONSTRAINT NN_MAIL NOT NULL,
IFSC_CODE VARCHAR2(8) CONSTRAINT FK_IFSC_EMP NOT
REFERENCES
BRANCH(IFSC_CODE),
DEPT_NO NUMBER(2) CONSTRAINT FK_DEPTNO_EMP
REFERENCES DEPT(DEPTNO));
```

- CREATE TABLE CUSTOMER(  
C\_ID NUMBER(11) CONSTRAINT PK\_CUSID PRIMARY KEY,  
CNAME VARCHAR2(100) CONSTRAINT NN\_CNAME NOT  
NULL,  
DT\_BIRTH DATE CONSTRAINT NN\_BDAY NOT NULL,  
ADDR VARCHAR2(150) CONSTRAINT NN\_ADDR NOT NULL,  
PHN\_NO1 NUMBER CONSTRAINT NN\_PHN1 NOT NULL,  
PHN\_NO2 NUMBER CONSTRAINT NN\_PHN2 NOT NULL,  
MAIL\_ID VARCHAR2(100) CONSTRAINT NN\_MAIL NOT NULL,  
ADHAAR\_NO NUMBER(14) CONSTRAINT UK\_ADHAAR  
UNIQUE,  
PAN\_NO VARCHAR2(10) CONSTRAINT UK\_PAN UNIQUE,  
OCCUPATION VARCHAR2(25) CONSTRAINT  
NN\_OCCUPATION NOT NULL);



- CREATE TABLE ACCOUNT(  
ACC\_NO NUMBER(16) CONSTRAINT PK\_ACCNO PRIMARY  
KEY,  
IFSC\_CODE VARCHAR2(8) CONSTRAINT FK\_IFSCCODE  
REFERENCES BRANCH(IFSC\_CODE) NOT NULL,  
BALANCE NUMBER(20,2) CONSTRAINT NN\_BAL NOT NULL,  
OPENING\_DT DATE CONSTRAINT NN\_OPDT NOT NULL,  
ACC\_TYPE VARCHAR2(10) CONSTRAINT NN\_TYPE NOT  
NULL,  
TEMP\_INTEREST NUMBER(10) CONSTRAINT  
NN\_TEMP\_INTEREST NOT NULL,  
MAJOR\_CID NUMBER(11) CONSTRAINT FK\_MCID  
REFERENCES CUSTOMER(C\_ID) NOT NULL);
- CREATE TABLE NOMINEE(  
N\_ID NUMBER(10) CONSTRAINT PK\_NOM\_ID PRIMARY KEY,  
NNAME VARCHAR2(100) CONSTRAINT NN\_NOM\_NAME  
NOT NULL,  
DT\_BIRTH DATE CONSTRAINT NN\_NOM\_BDAY NOT NULL,  
ADDR VARCHAR2(150) CONSTRAINT NN\_NOM\_ADDR NOT  
NULL,  
PHN\_NO NUMBER CONSTRAINT NN\_NOM\_PHN NOT NULL,  
MAIL\_ID VARCHAR2(100) CONSTRAINT NN\_NOM\_MAIL  
NOT NULL,  
ADHAAR\_NO NUMBER(14) CONSTRAINT  
UK\_NOM\_ADHAAR UNIQUE,  
PAN\_NO VARCHAR2(10) CONSTRAINT UK\_NOM\_PAN  
UNIQUE,  
OCCUPATION VARCHAR2(25) CONSTRAINT  
NN\_NOM\_OCCUPATION NOT NULL);



- CREATE TABLE ACCOUNT\_HAS\_NOMINEE(  
N\_ID CONSTRAINT FK\_AHN\_NID REFERENCES  
NOMINEE(N\_ID),  
ACC\_NO CONSTRAINT FK\_AHN\_ACCNO REFERENCES  
ACCOUNT(ACC\_NO),  
CONSTRAINT PK\_NID#ACCNO PRIMARY  
KEY(N\_ID,ACC\_NO));
- CREATE TABLE CUSTOMER\_OPENS\_ACCOUNT(  
C\_ID CONSTRAINT FK\_COA\_CID REFERENCES  
CUSTOMER(C\_ID),  
ACC\_NO CONSTRAINT FK\_ACCNO REFERENCES  
ACCOUNT(ACC\_NO),  
CONSTRAINT PK\_COA\_CID#ACCNO PRIMARY  
KEY(C\_ID,ACC\_NO));
- CREATE TABLE ACCOUNT\_PENALTY(  
ACC\_NO CONSTRAINT FK\_AP\_ACCNO REFERENCES  
ACCOUNT(ACC\_NO),  
DUE\_DT DATE,  
CURRENT\_PENALTY\_AMT NUMBER(10),  
CONSTRAINT PK\_AP\_ACCNO PRIMARY KEY(ACC\_NO));
- CREATE TABLE CURRENT\_ACCOUNT(  
MIN\_BAL NUMBER(10),  
PENALTY NUMBER(4,2),  
CHEQUE\_BOOK\_PRICE NUMBER(6),  
MAINTENANCE\_PRICE NUMBER(6));
- CREATE TABLE SALARY(  
CHEQUE\_BOOK\_PRICE NUMBER(6),  
MAINTENANCE\_PRICE NUMBER(6));



- CREATE TABLE SAVING(  
MIN\_BAL NUMBER(10),  
REGULAR\_INTEREST\_RATE NUMBER(4,2),  
SENIOR\_CITIZEN\_INTEREST\_RATE NUMBER(4,2),  
NO\_OF\_TRANSACTION NUMBER,  
TRANSACTION\_LIMIT NUMBER(10),  
PENALTY NUMBER(4,2),  
CHEQUE\_BOOK\_PRICE NUMBER(6),  
MAINTENANCE\_PRICE NUMBER(6));
- CREATE TABLE PF\_ACCOUNT(  
ACC\_NO NUMBER(16) CONSTRAINT FK\_PFACCNO  
REFERENCES ACCOUNT(ACC\_NO),  
SAL NUMBER(15),  
SAL\_PERCENT NUMBER(4,2),  
OPENING\_DT DATE,  
DUE\_DT DATE,  
PF\_BALANCE NUMBER(15));
- CREATE TABLE UPI (  
UPI\_ID NUMBER(35) CONSTRAINT PK\_UPI PRIMARY KEY,  
ACC\_NO NUMBER(16) CONSTRAINT NN\_ACC\_UPI NOT  
NULL,  
UPI\_PIN NUMBER(4) CONSTRAINT NN\_UPI\_PIN NOT NULL,  
CONSTRAINT FK\_ACC\_UPI FOREIGN KEY(ACC\_NO)  
REFERENCES ACCOUNT(ACC\_NO) ON DELETE CASCADE);



- CREATE TABLE LOAN\_TYPE(  
L\_TYPE\_ID NUMBER(3) CONSTRAINT PK\_LOAN\_TYPE PRIMARY KEY,  
LOAN\_TYPE VARCHAR2(30) CONSTRAINT NN\_LOAN\_TYPE NOT NULL,  
INTEREST\_RT NUMBER(4,2) CONSTRAINT NN\_INTEREST\_LOAN NOT NULL);
- CREATE TABLE LOAN(  
L\_ID NUMBER(10) CONSTRAINT PK\_LID PRIMARY KEY,  
LOAN\_AMT NUMBER(9,2) CONSTRAINT NN\_LOAN\_AMT NOT NULL,  
ISSUE\_DT DATE CONSTRAINT NN\_ISSUE\_DT NOT NULL,  
DURATION NUMBER(4,2) CONSTRAINT NN\_DURATION NOT NULL,  
EMI NUMBER(5,2) CONSTRAINT NN\_EMI NOT NULL,  
L\_TYPE\_ID NUMBER(3) CONSTRAINT FK\_LOAN\_TYPE REFERENCES LOAN\_TYPE(LOAN\_TYPE\_ID),  
G\_ID NUMBER(10) CONSTRAINT FK\_GID REFERENCES GUARANTOR(G\_ID),  
ACCNO NUMBER(16) CONSTRAINT FK\_ACCNO\_LOAN REFERENCES ACCOUNT(ACCNO));
- CREATE TABLE GUARANTOR(  
G\_ID NUMBER(10) CONSTRAINT PK\_GID PRIMARY KEY,  
G\_NAME VARCHAR2(30) CONSTRAINT NN\_GNAME NOT NULL,  
DT\_BIRTH DATE CONSTRAINT NN\_BIRTHDT NOT NULL,  
ADDR VARCHAR2(100) CONSTRAINT NN\_ADDRESS NOT NULL,  
PHN\_NO NUMBER(10) CONSTRAINT NN\_PHN1 NOT NULL,  
E\_MAIL VARCHAR2(30) CONSTRAINT NN\_MAIL NOT NULL,  
AADHAR\_NO NUMBER(12) CONSTRAINT NN\_AADHAR\_NO NOT NULL,  
PAN\_NO NUMBER(10) CONSTRAINT NN\_PAN\_NO NOT NULL,  
OCCUPATION VARCHAR2(20) CONSTRAINT NN\_OCC NOT NULL,  
ANNUAL\_INCOME NUMBER(9,2) CONSTRAINT NN\_INCOME NOT NULL,  
ACCNO NUMBER(16) CONSTRAINT FK\_ACCNO\_GUARANTOR REFERENCES LOAN(ACCNO));



- CREATE TABLE PENDING\_EMI(  
    AMT NUMBER(5,2),  
    PENDING\_DT DATE;  
    L\_ID NUMBER(10) CONSTRAINT FK\_LID REFERENCES LOAN(L\_ID);  
    CONSTRAINT PK\_EMI PRIMARY KEY(L\_ID, PENDING\_DT));
- CREATE TABLE CHEQUEBOOK(  
    CHEQUEBOOK\_NO NUMBER(10) CONSTRAINT  
    PK\_CHEQUE\_BOOK\_NO PRIMARY KEY,  
    START\_NUM NUMBER(6) CONSTRAINT NN\_START NOT NULL,  
    END\_NUMBER NUMBER(6) CONSTRAINT NN\_END NOT NULL,  
    ISSUE\_DT DATE CONSTRAINT NN\_ISSUE\_DT\_CHEQUE NOT NULL,  
    ACCNO NUMBER(16) CONSTRAINT FK\_ACCNO\_CHEQUE  
    REFERENCES ACCOUNT(ACC\_NO));
- CREATE TABLE PENDINGCHEQUE(  
    CHEQUE\_NO NUMBER(6) CONSTRAINT PK\_CHEQUE\_NO PRIMARY  
    KEY,  
    CHEQUE\_TYPE VARCHAR2(20) CONSTRAINT NN\_CHEQUE\_TYPE  
    NOT NULL,  
    ISSUE\_DT DATE CONSTRAINT NN\_ISSUE\_DT NOT NULL,  
    AMT NUMBER(9,2) CONSTRAINT NN\_AMT NOT NULL,  
    DUE\_DT DATE CONSTRAINT NN\_DUE\_DT NOT NULL,  
    RECEIVER\_ACCNO NUMBER(16) CONSTRAINT FK\_REC\_ACCNO  
    REFERENCES ACCOUNT(ACC\_NO));



- CREATE TABLE FD(  
FD\_ID NUMBER(10) CONSTRAINT PK\_FD\_ID PRIMARY KEY,  
INIT\_AMT NUMBER(9,2) CONSTRAINT NN\_INITIAL\_AMT  
NOT NULL,  
CURR\_AMT NUMBER(9,2) CONSTRAINT NN\_CURRENT\_AMT  
NOT NULL,  
INTEREST NUMBER(4,2) CONSTRAINT NN\_INTEREST\_RT  
NOT NULL,  
DEPOSITION\_DT DATE CONSTRAINT NN\_DEP\_DT NOT  
NULL,  
PREV\_MT\_DT DATE,  
MT\_INTERVAL NUMBER(2) CONSTRAINT NN\_MAT\_INT NOT  
NULL,  
DUE\_DT DATE CONSTRAINT NN\_DUE\_DT NOT NULL,  
PENALTY NUMBER(4,2) CONSTRAINT NN\_PENALTY NOT  
NULL  
ACCNO NUMBER(16) CONSTRAINT FK\_ACCNO\_FD  
REFERENCES ACCOUNT(ACCNO));
- CREATE TABLE PF\_ACCOUNT(  
ACC\_NO NUMBER(16) CONSTRAINT FK\_PFACCNO  
REFERENCES ACCOUNT(ACC\_NO),  
SAL NUMBER(15),  
SAL\_PERCENT NUMBER(4,2),  
OPENING\_DT DATE,  
DUE\_DT DATE,  
PF\_BALANCE NUMBER(15));
- CREATE TABLE DEBIT (  
DEBIT\_CARD\_NO NUMBER(16) CONSTRAINT  
PK\_DEBIT\_CARD\_NO PRIMARY KEY,  
CVV\_NO NUMBER(4) CONSTRAINT NN\_CVV\_NO\_DEBIT  
NOT NULL,  
PIN NUMBER(4) CONSTRAINT NN\_PIN\_DEBIT NOT NULL,  
SET\_LIMIT NUMBER(20, 2) CONSTRAINT CK\_LIMIT\_DEBIT  
CHECK(SET\_LIMIT > 0),  
ISSUE\_DT DATE,



```
EXP_DT DATE,  
ACC_NO NUMBER(16) CONSTRAINT FK_ACC_NO_DEBIT  
REFERENCES ACCOUNT(ACC_NO),  
DTYPE_ID NUMBER(3) CONSTRAINT FK_DTYPE_ID  
REFERENCES DEBIT_TYPE(DTYPE_ID));
```

- CREATE TABLE DEBIT\_TYPE (  
DTYPE\_ID NUMBER(3) CONSTRAINT PK\_DTYPE\_ID  
PRIMARY KEY,  
COMP\_NAME VARCHAR2(20) CONSTRAINT  
NN\_COMP\_NAME\_DEBIT NOT NULL,  
TYPE VARCHAR2(20) CONSTRAINT NN\_TYPE\_DEBIT NOT  
NULL,  
LOCAL\_INTERNATIONAL VARCHAR2(1) CONSTRAINT  
NN\_LOC\_INTNL\_DEBIT NOT NULL,  
LOCAL\_LIMIT NUMBER(20, 2) CONSTRAINT  
CK\_LOC\_LIMIT\_DEBIT CHECK(LOCAL\_LIMIT > 0),  
INTERNATIONAL\_LIMIT NUMBER(20, 2) CONSTRAINT  
CK\_INTNL\_LIMIT\_DEBIT CHECK(INTERNATIONAL\_LIMIT >  
0));
- CREATE TABLE CREDIT (  
CREDIT\_CARD\_NO NUMBER(16) CONSTRAINT PK\_CREDIT\_CARD\_NO  
PRIMARY KEY,  
CVV\_NO NUMBER(4) CONSTRAINT NN\_CVV\_NO\_CREDIT NOT NULL,  
PIN NUMBER(4) CONSTRAINT NN\_PIN\_CREDIT NOT NULL,  
LIMIT NUMBER(20, 2) CONSTRAINT CK\_LIMIT\_CREDIT CHECK(LIMIT >  
0),  
CREDIT\_SCORE NUMBER(3),  
EXPENSE NUMBER(20, 2) CONSTRAINT CK\_EXPENSE\_CREDIT  
CHECK(EXPENSE >= 0),  
INTEREST\_AMT NUMBER(20, 2) CONSTRAINT  
CK\_INTEREST\_AMT\_CREDIT CHECK(INTEREST\_AMT >= 0),  
ISSUE\_DT DATE,  
EXP\_DT DATE,  
ACC\_NO NUMBER(16) CONSTRAINT FK\_ACC\_NO\_CREDIT





```
REFERENCES ACCOUNT(ACC_NO),  
CTYPE_ID NUMBER(3) CONSTRAINT FK_CTYPE_ID REFERENCES  
CREDIT_TYPE(CTYPE_ID));
```

- CREATE TABLE CREDIT\_TYPE (  
    CTYPE\_ID NUMBER(3) CONSTRAINT PK\_CTYPE\_ID PRIMARY KEY,  
    COMP\_NAME VARCHAR2(20) CONSTRAINT  
    NN\_COMP\_NAME\_CREDIT NOT NULL,  
    TYPE VARCHAR2(20) CONSTRAINT NN\_TYPE\_CREDIT NOT NULL,  
    LOCAL\_INTERNATIONAL VARCHAR2(1) CONSTRAINT  
    NN\_LOC\_INTNL\_CREDIT NOT NULL,  
    LOCAL\_LIMIT NUMBER(20, 2) CONSTRAINT CK\_LOC\_LIMIT\_CREDIT  
    CHECK(LOCAL\_LIMIT > 0),  
    INTERNATIONAL\_LIMIT NUMBER(20, 2) CONSTRAINT  
    CK\_INTNL\_LIMIT\_CREDIT CHECK(INTERNATIONAL\_LIMIT > 0),  
    PAYMENT\_INTERVAL NUMBER(3) CONSTRAINT  
    CK\_PAY\_INT\_CREDIT CHECK(PAYMENT\_INTERVAL > 0));
- CREATE TABLE CREDIT\_DUE (  
    CREDIT\_CARD\_NO NUMBER(16) CONSTRAINT  
    PK\_CREDIT\_CARD\_NO\_CREDITDUE PRIMARY KEY,  
    DUE\_DT DATE,  
    DUE\_AMT NUMBER(20, 2) CONSTRAINT CK\_DUE\_AMT  
    CHECK(DUE\_AMT > 0),  
    FINE\_AMT NUMBER(20, 2) CONSTRAINT CK\_FINE\_AMT  
    CHECK(FINE\_AMT >= 0),  
    ISBLOCKED VARCHAR2(1),  
    CONSTRAINT FK\_CREDIT\_CARD\_NO\_CREDITDUE FOREIGN  
    KEY(CREDIT\_CARD\_NO) REFERENCES CREDIT(CREDIT\_CARD\_NO)  
    );

- CREATE TABLE TRANSACTION (  
TRANS\_ID NUMBER(38) CONSTRAINT PK\_TR\_ID PRIMARY  
KEY,  
SENDER\_ACC NUMBER(16),RECEIVER\_ACC NUMBER(16),  
TRANS\_DT DATE CONSTRAINT NN\_TR\_DATE NOT NULL,  
AMT NUMBER(12,2) ,  
TYPE VARCHAR2(2) CONSTRAINT NN\_TR\_TYPE NOT NULL,  
SENDER\_INSTANT\_BAL  
NUMBER(18),RECEIVER\_INSTANT\_BAL NUMBER(18),  
MOTTO VARCHAR2(75),  
CONSTRAINT FK\_SACC FOREIGN KEY(SENDER\_ACC)  
REFERENCES ACCOUNT(ACC\_NO) ON DELETE CASCADE,  
CONSTRAINT FK\_RACC FOREIGN KEY(RECEIVER\_ACC)  
REFERENCES ACCOUNT(ACC\_NO) ON DELETE CASCADE);
- CREATE TABLE CREDIT\_PAY (  
TRANS\_ID NUMBER(38) CONSTRAINT PK\_CR\_TR\_ID  
PRIMARY KEY,  
SENDER\_CC NUMBER(20) CONSTRAINT NN\_CC\_PAY NOT  
NULL,  
CONSTRAINT FK\_CR\_PAY FOREIGN KEY(SENDER\_CC)  
REFERENCES CREDIT(CREDIT\_CARD\_NO) ON DELETE  
CASCADE);
- CREATE TABLE SELF\_PAY (  
TRANS\_ID NUMBER(38) CONSTRAINT PK\_SELF\_TR\_ID  
PRIMARY KEY,  
PAY\_TYPE VARCHAR2(1) CONSTRAINT NN\_PAY\_TYPE NOT  
NULL);



- CREATE TABLE UPI\_PAY (  
TRANS\_ID NUMBER(38) CONSTRAINT PK\_UPI\_TR\_ID  
PRIMARY KEY,  
SENDER\_UPI NUMBER(35) CONSTRAINT NN\_SEN\_UPI NOT  
NULL ,  
RECEIVER\_UPI NUMBER(35) CONSTRAINT NN\_REC\_UPI  
NOT NULL,  
CONSTRAINT FK\_UPI\_SEN\_PAY FOREIGN  
KEY(SENDER\_UPI) REFERENCES UPI(UPI\_ID) ON DELETE  
CASCADE,  
CONSTRAINT FK\_UPI\_REC\_PAY FOREIGN  
KEY(RECEIVER\_UPI) REFERENCES UPI(UPI\_ID) ON DELETE  
CASCADE );
- CREATE TABLE ORDERCHEQUEVERIFICATION(  
CHEQUE\_NO NUMBER(6) CONSTRAINT FK\_CHEQUE\_NO  
REFERENCES PENDINGCHEQUE(CHEQUE\_NO),  
TRANS\_ID NUMBER(30) CONSTRAINT FK\_TRANS\_CHEQUE  
REFERENCES TRANSACTION(TRANS\_ID),  
EMPNO NUMBER(4) CONSTRAINT FK\_EMPNO\_CHEQUE REFERENCES  
EMPLOYEE(EMPNO),  
RECEIVERS\_ADHAAR\_NO NUMBER(12) CONSTRAINT  
NN\_REC\_AADHAR NOT NULL,  
IS\_VERIFIED BOOLEAN CONSTRAINT NN\_VERIFIED NOT NULL,  
CONSTRAINT PK\_CHEQUENO\_VER PRIMARY KEY(CHEQUE\_NO));
- CREATE TABLE PENDING\_CREDIT\_BILL (  
CREDIT\_CARD\_NO NUMBER(16) CONSTRAINT  
PK\_CREDIT\_CARD\_NO\_BILL PRIMARY KEY,  
BILL\_AMT NUMBER(20, 2) CONSTRAINT CK\_BILL\_AMT  
CHECK(BILL\_AMT >= 0),  
INTEREST\_AMT NUMBER(20, 2) CONSTRAINT CK\_INT\_AMT  
CHECK(INTEREST\_AMT >= 0),  
CONSTRAINT FK\_CREDIT\_CARD\_NO\_BILL FOREIGN  
KEY(CREDIT\_CARD\_NO) REFERENCES CREDIT(CREDIT\_CARD\_NO));



- CREATE TABLE LOCKER\_SIZE (  
    SIZE\_ID NUMBER(2) CONSTRAINT PK\_SIZEID PRIMARY KEY,  
    SIZE VARCHAR2(3) CONSTRAINT NN\_SIZE\_LSIZE NOT NULL,  
    CHARGE NUMBER(4, 2) CONSTRAINT CK\_CHG\_LSIZE  
    CHECK(CHARGE >= 0),  
    CAPACITY NUMBER(7, 2) CONSTRAINT CK\_CAP\_LSIZE  
    CHECK(CAPACITY >= 0));
- CREATE TABLE MAIN\_LOCKER (  
    IFSC\_CODE VARCHAR2(8) CONSTRAINT FK\_IFSC\_MLOC  
    REFERENCES BRANCH(IFSC\_CODE),  
    LOCKER\_NO NUMBER(3),  
    IS\_OCCUPIED VARCHAR2(1) CONSTRAINT NN\_ISOCC\_MLOC NOT  
    NULL,  
    SIZE\_ID NUMBER(2) CONSTRAINT FK\_SIZEID\_MLOC REFERENCES  
    LOCKER\_SIZE(SIZE\_ID),  
    CONSTRAINT PK\_IFSC\_LNO\_MLOC PRIMARY KEY(IFSC\_CODE,  
    LOCKER\_NO));
- CREATE TABLE LOCKER (  
    IFSC\_CODE VARCHAR2(8),  
    LOCKER\_NO NUMBER(3),  
    OPENING\_DT DATE,  
    ACCNO NUMBER(16) CONSTRAINT FK\_ACCNO\_LOC REFERENCES  
    ACCOUNT(ACNO),  
    CONSTRAINT FK\_IFSC\_LNO\_LOC FOREIGN KEY(IFSC\_CODE,  
    LOCKER\_NO) REFERENCES MAIN\_LOCKER(IFSC\_CODE,  
    LOCKER\_NO),  
    CONSTRAINT PK\_LOCKER PRIMARY KEY(IFSC\_CODE, LOCKER\_NO));
- CREATE TABLE OPERATOR (  
    O\_ID NUMBER(11) CONSTRAINT PK\_OID\_OPR PRIMARY KEY,  
    O\_NAME VARCHAR2(100) CONSTRAINT NN\_ONAME NOT NULL,  
    ADDR VARCHAR2(150) CONSTRAINT NN\_OADDR NOT NULL,  
    O\_PHN NUMBER(10) CONSTRAINT NN\_OPHN NOT NULL,

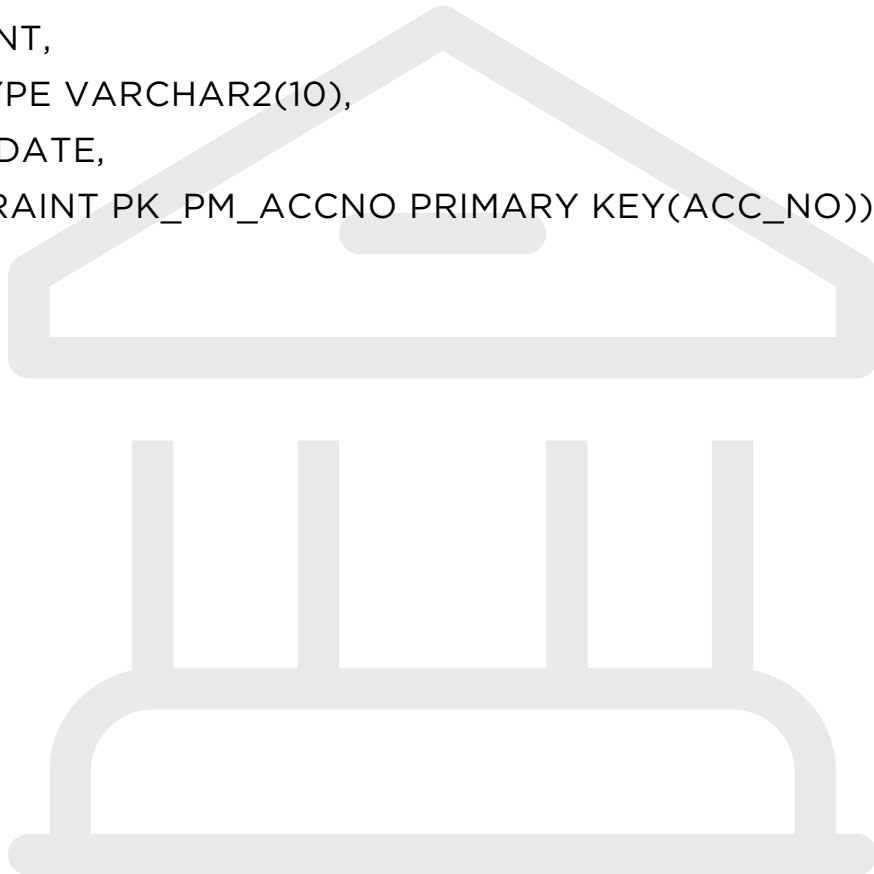


```
O_MAIL VARCHAR2(100) CONSTRAINT NN_OMAIL NOT NULL,  
O_ADHAAR NUMBER(14) CONSTRAINT UK_OADHAAR UNIQUE,  
O_PAN VARCHAR2(10) CONSTRAINT UK_OPAN UNIQUE);
```

- CREATE TABLE LOC\_OPR (  
O\_ID NUMBER(11) CONSTRAINT FK\_OID\_LOCOPR REFERENCES  
OPERATOR(O\_ID),  
IFSC\_CODE VARCHAR2(8),  
LOCKER\_NO NUMBER(3),  
CONSTRAINT FK\_IFSC\_LNO\_LOCOPR FOREIGN KEY(IFSC\_CODE,  
LOCKER\_NO) REFERENCES LOCKER(IFSC\_CODE, LOCKER\_NO),  
CONSTRAINT PK\_OID\_IFSC\_LNO\_LOCOPR PRIMARY KEY(O\_ID,  
IFSC\_CODE, LOCKER\_NO));
- CREATE TABLE LOCKER\_VISIT (  
VISIT\_ID NUMBER(10) CONSTRAINT PK\_VID PRIMARY KEY,  
ENTRY\_DT DATE,  
EXIT\_DT DATE,  
EMP\_VISITED NUMBER(11) CONSTRAINT FK\_EMPNO\_LOCV  
REFERENCES EMPLOYEE(EMPNO));
- CREATE TABLE OPR\_VISITS\_LOC (  
O\_ID NUMBER(11),  
IFSC\_CODE VARCHAR2(8),  
LOCKER\_NO NUMBER(3),  
V\_ID NUMBER(10) CONSTRAINT FK\_VID\_OPRVLOC REFERENCES  
LOCKER\_VISIT(V\_ID),  
CONSTRAINT FK\_OID\_IFSC\_LNO\_OPRVLOC FOREIGN KEY(O\_ID,  
IFSC\_CODE, LOCKER\_NO) REFERENCES LOC\_OPR(O\_ID, IFSC\_CODE,  
LOCKER\_NO),  
CONSTRAINT PK\_OPRVLOC PRIMARY KEY(O\_ID, IFSC\_CODE,  
LOCKER\_NO, V\_ID));



- CREATE TABLE LOCKER\_WAITING (  
    WAITING\_ID NUMBER(10) CONSTRAINT PK\_LOCWAIT PRIMARY KEY,  
    APPLIED\_DT DATE,  
    ACCNO NUMBER(16) CONSTRAINT FK\_ACCNO\_LOCWAIT  
    REFERENCES ACCOUNT(ACCNO),  
    SIZE\_ID NUMBER(2) CONSTRAINT FK\_SIZEID\_LOCWAIT REFERENCES  
    LOCKER\_SIZE(SIZE\_ID),  
    REQUIRED\_IFSC VARCHAR2(8) CONSTRAINT FK\_IFSC\_LOCWAIT  
    REFERENCES BRANCH(IFSC\_CODE));
- CREATE TABLE PENDINGMAINTENANCE(  
    ACC\_NO NUMBER(16) CONSTRAINT FK\_PM\_ACCNO REFERENCES  
    ACCOUNT,  
    ACC\_TYPE VARCHAR2(10),  
    ISS\_DT DATE,  
    CONSTRAINT PK\_PM\_ACCNO PRIMARY KEY(ACC\_NO));





# PROCEDURES

## 1) A PROCEDURE TO GENERATE IFSC CODE FOR BRANCH (4 DIGIT – BANK NAME||4 DIGIT – BRANCH NO)

```
CREATE OR REPLACE PROCEDURE GENERATE_IFSC(IFSC_CODE OUT VARCHAR2)
IS
  BANK_NAME VARCHAR2(4):='BANK';
  BRANCHNO VARCHAR2(4);
  TEMP NUMBER(4);
BEGIN
  SELECT COUNT(*) INTO TEMP FROM BRANCH;
  TEMP:=TEMP+1;
  BRANCHNO:=TO_CHAR(TEMP,'FM0000');
  IFSC_CODE:=BANK_NAME || BRANCHNO;
END;
/
```

## 2) PROCEDURE TO GENERATE ACC NO. (4 DIGIT – BANK ID || 4 DIGIT BRANCH NO || 4 DIGIT (MMYY) || 4 DIGIT - NUMBER OF ACCOUNT IN CURRENT MONTH)

```
CREATE OR REPLACE PROCEDURE GENERATE_ACCNO
(BRANCHNO IN VARCHAR2,ACCNO OUT NUMBER)
IS
  BANK_ID NUMBER(4) := 1111;
  ACC_IN_MONTH NUMBER(4);
  TEMP_ACCNO VARCHAR2(16);
  TEMP VARCHAR2(4);
BEGIN
  TEMP := TO_CHAR(BANK_ID);
  TEMP_ACCNO := TEMP || BRANCHNO;
  TEMP := TO_CHAR(SYSDATE,'MMYY');
  TEMP_ACCNO := TEMP_ACCNO || TEMP;
  SELECT COUNT(*) INTO ACC_IN_MONTH FROM ACCOUNT WHERE
  TO_CHAR(OPENING_DT,'MMYY') = TO_CHAR(SYSDATE,'MMYY');
  ACC_IN_MONTH := C+1;
  TEMP := TO_CHAR(ACC_IN_MONTH,'FM0000');
  TEMP_ACCNO := TEMP_ACCNO || TEMP;
  ACCNO := TO_NUMBER(TEMP_ACCNO);
END;
```

**3) EVERYTIME THIS PROCEDURE IS CALLED IT WILL CALCULATE INTEREST OF SAVING ACCOUNT**

```
CREATE OR REPLACE PROCEDURE UPDATEINTEREST
IS
CURSOR C IS SELECT * FROM ACCOUNT WHERE ACC_TYPE LIKE '%SAVING%';
S_IR SAVING.REGULAR_INTEREST_RATE%TYPE;
S_SIR SAVING.SENIOR_CITIZEN_INTEREST_RATE%TYPE;
BEGIN
SELECT REGULAR_INTEREST_RATE,SENIOR_CITIZEN_INTEREST_RATE INTO S_IR,S_SIR
FROM SAVING;
FOR R IN C
LOOP
IF(R.ACC_TYPE LIKE '%SAVING%') THEN
IF(ISSENIOR(R.MAJOR_CID)) THEN
R.TEMP_INTEREST := (R.BALANCE + R.TEMP_INTEREST)*(S_SIR/100);
ELSE
R.TEMP_INTEREST := (R.BALANCE + R.TEMP_INTEREST)*(S_IR/(100));
END IF;
END IF;
UPDATE ACCOUNT SET TEMP_INTEREST = R.TEMP_INTEREST WHERE ACC_NO =
R.ACC_NO;
COMMIT;
END LOOP;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO DATA FOUND');
END UPDATEINTEREST;
/
```

**SCHEDULE:-**

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
JOB_NAME => 'DAILY_SCHEDULE_JOB_1',
JOB_TYPE => 'PLSQL_BLOCK',
JOB_ACTION => 'BEGIN CALCPENALTY(); END;',
START_DATE => SYSTIMESTAMP,
REPEAT_INTERVAL => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=10; BYSECOND=0'
END_DATE => NULL,
ENABLED => TRUE,
COMMENTS => 'CALCULATES PENALTY AND UPDATES PENALTY OF ACCOUNT HOLDER');
END;
/
```



**4) SCHEDULED PROCEDURE: EVERYTIME THIS PROCEDURE IS CALLED IT WILL CALCULATE PENALTY FOR ACCOUNT**

CREATE OR REPLACE PROCEDURE CALCPENALTY  
IS

CURSOR P IS SELECT \* FROM ACCOUNTPENALTY;  
PEN SAVING.PENALTY%TYPE;  
M SAVING.MIN\_BAL%TYPE;  
T ACCOUNT.ACC\_TYPE%TYPE;  
B ACCOUNT.BALANCE%TYPE;  
BEGIN  
FOR R IN P  
LOOP

SELECT BALANCE,ACC\_TYPE INTO B,T FROM ACCOUNT WHERE ACC\_NO = R.ACC\_NO;  
IF(T LIKE '%SAVING%') THEN  
SELECT PENALTY,MIN\_BAL INTO PEN,M FROM SAVING;  
IF(B<M) THEN  
R.CURRENT\_PENALTY\_AMT := NVL(R.CURRENT\_PENALTY\_AMT,0) + ((M-B)\*(PEN)/100);  
ELSIF(B>M+1000) THEN  
INSERT INTO TRANSACTION  
VALUES(GET\_TRANS\_ID(),R.ACC\_NO,NULL,SYSDATE,'AP',R.CURRENT\_PENALTY\_AMT,B-  
R.CURRENT\_PENALTY\_AMT,NULL,'ACCOUNT PENALTY');  
END IF;  
ELSIF(T LIKE '%CURRENT%') THEN  
SELECT PENALTY,MIN\_BAL INTO PEN,M FROM CURRENTACCOUNT;  
IF(B<M) THEN  
R.CURRENT\_PENALTY\_AMT := NVL(R.CURRENT\_PENALTY\_AMT,0) + ((M-B)\*(PEN)/100);  
ELSIF(B>M+1000) THEN  
INSERT INTO TRANSACTION  
VALUES(GET\_TRANS\_ID(),R.ACC\_NO,NULL,SYSDATE,'AP',R.CURRENT\_PENALTY\_AMT,B-  
R.CURRENT\_PENALTY\_AMT,NULL,'ACCOUNT PENALTY');  
END IF;  
END IF;  
UPDATE ACCOUNTPENALTY SET CURRENT\_PENALTY\_AMT =  
R.CURRENT\_PENALTY\_AMT WHERE ACC\_NO = R.ACC\_NO;  
COMMIT;  
END LOOP;  
END CALCPENALTY;

**SCHEDULE:- TO CALCULATE AND UPDATE PENALTY DAILY**

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
  JOB_NAME => 'DAILY_SCHEDULE_JOB_1',
  JOB_TYPE => 'PLSQL_BLOCK',
  JOB_ACTION => 'BEGIN CALCPENALTY(); END;',
  START_DATE => SYSTIMESTAMP,
  REPEAT_INTERVAL => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=10; BYSECOND=0'
  END_DATE => NULL,
  ENABLED => TRUE,
  COMMENTS => 'CALCULATES PENALTY AND UPDATES PENALTY OF ACCOUNT
HOLDER');
END;
/
```

**5) SCHEDULED PROCEDURE: TO GENERATE THE BILL OF EXPENSES OF A CREDIT CARD EVERY MONTH AND SET THE EXPENSE AND INTEREST AMOUNT IN THE CREDIT CARD EQUAL TO 0(CALLED ON 1ST DATE OF EVERY MONTH) :**

```
CREATE OR REPLACE PROCEDURE GENERATECREDITBILL
AS
  LAST_TRANS_DT DATE;
  R2 CREDIT_DUE%ROWTYPE;
  CURSOR C1 IS SELECT * FROM CREDIT;
BEGIN
  FOR R1 IN C1
  LOOP
    IF(R1.EXPENSE = 0) THEN CONTINUE;
    END IF;

    SELECT MAX(TRANS_DT) INTO LAST_TRANS_DT FROM TRANSACTION
    WHERE TRANS_ID IN (
      SELECT TRANS_ID FROM CREDIT_PAY
      WHERE SUBSTR(TRANS_ID, 24, 6) = TO_CHAR(SYSDATE, 'MMYYYY')
      AND SENDER_CC = R1.CREDIT_CARD_NO
    );
    R1.INTEREST_AMT := R1.INTEREST_AMT + CALCULATECREDITINTEREST(R1.EXPENSE,
    ROUND(SYSDATE - LAST_TRANS_DT));
    IF(CHECKEXISTANCEINCREDITDUE(R1.CREDIT_CARD_NO)) THEN
      SELECT * INTO R2 FROM CREDIT_DUE WHERE CREDIT_CARD_NO =
      R1.CREDIT_CARD_NO;
```



```
R2.FINE_AMT := R2.FINE_AMT + CALCULATECREDITINTEREST(R2.DUE_AMT,  
ROUND(SYSDATE - R2.DUE_DT));
```

```
UPDATE CREDIT_DUE SET DUE_AMT = R2.DUE_AMT + R1.EXPENSE,  
FINE_AMT = R2.FINE_AMT + R1.INTEREST_AMT,  
DUE_DT = SYSDATE,  
ISBLOCKED = 'Y'  
WHERE CREDIT_CARD_NO = R1.CREDIT_CARD_NO;  
ELSE  
INSERT INTO PENDING_CREDIT_BILL  
VALUES(R1.CREDIT_CARD_NO, R1.EXPENSE, R1.INTEREST_AMT);  
END IF;
```

```
UPDATE CREDIT SET INTEREST_AMT = 0,  
EXPENSE = 0  
WHERE CREDIT_CARD_NO = R1.CREDIT_CARD_NO;  
END LOOP;  
COMMIT;  
END GENERATECREDITBILL;  
/
```

**SCHEDULE:- SCHEDULE WHICH RUNS A JOB ON 1ST DATE OF EVERY MONTH AND CALLS THE PROCEDURE GENERATECREDITBILL(), WHICH GENERATES THE BILLS OF EACH CREDIT CARD :**  
**BEGIN**

```
DBMS_SCHEDULER.CREATE_JOB (  
JOB_NAME => 'MONTHLY_SCHEDULE_JOB_1',  
JOB_TYPE => 'PLSQL_BLOCK',  
JOB_ACTION => 'BEGIN GENERATECREDITBILL(); END;',  
START_DATE => SYSTIMESTAMP,  
REPEAT_INTERVAL => 'FREQ=MONTHLY; BYMONTHDAY=1'; BYHOUR=0; BYMINUTE=0;  
BYSECOND=0'  
END_DATE => NULL,  
ENABLED => TRUE,  
COMMENTS => 'JOB TO GENERATE CREDIT CARD BILLS ON 1ST OF EVERY MONTH';  
END;  
/
```



**6)SCHEDULED PROCEDURE : TO MOVE THE CREDIT CARD RECORDS FROM  
PENDING\_CREDIT\_BILL TABLE TO CREDIT\_DUE TABLE AFTER THE DUE  
DATE(PROCEDURE CALLED ON 21ST OF EVERY MONTH) :**

```
CREATE OR REPLACE PROCEDURE CREDITDUEDATE
AS
CURSOR C1 IS SELECT * FROM PENDING_CREDIT_BILL;
R2 CREDIT_DUE%ROWTYPE;
BEGIN
FOR R1 IN C1
LOOP
R1.INTEREST_AMT := R1.INTEREST_AMT + GENERATEDUEDATEPENALTY(R1.BILL_AMT) +
CALCULATECREDITINTEREST(R1.BILL_AMT, 20);
IF(CHECKEXISTANCEINCREDITDUE(R1.CREDIT_CARD_NO)) THEN
SELECT * INTO R2 FROM CREDIT_DUE
WHERE CREDIT_CARD_NO = R1.CREDIT_CARD_NO;

R2.FINE_AMT := R2.FINE_AMT + CALCULATECREDITINTEREST(R2.DUE_AMT,
ROUND(SYSDATE - R2.DUE_DT));

UPDATE CREDIT_DUE
SET DUE_AMT = DUE_AMT + R1.BILL_AMT,
FINE_AMT = R2.FINE_AMT + R1.INTEREST_AMT,
DUE_DT = SYSDATE,
ISBLOCKED = 'Y'
WHERE CREDIT_CARD_NO = R1.CREDIT_CARD_NO;
ELSE
INSERT INTO CREDIT_DUE
VALUES(R1.CREDIT_CARD_NO, SYSDATE, R1.BILL_AMT, R1.INTEREST_AMT, 'N');
END IF;

DELETE FROM PENDING_CREDIT_BILL
WHERE CREDIT_CARD_NO = R1.CREDIT_CARD_NO;
END LOOP;
END CREDITDUEDATE;
/
```



**SCHEDULE:- SCHEDULE WHICH RUNS A JOB ON 21ST OF EVERY MONTH AND CALLS THE PROCEDURE CREDITDUE DATE(), WHICH FINDS THE CREDIT CARDS WHOSE BILLS ARE PENDING EVEN AFTER THE DUE DATE :**

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
  JOB_NAME      => 'MONTHLY_SCHEDULE_JOB_21',
  JOB_TYPE      => 'PLSQL_BLOCK',
  JOB_ACTION    => 'BEGIN CREDITDUE DATE(); END;',
  START_DATE    => SYSTIMESTAMP,
  REPEAT_INTERVAL => 'FREQ=MONTHLY; BYMONTHDAY=21'; BYHOUR=0; BYMINUTE=0;
  BYSECOND=0,
  END_DATE      => NULL,
  ENABLED       => TRUE,
  COMMENTS      => 'JOB TO INSERT PENDING BILLS TO CREDIT_DUE ON 21ST OF EVERY
MONTH');
END;
/
```

**7)PROCEDURE TO UPDATE THE EXPENSE OF A CREDIT CARD AFTER EACH TRANSACTION PERFORMED THROUGH THAT CREDIT CARD, AND TO DEAL WITH THE OVERLIMIT EXPENSE CONDITIONS AS WELL :**

```
CREATE OR REPLACE PROCEDURE UPDATECREDITONTRANSACTION(CNO NUMBER, AMT
NUMBER)
IS
  LAST_TRANS_DT DATE;
  R1 CREDIT%ROWTYPE;
BEGIN
  SELECT * INTO R1 FROM CREDIT
  WHERE CREDIT_CARD_NO = CNO;

  SELECT MAX(TRANS_DT) INTO LAST_TRANS_DT FROM TRANSACTION
  WHERE TRANS_ID IN (
  SELECT TRANS_ID FROM CREDIT_PAY
  WHERE SUBSTR(TRANS_ID, 24, 6) = TO_CHAR(SYSDATE, 'MMYYYY')
  AND SENDER_CC = R1.CREDIT_CARD_NO
  );

  R1.INTEREST_AMT := R1.INTEREST_AMT + CALCULATECREDITINTEREST(R1.EXPENSE,
  ROUND(SYSDATE - LAST_TRANS_DT));
```



UPDATE CREDIT

```
SET EXPENSE = EXPENSE + AMT,  
INTEREST_AMT = R1.INTEREST_AMT  
WHERE CREDIT_CARD_NO = CNO;  
COMMIT;
```

```
IF((R1.EXPENSE + AMT) > R1.LIMIT) THEN  
CREDITDUEAMOUNT(CNO, R1.EXPENSE + AMT, R1.INTEREST_AMT, R1.LIMIT);  
END IF;  
END UPDATECREDITONTRANSACTION;
```

## 8) PROCEDURE TO PERFORM THE TASK OF CREDIT CARD BILL PAYMENT

CREATE OR REPLACE PROCEDURE CREDITBILLPAYMENT  
AS

```
CNO CREDIT.CREDIT_CARD_NO%TYPE;  
R1 CREDIT_DUE%ROWTYPE;  
R2 PENDING_CREDIT_BILL%ROWTYPE;  
PAY_AMT CREDIT_DUE.DUE_AMT%TYPE;  
TRANS_ID TRANSACTION.TRANS_ID%TYPE;  
ACCNO ACCOUNT.ACC_NO%TYPE;  
BAL ACCOUNT.BALANCE%TYPE;
```

BEGIN

```
DBMS_OUTPUT.PUT('ENTER YOUR CREDIT CARD NUMBER : ');  
CNO := 1234;  
TRANS_ID = GET_TR_ID();
```

```
IF(CHECKCREDITNUMBERVALIDITY(CNO)) THEN  
SELECT ACC_NO INTO ACCNO FROM CREDIT  
WHERE CREDIT_CARD_NO = CNO;
```

```
SELECT BALANCE INTO BAL FROM ACCOUNT  
WHERE ACC_NO = ACCNO;
```

```
IF(CHECKEXISTANCEINCREDITDUE(CNO)) THEN  
SELECT * INTO R1 FROM CREDIT_DUE  
WHERE CREDIT_CARD_NO = CNO;
```

```
R1.FINE_AMT := R1.FINE_AMT +CALCULATECREDITINTEREST(R1.DUE_AMT,  
ROUND(SYSDATE - R1.DUE_DT));  
PAY_AMT := R1.DUE_AMT + R1.FINE_AMT;
```

```
IF(BAL > PAY_AMT) THEN  
INSERT INTO TRANSACTION  
VALUES(TRANS_ID, ACCNO, NULL, SYSDATE, PAY_AMT, 'CB', BAL - PAY_AMT, NULL,  
NULL);
```



```
DELETE FROM CREDIT_DUE
WHERE CREDIT_CARD_NO = CNO;
DBMS_OUTPUT.PUT_LINE('BILL PAID !!!');
ELSE DBMS_OUTPUT.PUT_LINE('BILL PAYMENT UNSUCCESSFUL ...');
END IF;
ELSE
SELECT * INTO R2 FROM PENDING_CREDIT_BILL
WHERE CREDIT_CARD_NO = CNO;
PAY_AMT := R2.BILL_AMT;

IF(BAL > PAY_AMT) THEN
INSERT INTO TRANSACTION
VALUES(TRANS_ID, ACCNO, NULL, SYSDATE, PAY_AMT, 'CB', BAL - PAY_AMT, NULL,
NULL);
DELETE FROM PENDING_CREDIT_BILL
WHERE CREDIT_CARD_NO = CNO;
DBMS_OUTPUT.PUT_LINE('BILL PAID !!!');
ELSE DBMS_OUTPUT.PUT_LINE('BILL PAYMENT UNSUCCESSFUL ...');
END IF;
END IF;
ELSE
DBMS_OUTPUT.PUT_LINE('CREDIT CARD NUMBER : ' || CNO || ' DOES NOT EXIST ');
END IF;
COMMIT;

EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('CREDIT CARD NUMBER : ' || CNO || ' DOESNOT HAVE ANY
PENDING BILLS ... ');
END;
/
```

## 9) DEDUCTS MAINTENANCE AMMOUNT YEARLY OPENING ACCOUNT

CREATE OR REPLACE PROCEDURE DEDUCTMAINTENANCE  
IS

```
A SAVING.MIN_BAL%TYPE;
M SAVING.MAINTENANCE_PRICE%TYPE;
CURSOR C IS SELECT ACC_NO,ACC_TYPE,BALANCE FROM ACCOUNT WHERE
TO_CHAR(SYSDATE,'DD-MM-YYYY') = TO_CHAR(OPENING_DT+365,'DD-MM-YYYY');
```

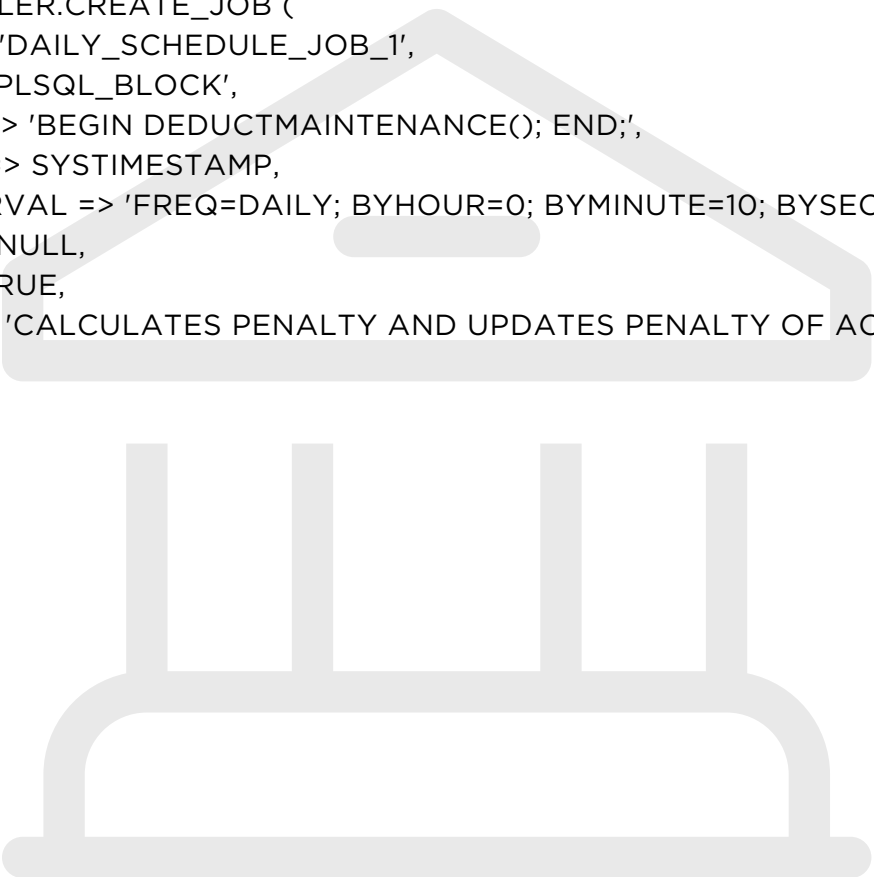
```
BEGIN
FOR R IN C LOOP
IF(R.ACC_TYPE LIKE '%SAVING%') THEN
SELECT MAINTENANCE_PRICE,MIN_BAL INTO M,A FROM SAVING;
ELSIF(R.ACC_TYPE LIKE '%CURRENT%') THEN
SELECT MAINTENANCE_PRICE,MIN_BAL INTO M,A FROM CURRENTACCOUNT;
ELSIF(R.ACC_TYPE LIKE '%SALARY%') THEN
SELECT MAINTENANCE_PRICE INTO M FROM SALARY;
A := 0;
```



```
END IF;
IF(R.BALANCE-M > A) THEN
INSERT INTO TRANSACTION
VALUES(GET_TR_ID(),R.ACC_NO,NULL,SYSDATE,M,'ZZ',R.BALANCE -
M,NULL,'MAINTENANCE');
ELSE
INSERT INTO PENDINGMAINTENANCE VALUES(R.ACC_NO,R.ACC_TYPE,SYSDATE);
COMMIT;
END IF;
END LOOP;
END;
/
```

**SCHEDULE:- TO DEDUCT MAINTENANCE CHARGES OF ACCOUNT BY YEARLY AFTER  
OPENING ACCOUNT (SCHEDULE RUNS DAILY)**

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
JOB_NAME => 'DAILY_SCHEDULE_JOB_1',
JOB_TYPE => 'PLSQL_BLOCK',
JOB_ACTION => 'BEGIN DEDUCTMAINTENANCE(); END;',
START_DATE => SYSTIMESTAMP,
REPEAT_INTERVAL => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=10; BYSECOND=0'
END_DATE => NULL,
ENABLED => TRUE,
COMMENTS => 'CALCULATES PENALTY AND UPDATES PENALTY OF ACCOUNT
HOLDER');
END;
/
```





**10)AS PER THE GIVEN ACCOUNT NO IT GENERATES ACCOUNT TRANSACTION HISTORY FROM FIRST TRANSACTION TO LAST (EARLIER FIRST AND LATEST LAST)**

```
CREATE OR REPLACE PROCEDURE PRINT_PASSBOOK
IS
    ACCNO ACCOUNT.ACC_NO%TYPE;
    R1 ACCOUNT%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('ENTER ACCOUNT NO : '||1200);
    ACCNO:=1200;
    SELECT * INTO R1 FROM ACCOUNT WHERE ACC_NO=ACCNO;
    DECLARE
    CURSOR C1 IS SELECT * FROM TRANSACTION WHERE SENDER_ACC=ACCNO OR
    RECEIVER_ACC=ACCNO ORDER BY ROWID;
    R2 C1%ROWTYPE;
    MSG VARCHAR2(100);
    BEGIN
    FOR R2 IN C1 LOOP
    IF(R2.SENDER_ACC=ACCNO) THEN
        IF(R2.RECEIVER_ACC IS NOT NULL) THEN
            MSG:='SENT '||R2.AMT||' TO '||' ACCOUNT NO '||R2.RECEIVER_ACC||' WITH MOTO
            '||R2.MOTTO||' AT TIME '||TO_CHAR(R2.TRANS_DT,'DD-MM-YYYY HH:MI:SS AM');
        ELSE
            MSG:='SELF WITHDRAWAL OF '||R2.AMT||' WITH MOTO '||R2.MOTTO||' AT TIME
            '||TO_CHAR(R2.TRANS_DT,'DD-MM-YYYY HH:MI:SS AM');

        END IF;
    ELSE
        IF(R2.SENDER_ACC IS NOT NULL) THEN
            MSG:='RECEIVED '||R2.AMT||' FROM '||' ACCOUNT NO '||R2.SENDER_ACC||' WITH MOTO
            '||R2.MOTTO||' AT TIME '||TO_CHAR(R2.TRANS_DT,'DD-MM-YYYY HH:MI:SS AM');
        ELSIF(R2.TYPE='I') THEN
            MSG:='INTEREST DEPOSITE OF '||R2.AMT||' WITH MOTO '||R2.MOTTO||' AT TIME
            '||TO_CHAR(R2.TRANS_DT,'DD-MM-YYYY HH:MI:SS AM');
        ELSE
            MSG:='SELF DEPOSITE '||R2.AMT||' WITH MOTO '||R2.MOTTO||' AT TIME
            '||TO_CHAR(R2.TRANS_DT,'DD-MM-YYYY HH:MI:SS AM');
        END IF;
    END IF;
    DBMS_OUTPUT.PUT_LINE(MSG);
    END LOOP;
END;
END;
```

**11) ASKS USER FOR TRANSACTION METHOD FOR DECIDING TYPE OF TRANSACTION AND EXECUTE SUCCESSFUL OR UNSUCCESSFUL INSERT QUERY ON TRANSACTION**

```
CREATE OR REPLACE PROCEDURE PERFORM_TRANSACTION
```

```
IS
```

```
RES NUMBER(1);
```

```
TYPE1 TRANSACTION.TYPE%TYPE;
```

```
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('CHOOSE TRANSACTION MEDIUMS ');
```

```
DBMS_OUTPUT.NEW_LINE;
```

```
DBMS_OUTPUT.PUT_LINE('1.VIA CREDIT CARD ');
```

```
DBMS_OUTPUT.NEW_LINE;
```

```
DBMS_OUTPUT.PUT_LINE('2.VIA DEBIT CARD ');
```

```
DBMS_OUTPUT.NEW_LINE;
```

```
DBMS_OUTPUT.PUT_LINE('3.VIA UPI ');
```

```
DBMS_OUTPUT.NEW_LINE;
```

```
DBMS_OUTPUT.PUT_LINE('4.SELF PAYMENT (DEPOSITE OR WITHDRAWAL) ');
```

```
DBMS_OUTPUT.NEW_LINE;
```

```
DBMS_OUTPUT.PUT_LINE('CHOOSE OPTION : '||4);
```

```
RES :=4;
```

```
CASE RES
```

```
WHEN 1 THEN TYPE1:='C';
```

```
WHEN 2 THEN TYPE1:='DC';
```

```
WHEN 3 THEN TYPE1:='U';
```

```
WHEN 4 THEN TYPE1:='S';
```

```
ELSE DBMS_OUTPUT.PUT_LINE('ENTER OPTION FROM DESCRIBED OPTIONS');
```

```
END CASE;
```

```
IF (RES >= 1 AND RES <= 4) THEN
```

```
INSERT INTO TRANSACTION
```

```
VALUES(GET_TR_ID(),NULL,NULL,SYSDATE,NULL,TYPE1,NULL,NULL,NULL);
```

```
END IF;
```

```
END PERFORM_TRANSACTION;
```

```
/
```



**12)IF METHOD OF TRANSACTION IS UPI THEN PROCEDURE ASKS FOR SENDER UPI,RECEIVER UPI,AMOUNT AND IF UPI IS NOT REGISTERED AGAINST ANY ACCOUNT OR PIN IS WRONG THEN RETURN APPROPRIATE ERROR STATEMENT TO CALLING BLOCK**

```
CREATE OR REPLACE PROCEDURE TRANSACTION_VIA_UPI(
    S_UPI OUT UPI.UPI_ID%TYPE,
    R_UPI OUT UPI.UPI_ID%TYPE,
    S_ACC OUT ACCOUNT.ACC_NO%TYPE,
    R_ACC OUT ACCOUNT.ACC_NO%TYPE,
    AMT OUT TRANSACTION.AMT%TYPE,
    ERR OUT TRANSACTION.MOTTO%TYPE
)
IS
    PIN UPI.UPI_PIN%TYPE;
    TRUE_PIN UPI.UPI_PIN%TYPE;
    BEGIN
        ERR:=NULL;
        DBMS_OUTPUT.PUT_LINE('ENTER YOUR UPI ID : '||12);
        S_UPI:=12;
        SELECT UPI_PIN INTO TRUE_PIN FROM UPI WHERE UPI_ID=S_UPI;
        DBMS_OUTPUT.PUT_LINE('ENTER UPI PIN : '||6789||TRUE_PIN);
        PIN:=6789;
        IF(PIN=TRUE_PIN)THEN
            DBMS_OUTPUT.PUT_LINE('ENTER RECEIVERS UPI ID : '||15);
            R_UPI:=15;
            DBMS_OUTPUT.PUT_LINE('ENTER AMOUNT : '||1200);
            AMT:=1200;
            SELECT ACC_NO INTO S_ACC FROM UPI WHERE UPI_ID=S_UPI;
            SELECT ACC_NO INTO R_ACC FROM UPI WHERE UPI_ID=R_UPI;
        ELSE
            ERR:='PIN OR CVV IS WRONG';
            S_UPI:=NULL;
            R_UPI:=NULL;
            S_ACC:=NULL;
            R_ACC:=NULL;
            AMT:=NULL;
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            ERR:='NO SENDER OR RECEIVER UPI EXIST';
            S_UPI:=NULL;
            R_UPI:=NULL;
            S_ACC:=NULL;
```



```
R_ACC:=NULL;  
AMT:=NULL;  
END;  
/
```

**13)DEPOSITE MONTHLY INTEREST IN EVERY ACCOUNT IN BANK ACCORDING TO ACCOUNT TYPE AND SENIORITY OF MAIN ACCOUNT HOLDER.**

```
CREATE OR REPLACE PROCEDURE INTEREST_PAY  
IS  
    CURSOR C1 IS SELECT * FROM ACCOUNT;  
    R1 C1%ROWTYPE;  
BEGIN  
    FOR R1 IN C1 LOOP  
        INSERT INTO TRANSACTION VALUES  
(GET_TR_ID(),NULL,R1.ACC_NO,SYSDATE,R1.TEMP_INTEREST,'I',NULL,NULL,'IPAY');  
    END LOOP;  
END;  
/
```

**14)IN CASE OF SELF DEPOSITE OR SELF WITHDRAWAL ASKS FOR ACCOUNT NO AND AMOUNT SENDS IF ACCOUNT IS REGISTERED IN BANK OTHERWISE APPROPRIATE ERROR STATEMENT.**

```
CREATE OR REPLACE PROCEDURE SELF_TRANSACTION(  
    S_ACC OUT ACCOUNT.ACC_NO%TYPE,  
    R_ACC OUT ACCOUNT.ACC_NO%TYPE,  
    AMT OUT TRANSACTION.AMT%TYPE,  
    TYPE OUT UPI.UPI_PIN%TYPE,  
    ERR OUT TRANSACTION.MOTTO%TYPE  
)  
IS  
    R1 ACCOUNT%ROWTYPE;  
BEGIN  
    ERR:=NULL;  
    DBMS_OUTPUT.PUT_LINE('1.WITHDRAWAL 2.DEPOSITE : '||2);  
    TYPE:=2;  
    DBMS_OUTPUT.PUT_LINE('ENTER AMOUNT : '||1200);  
    AMT:=1200;  
    DBMS_OUTPUT.PUT_LINE('ENTER YOUR ACCOUNT NO : '||1200);  
  
    IF TYPE=1 THEN  
        S_ACC:=1200;
```



```
SELECT * INTO R1 FROM ACCOUNT WHERE ACC_NO=S_ACC;
R_ACC:=NULL;
ELSE
  R_ACC:=1200;
SELECT * INTO R1 FROM ACCOUNT WHERE ACC_NO=R_ACC;
S_ACC:=NULL;

END IF;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    ERR:='NO SENDER ACCOUNT EXIST';
    S_ACC:=NULL;
    R_ACC:=NULL;
    AMT:=NULL;
    TYPE:=NULL;

END;
/
```

**15)IF METHOD OF TRANSACTION IS CREDIT CARD THEN PROCEDURE ASKS FOR SENDER CREDIT CARD NO,RECEIVER ACCOUNT,AMOUNT AND IF CREDIT CARD OR RECEIVER ACCOUNT IS NOT REGISTERED AGAINST ANY ACCOUNT OR PIN(AND CVV) IS WRONG OR CREDIT CARD IS BLOCKED THEN RETURN APPROPRIATE ERROR STATEMENT TO CALLING BLOCK**

```
CREATE OR REPLACE PROCEDURE TRANSACTION_VIA_CC(
  S_CC OUT UPI.UPI_ID%TYPE,
  S_ACC OUT ACCOUNT.ACC_NO%TYPE,
  R_ACC OUT ACCOUNT.ACC_NO%TYPE,
  AMT OUT TRANSACTION.AMT%TYPE,
  ERR OUT TRANSACTION.MOTTO%TYPE
)
IS
  PIN CREDIT.PIN%TYPE;
  TRUE_PIN CREDIT.PIN%TYPE;
  CVV CREDIT.CVV_NO%TYPE;
  TRUE_CVV CREDIT.CVV_NO%TYPE;
  ISB CREDITDUE.ISBLOCKED%TYPE;
  TEMP NUMBER(2);
  R1 ACCOUNT%ROWTYPE;
BEGIN
```



```
ERR:=NULL;
DBMS_OUTPUT.PUT_LINE('ENTER YOUR CREDIT CARD NO : '||11);
S_CC:=11;
ISB:=NULL;
SELECT COUNT(*) INTO TEMP FROM CREDITDUE WHERE CREDIT_CARD_NO=S_CC;
IF(TEMP!=0) THEN
SELECT ISBLOCKED INTO ISB FROM CREDITDUE WHERE CREDIT_CARD_NO=S_CC ;
END IF;
IF(ISB IS NULL OR ISB='N') THEN
    DBMS_OUTPUT.PUT_LINE('ENTER AMOUNT : '||10);
    AMT:=10;
    SELECT PIN INTO TRUE_PIN FROM CREDIT WHERE CREDIT_CARD_NO=S_CC;
    SELECT CVV_NO INTO TRUE_CVV FROM CREDIT WHERE CREDIT_CARD_NO=S_CC;
    DBMS_OUTPUT.PUT_LINE('ENTER PIN : '||2009);
    PIN:=2009;
    DBMS_OUTPUT.PUT_LINE('ENTER CVV NO : '||2341);
    CVV:=2341;
    IF(PIN=TRUE_PIN AND CVV=TRUE_CVV)THEN
        DBMS_OUTPUT.PUT_LINE('ENTER RECEIVERS ACC NO : '||1201);
        R_ACC:=1201;
        SELECT * INTO R1 FROM ACCOUNT WHERE ACC_NO=R_ACC;
        SELECT ACC_NO INTO S_ACC FROM CREDIT WHERE CREDIT_CARD_NO=S_CC;
    ELSE
        ERR:='WRONG CVV OR PIN';
        S_CC:=NULL;
        S_ACC:=NULL;
        R_ACC:=NULL;
        AMT:=NULL;
    END IF;
ELSE
    ERR:='YOUR CREDIT CARD IS BLOCKED'||TEMP;
    S_CC:=NULL;
    S_ACC:=NULL;
    R_ACC:=NULL;
    AMT:=NULL;
END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        ERR:='NO SENDER CREDIT CARD OR RECEIVER ACCOUNT EXIST';
        S_CC:=NULL;
        S_ACC:=NULL;
        R_ACC:=NULL;
        AMT:=NULL;
```



END;

/

**16) IF METHOD OF TRANSACTION IS DEBIT CARD THEN PROCEDURE ASKS FOR SENDER DEBIT CARD NO,RECEIVER ACCOUNT,AMOUNT AND IF DEBIT CARD OR RECEIVER ACCOUNT IS NOT REGISTERED AGAINST ANY ACCOUNT OR PIN(AND CVV) IS WRONG OR AMOUNT EXCEEDS LIMIT OF DEBIT CARD THEN RETURN APPROPRIATE ERROR STATEMENT TO CALLING BLOCK**

```
CREATE OR REPLACE PROCEDURE TRANSACTION_VIA_DC(  
S_ACC OUT ACCOUNT.ACC_NO%TYPE,  
R_ACC OUT ACCOUNT.ACC_NO%TYPE,  
AMT OUT TRANSACTION.AMT%TYPE,  
ERR OUT TRANSACTION.MOTTO%TYPE  
)  
IS
```

```
PIN DEBIT.PIN%TYPE;  
TRUE_PIN DEBIT.PIN%TYPE;  
CVV DEBIT.CVV_NO%TYPE;  
TRUE_CVV DEBIT.CVV_NO%TYPE;  
LIM DEBIT.SET_LIMIT%TYPE;  
S_DC DEBIT.DEBIT_CARD_NO%TYPE;  
R1 ACCOUNT%ROWTYPE;
```

```
BEGIN
```

```
ERR:=NULL;  
DBMS_OUTPUT.PUT_LINE('ENTER YOUR DEBIT CARD NO : '||13);  
S_DC:=13;
```

```
DBMS_OUTPUT.PUT_LINE('ENTER AMOUNT : ');  
AMT:=90;  
SELECT SET_LIMIT INTO LIM FROM DEBIT WHERE DEBIT_CARD_NO=S_DC;  
IF(AMT<LIM) THEN
```

```
SELECT CVV_NO INTO TRUE_CVV FROM DEBIT WHERE DEBIT_CARD_NO=S_DC;  
SELECT PIN INTO TRUE_PIN FROM DEBIT WHERE DEBIT_CARD_NO=S_DC;  
DBMS_OUTPUT.PUT_LINE('ENTER PIN : ');  
PIN:=1009;  
DBMS_OUTPUT.PUT_LINE('ENTER CVV NO : ');  
CVV:=3341;
```



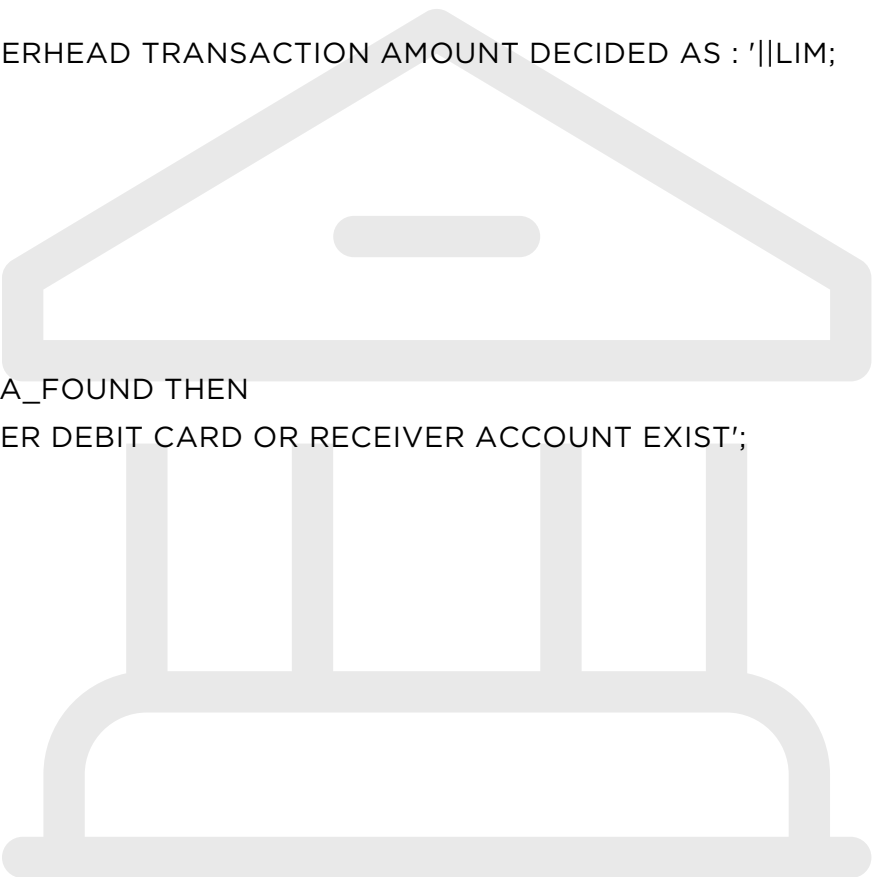
```
IF(PIN=TRUE_PIN AND CVV=TRUE_CVV)THEN
DBMS_OUTPUT.PUT_LINE('ENTER RECEIVERS ACC NO : ');
R_ACC:=1201;
SELECT * INTO R1 FROM ACCOUNT WHERE ACC_NO=R_ACC;
SELECT ACC_NO INTO S_ACC FROM DEBIT WHERE DEBIT_CARD_NO=S_DC;
ELSE

ERR:='WRONG CVV OR PIN';

S_ACC:=NULL;
R_ACC:=NULL;
AMT:=NULL;
END IF;
ELSE
ERR:='CANT OVERHEAD TRANSACTION AMOUNT DECIDED AS : '||LIM;
S_ACC:=NULL;
R_ACC:=NULL;
AMT:=NULL;
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
ERR:='NO SENDER DEBIT CARD OR RECEIVER ACCOUNT EXIST';
S_ACC:=NULL;
R_ACC:=NULL;
AMT:=NULL;

END;
/
```





**17)SCHEDULED PROCEDURE : TO ADD INTEREST IN CURRENT AMT IN FD AT MATURITY\_INTERVAL\_DT.**

```
CREATE OR REPLACE PACKAGE FD_INTEREST_PCK
```

```
AS
```

```
  PROCEDURE FD_INTEREST_PROC();
```

```
END FD_INTEREST_PCK;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY FD_INTEREST_PCK
```

```
AS
```

```
  PROCEDURE FD_INTEREST_PROC()
```

```
  AS
```

```
  CURSOR C1 IS SELECT * FROM FD;
```

```
  INTERVAL C1.MT_INTERVAL%TYPE;
```

```
  NEXT_MATURITY_DT DATE;
```

```
  BEGIN
```

```
  FOR R1 IN C1
```

```
  LOOP
```

```
  SELECT MT_INTERVAL INTO INTERVAL FROM FD WHERE
```

```
  R1.FD_ID=FD_ID;
```

```
  IF R1.PREV_MT_DT=NULL THEN
```

```
  SELECT ADDDATE(MONTH,INTERVAL,R1.DEPOSITION_DT) INTO NEXT_MATURITY_DT;
```

```
  ELSE
```

```
  SELECT ADDDATE(MONTH,INTERVAL,R1.PREV_MT_DT) INTO NEXT_MATURITY_DT;
```

```
  END IF;
```

```
  IF TO_DATE(NEXT_MATURITY_DATE,'DD-MM-YY') =
```

```
  TO_DATE(SYSDATE,'DD-MM-YY') THEN
```

```
  R1.CURR_AMT := R1.CURR_AMT + (R1.CURRENT_AMT *
```

```
  R1.INTEREST_RT)/100;
```

```
  UPDATE FD SET CURR_AMT = R1.CURR_AMT, PREV_MT_DT=SYSDATE WHERE
```

```
  R1.FD_ID=FD_ID;
```

```
  END IF;
```

```
  END LOOP;
```

```
  COMMIT;
```

```
  EXCEPTION
```

```
  WHEN NO_DATA_FOUND THEN
```

```
  DBMS_OUTPUT.PUT_LINE('FD DOES NOT EXIST');
```



```
END FD_INTEREST_PROC;  
END FD_INTEREST_PCK;  
/
```

**SCHEDULE : -**

```
BEGIN  
  DBMS_SCHEDULER.CREATE_JOB (  
    JOB_NAME => 'FD INTEREST SCHEDULER',  
    JOB_TYPE => 'STORED_PROCEDURE',  
    JOB_ACTION => 'FD_INTEREST_PCK.FD_INTEREST_PROC',  
    START_DATE => SYSTIMESTAMP,  
    REPEAT_INTERVAL => 'FREQ=DAILY; INTERVAL=1',  
    ENABLED => TRUE);  
END;  
/
```

**18)SCHEDULED PROCEDURE : TO CUT EMI OF LOAN MONTHLY.**

```
CREATE OR REPLACE PACKAGE LOAN_EMI_PCK  
AS  
  PROCEDURE LOAN_EMI_PROC()  
END LOAN_EMI_PCK;  
/
```

```
CREATE OR REPLACE PACKAGE BODY LOAN_EMI_PCK  
AS  
  PROCEDURE LOAN_EMI_PROC()  
  AS  
    CURSOR C1 IS SELECT * FROM LOAN;  
    CHECK_DT DATE;  
    T_ID NUMBER(30);  
  BEGIN  
    FOR R1 IN C1  
    LOOP  
      SELECT ISSUE_DT INTO CHECK_DT FROM LOAN WHERE L_ID=R1.L_ID;  
      WHILE TO_CHAR(CHECK_DT,'DD-MM-YYYY')  
        <= TO_CHAR(SYSDATE,'DD-MM-YYYY')  
      LOOP  
        SELECT ADDDATE(MONTHS,1,CHECK_DT) INTO CHECK_DT;  
        IF TO_CHAR(SYSDATE,'DD-MM-YYYY') =  
          TO_CHAR(CHECK_DT,'DD-MM-YYYY') THEN  
          INSERT INTO PENDING_EMI VALUES(R1.EMI, SYSDATE, R1.L_ID);
```



```
T_ID := GET_TR_ID();
INSERT INTO TRANSACTION VALUES(T_ID,R1.ACCNO,NULL,SYSDATE,
R1.EMI,ET,R1.BALANCE-R1.EMI,R1.L_ID);

END IF;
END LOOP;
END LOOP;
COMMIT;
END LOAN_EMI_PROC;
END LOAN_EMI_PCK;
/
```

**SCHEDULE:-**

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
JOB_NAME => 'LOAN EMI SCHEDULER',
JOB_TYPE => 'STORED_PROCEDURE',
JOB_ACTION => 'LOAN_EMI_PCK.LOAN_EMI_PROC',
START_DATE => SYSTIMESTAMP,
REPEAT_INTERVAL => 'FREQ=DAILY; INTERVAL=1',
ENABLED => TRUE);
END;
/
```

**19)SCHEDULED PROCEDURE ON PENDINGCHEQUE.**

**ASSUMPTION :- CREDITORS ACCOUNT AND DEPOSITORS ACCOUNT BELONG TO SAME BANK.**

```
CREATE OR REPLACE PACKAGE PENDING_CHEQUE_PCK
AS
    PROCEDURE PENDING_CHEQUE_PROC();
END PENDING_CHEQUE_PCK;
/
```

```
CREATE OR REPLACE PACKAGE BODY PENDING_CHEQUE_PCK
AS
    PROCEDURE PENDING_CHEQUE_PROC()
    AS
        CURSOR C1 IS SELECT * FROM PENDING_CHEQUE;
        ISSUE_DT DATE;
        REC_ADHAAR_NO NUMBER(12);
        SENDER_ACCNO NUMBER(16);
```



```
EMPNO NUMBER(4);
SEN_BALANCE NUMBER(9,2);
TYPE VARCHAR2(2);
TRANS_ID NUMBER(30);
BEGIN
FOR R1 IN C1
LOOP
SELECT R1.ISSUE_DT INTO ISSUE_DT FROM PENDING_CHEQUE WHERE
CHEQUE_NO = R1.CHEQUE_NO;
IF TO_CHAR(ISSUE_DT+1,'DD-MM-YYYY')=TO_CHAR(SYSDATE,'DD-MM-YYYY')
THEN
IF R1.CHEQUE_TYPE=ORDER_CHECK_VERIFICATION THEN
SELECT ADHAAR_NO INTO REC_ADHAAR_NO FROM ACCOUNT WHERE
R1.ACCNO=ACCNO;
TRANS_ID:=GET_TR_ID();
EMPNO := &EMPNO;
INSERT INTO ORDER_CHECK_VERIFICATION VALUES
( R1.CHEQUE_NO, TRANS_ID, EMPNO, REC_ADHAAR_NO,'N');

SELECT ACCNO INTO SENDER_ACCNO FROM CHEQUE_BOOK WHERE
R1.CHEQUE_NO >= START_NUM AND R1.CHEQUE_NO <= END_NUM;

SELECT BALANCE INTO SEN_BALANCE FROM ACCOUNT WHERE
ACCNO=SENDER_ACCNO;

INSERT INTO TRANSACTION VALUES
( TRANS_ID, SENDER_ACCNO,R1.RECEIVER_ACCNO, SYSDATE, R1.AMT,
'OC', SEN_BALANCE - R1.AMT, R1.CHEQUE_NO);

ELSE
IF R1.CHEQUE_TYPE=AC_PAYEE THEN TYPE := 'AC';
ELSE TYPE := 'BC';
END IF;

SELECT ACCNO INTO SENDER_ACCNO FROM CHEQUE_BOOK WHERE
R1.CHEQUE_NO >= START_NUM AND R1.CHEQUE_NO <= END_NUM;

SELECT BALANCE INTO SEN_BALANCE FROM ACCOUNT WHERE
ACCNO=SENDER_ACCNO;

TRANS_ID:=GET_TR_ID();
INSERT INTO TRANSACTION VALUES
( TRANS_ID, SENDER_ACCNO,R1.RECEIVER_ACCNO, SYSDATE, R1.AMT,
TYPE, SEN_BALANCE - R1.AMT, R1.CHEQUE_NO);
```



```
END IF;  
END IF;  
END LOOP;  
  
END PENDING_CHEQUE_PROC;  
END PENDING_CHEQUE_PCK;  
/
```

**SCHEDULE:-**

```
BEGIN  
DBMS_SCHEDULER.CREATE_JOB (  
  JOB_NAME => 'PENDING CHEQUE SCHEDULER',  
  JOB_TYPE => 'STORED_PROCEDURE',  
  JOB_ACTION => 'PENDING_CHEQUE_PCK.PENDING_CHEQUE_PROC',  
  START_DATE => SYSTIMESTAMP,  
  REPEAT_INTERVAL => 'FREQ=DAILY; INTERVAL=1',  
  ENABLED => TRUE);  
END;  
/
```

**20) A PROCEDURE TO ENTER A RECORD IN THE CREDIT\_DUE TABLE IF EXPENSES OF THE CREDIT CARD EXCEED THE LIMIT OF THAT CREDIT CARD :**

```
CREATE OR REPLACE PROCEDURE CREDITDUEAMOUNT(CNO IN NUMBER, AMT IN  
NUMBER, INTR IN NUMBER, LIMIT IN NUMBER)  
IS  
  R1 CREDIT_DUE%ROWTYPE;  
BEGIN  
  IF(CHECKEXISTANCEINCREDITDUE(CNO)) THEN  
    SELECT * INTO R1 FROM CREDIT_DUE  
    WHERE CREDIT_CARD_NO = CNO;  
  
    R1.FINE_AMT := R1.FINE_AMT + GENERATEDUEAMOUNTPENALTY(AMT - LIMIT) +  
    CALCULATECREDITINTEREST(R1.DUE_AMT, ROUND(SYSDATE - R1.DUE_DT));  
  
    UPDATE CREDIT_DUE  
    SET DUE_DT = SYSDATE,  
        DUE_AMT = R1.DUE_AMT + AMT,  
        FINE_AMT = R1.FINE_AMT + INTR,  
        ISBLOCKED = 'Y'  
    WHERE CREDIT_CARD_NO = CNO;
```



ELSE

INSERT INTO CREDIT\_DUE

VALUES(CNO, SYSDATE, AMT, GENERATEDUEAMOUNTPENALTY(AMT - LIMIT), 'N');

END IF;

UPDATE CREDIT

SET EXPENSE = 0,

INTEREST\_AMT = 0

WHERE CREDIT\_CARD\_NO = CNO;

COMMIT;

END CREDITDUEAMOUNT;

/





# FUNCTIONS

## 1) RETURNS TRUE IF AGE IS BELOW 18 ELSE RETURNS FALSE

```
CREATE OR REPLACE FUNCTION ISMINOR(C CUSTOMER.C_ID%TYPE) RETURN BOOLEAN
IS
    D CUSTOMER.DT_BIRTH%TYPE;
BEGIN
    SELECT DT_BIRTH INTO D FROM CUSTOMER WHERE C_ID = C;
    IF((SYSDATE - D)/365 < 18) THEN
        RETURN TRUE;
    END IF;
    RETURN FALSE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO DATA FOUND');
END ISMINOR;
/
```

## 2) RETURNS TRUE IF A CUSTOMER IS SENIOR CITIZEN (ABOVE 60 YEARS AGE) OTHERWISE RETURNS FALSE.

```
CREATE OR REPLACE FUNCTION ISSENIOR(C CUSTOMER.C_ID%TYPE) RETURN BOOLEAN
IS
    D CUSTOMER.DT_BIRTH%TYPE;
BEGIN
    SELECT DT_BIRTH INTO D FROM CUSTOMER WHERE C_ID = C;
    IF((SYSDATE - D)/365 >= 60) THEN
        RETURN TRUE;
    END IF;
    RETURN FALSE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO DATA FOUND');
END ISSENIOR;
/
```







```
IF(FINE < MINIMUM) THEN FINE := MINIMUM;
END IF;
RETURN FINE;
END GENERATEDUEAMOUNTPENALTY;
/
```

**5)FUNCTION TO CALCULATE THE PENALTY FOR A CREDIT CARD BILL, WHICH HAS NOT BEEN PAID BEFORE THE DUE DATE :**

```
CREATE OR REPLACE FUNCTION GENERATEDUEDATEPENALTY(AMT IN NUMBER)
RETURN NUMBER
IS
BEGIN
CASE
WHEN(AMT <= 500) THEN RETURN 0;
WHEN(AMT <= 1000) THEN RETURN 400;
WHEN(AMT <= 10000) THEN RETURN 750;
WHEN(AMT <= 25000) THEN RETURN 950;
WHEN(AMT <= 50000) THEN RETURN 1100;
ELSE RETURN 1300;
END CASE;
END GENERATEDUEDATEPENALTY;
/
```

**6)FUNCTION TO CHECK IF A CREDIT CARD NUMBER PROVIDED BY THE USER IS VALID OR NOT :**

```
CREATE OR REPLACE FUNCTION CHECKCREDITNUMBERVALIDITY(CNO IN NUMBER)
RETURN BOOLEAN
IS
TEMP CREDIT.CREDIT_CARD_NO%TYPE;
BEGIN
SELECT CREDIT_CARD_NO INTO TEMP FROM CREDIT
WHERE CREDIT_CARD_NO = CNO;
RETURN TRUE;

EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN FALSE;
END CHECKCREDITNUMBERVALIDITY;
/
```

**7)FUNCTION TO CHECK WHETHER A CREDIT CARD IS DUE OR NOT(EITHER BY AMOUNT OR BY DATE) :**

CREATE OR REPLACE FUNCTION CHECKEXISTANCEINCREDITDUE(CNO IN NUMBER)  
RETURN BOOLEAN

IS

TEMP CREDIT.CREDIT\_CARD\_NO%TYPE;

BEGIN

SELECT CREDIT\_CARD\_NO INTO TEMP FROM CREDIT\_DUE

WHERE CREDIT\_CARD\_NO = CNO;

RETURN TRUE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

END CHECKEXISTANCEINCREDITDUE;

/

**8)FUNCTION TO CALCULATE THE INTEREST ON THE EXPENSE OF A CREDIT CARD FOR A GIVEN NUMBER OF DAYS :**

CREATE OR REPLACE FUNCTION CALCULATECREDITINTEREST(AMT IN NUMBER,  
DURATION IN NUMBER)

RETURN NUMBER

IS

MPR CONSTANT NUMBER(2, 1) := 3.5;

BEGIN

RETURN ((DURATION \* AMT \* MPR \* 12) / 36500);

END CALCULATECREDITINTEREST;

/



# TRIGGERS

## 1) IT CHECKS WHETHER MAJOR ACCOUNT HOLDER IS ADULT(ABOVE AGE 18) OR NOT.

```
CREATE OR REPLACE TRIGGER CHECKFORMINOR
BEFORE INSERT ON ACCOUNT
FOR EACH ROW
BEGIN
  IF(ISMINOR(:NEW.MAJOR_CID)) THEN
    RAISE_APPLICATION_ERROR(-20001,'MAJOR ACCOUNT HOLDER CANNOT BE MINOR');
  END IF;
END CHECKFORMINOR;
/
```

## 2)IT CHECKS WHETHER OPENING ACCOUNT BALANCE IS ABOVE THE REQUIREMENTS.

```
CREATE OR REPLACE TRIGGER INITIALBALANCE
BEFORE INSERT ON ACCOUNT
FOR EACH ROW
DECLARE
  MINBAL NUMBER;
BEGIN

  IF(:NEW.ACC_TYPE LIKE '%SAVING%') THEN
    SELECT MINIMUM_BAL INTO MINBAL FROM SAVING;

    IF(:NEW.BALANCE < MINBAL) THEN
      RAISE_APPLICATION_ERROR(-20000,'OPENING BALANCE MUST BE GREATER OR EQUAL TO
      '|MINBAL);
    END IF;

  ELSIF(:NEW.ACC_TYPE LIKE '%CURRENT%') THEN
    SELECT MINIMUM_BAL INTO MINBAL FROM CURRENTACCOUNT;

    IF(:NEW.BALANCE < MINBAL) THEN
      RAISE_APPLICATION_ERROR(-20000,'OPENING BALANCE MUST BE GREATER OR EQUAL TO
      '|MINBAL);
    END IF;

  END IF;
```



EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('NO DATA FOUND.');

END INITIALBALANCE;

/

**3)EVERY TIME AFTER TRANSACTION IT CHECKS IF THE ACCOUNT BALANCE IS ABOVE MINIMUM REQUIRED BALANCE IF NOT THEN INSERTS THE VALUE IN ACCOUNT PENALTY**

CREATE OR REPLACE TRIGGER CHECKFORMINBAL

AFTER UPDATE ON ACCOUNT

FOR EACH ROW

DECLARE

MBAL NUMBER;

BEGIN

IF(:NEW.ACC\_TYPE LIKE '%SAVING%') THEN

SELECT MIN\_BAL INTO MBAL FROM SAVING;

IF(:NEW.BALANCE < MBAL) THEN

INSERT INTO ACCOUNTPENALTY VALUES(:NEW.ACC\_NO,SYSDATE,0);

END IF;

ELSIF(:NEW.ACC\_TYPE LIKE '%CURRENT%') THEN

SELECT MIN\_BAL INTO MBAL FROM CURRENTACCOUNT;

IF(:NEW.BALANCE < MBAL) THEN

INSERT INTO ACCOUNTPENALTY VALUES(:NEW.ACC\_NO,SYSDATE,0);

END IF;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('NO DATA FOUND.');

END CHECKFORMINBAL;

/

**4)TRIGGER TO CHECK PENALTY OF FD.**

CREATE OR REPLACE TRIGGER FD\_PENALTY

BEFORE DELETE ON FD

FOR EACH ROW

DECLARE

NO\_OF\_INTERVALS NUMBER(2);



```
NEW_INT_RT NUMBER(4,2);
BEGIN
IF TO_CHAR(SYSDATE,'DD-MM-YY') < TO_CHAR(DUE_DT,'DD-MM-YY') THEN
:OLD.INTEREST := :OLD.INTEREST - PENALTY;

NO_OF_INTERVALS := ROUND(MONTHS_BETWEEN(:OLD.PREV_MT_DT,
:OLD.DEPOSITION_DT))/:OLD.MT_INTERVAL;

:OLD.CURR_AMT := :OLD.INIT_AMT;
FOR I IN REVERSE 1..NO_OF_INTERVALS
LOOP
:OLD.CURR_AMT := :OLD.CURR_AMT +
(:OLD.INTEREST * :OLD.CURR_AMT)/100;
END LOOP;
END IF;
END FD_PENALTY;
/
```

#### **5)TRIGGER BEFORE CHEQUEBOOK TO SUBTRACT CHEQUEBOOK PRICE FROM BALANCE.**

```
CREATE OR REPLACE TRIGGER CHEQUEBOOKCOST
BEFORE INSERT ON CHEQUEBOOK
FOR EACH ROW
DECLARE
  ACCNO NUMBER(16);
  TRANS_ID NUMBER(30);
  NEW_BALANCE NUMBER(9,2);
  COST NUMBER(5,2) := 100;
BEGIN
  ACCNO := :NEW.ACCNO;
  SELECT BALANCE INTO NEW_BALANCE FROM ACCOUNT WHERE
  ACC_NO = :NEW.ACCNO;

  TRANS_ID := GET_TR_ID();
  INSERT INTO TRANSACTION VALUES ( TRANS_ID, ACCNO , NULL, SYSDATE ,
  COST, CT, NEW_BALANCE,NULL);
END CHEQUEBOOKCOST;
/
```



**6)BEFORE INSERT ON TRANSACTION : CALLS NECESSARY PROCEDURES ACCORDING TO METHOD OF TRANSACTION AND IF SENDER HAS OPTIMUM BALANCE IN ACCOUNT THEN ONLY TRANSACTION IS VALIDATED OTHERWISE RAISES APPROPRIATE ERROR.**

CREATE OR REPLACE TRIGGER T1 BEFORE INSERT ON TRANSACTION

FOR EACH ROW

DECLARE

SID UPI.UPI\_ID%TYPE;

RID UPI.UPI\_ID%TYPE;

SBAL ACCOUNT.BALANCE%TYPE;

RBAL ACCOUNT.BALANCE%TYPE;

ERR TRANSACTION.MOTTO%TYPE;

DAMT CREDITDUE.DUE\_AMT%TYPE;

BEGIN

CASE :NEW.TYPE

WHEN 'U' THEN

TRANSACTION\_VIA\_UPI(SID,RID,:NEW.SENDER\_ACC,:NEW.RECEIVER\_ACC,:NEW.AMT,ERR);

IF (:NEW.SENDER\_ACC=:NEW.RECEIVER\_ACC) THEN

RAISE\_APPLICATION\_ERROR(-20001,ERR);

ELSIF(NVL(:NEW.AMT,0)=0)

THEN

RAISE\_APPLICATION\_ERROR(-20001,ERR);

ELSE

:NEW.MOTTO:='C2C\_UPI';

INSERT INTO UPI\_PAY VALUES(:NEW.TRANS\_ID,SID,RID);

END IF;

WHEN 'C' THEN

RID:=NULL;

TRANSACTION\_VIA\_CC(SID,:NEW.SENDER\_ACC,:NEW.RECEIVER\_ACC,:NEW.AMT,ERR);

IF(NVL(:NEW.AMT,0)=0 OR :NEW.SENDER\_ACC=:NEW.RECEIVER\_ACC)

THEN

RAISE\_APPLICATION\_ERROR(-20001,ERR);

ELSE

:



```
NEW.MOTTO:='C2C_CC';
INSERT INTO CREDIT_PAY VALUES(:NEW.TRANS_ID,SID);

END IF;
WHEN 'DC' THEN
  RID:=NULL;
  SID:=NULL;
  TRANSACTION_VIA_DC(:NEW.SENDER_ACC,:NEW.RECEIVER_ACC,:NEW.AMT,ERR);
  IF(NVL(:NEW.AMT,0)=0 OR :NEW.SENDER_ACC=:NEW.RECEIVER_ACC)
  THEN
    RAISE_APPLICATION_ERROR(-20001,ERR);
  ELSE
    :NEW.MOTTO:='C2C_DC';

END IF;

WHEN 'S' THEN
  SID:=NULL;
  RID:=NULL;

  SELF_TRANSACTION(:NEW.SENDER_ACC,:NEW.RECEIVER_ACC,:NEW.AMT,DAMT,ERR);
  IF(NVL(:NEW.AMT,0)<=0 )
  THEN
    RAISE_APPLICATION_ERROR(-20001,ERR);
  ELSE
    :NEW.MOTTO:='B2C_SELF';
    IF(DAMT=1)THEN

INSERT INTO SELF_PAY VALUES(:NEW.TRANS_ID,'W');
ELSE
  INSERT INTO SELF_PAY VALUES(:NEW.TRANS_ID,'D');
END IF;
END IF;
WHEN 'AC' THEN
  RID:=NULL;
  SID:=NULL;

  IF(NVL(:NEW.AMT,0)<=0 OR :NEW.SENDER_ACC=:NEW.RECEIVER_ACC)
  THEN
    RAISE_APPLICATION_ERROR(-20001,'ENTER VALID AMOUNT AND BOTH ACCOUNTS
MUST BE DIFFERENT');
  ELSE
    DBMS_OUTPUT.PUT_LINE('');
```



```
END IF;
WHEN 'BC' THEN
  RID:=NULL;
  SID:=NULL;
  :NEW.RECEIVER_ACC:=NULL;
  IF(NVL(:NEW.AMT,0)<=0 )
  THEN
    RAISE_APPLICATION_ERROR(-20001,'ENTER VALID AMOUNT');
  ELSE
    DBMS_OUTPUT.PUT_LINE('');
  END IF;
WHEN 'OC' THEN
  RID:=NULL;
  SID:=NULL;
  :NEW.RECEIVER_ACC:=NULL;
  IF(NVL(:NEW.AMT,0)<=0 )
  THEN
    RAISE_APPLICATION_ERROR(-20001,'ENTER VALID AMOUNT');
  ELSE
    DBMS_OUTPUT.PUT_LINE('');
  END IF;
ELSE
  DBMS_OUTPUT.PUT_LINE('');
END CASE;
IF(:NEW.SENDER_ACC IS NOT NULL AND :NEW.TYPE!='C') THEN
  SELECT BALANCE INTO SBAL FROM ACCOUNT WHERE ACC_NO=:NEW.SENDER_ACC;
  IF SBAL>:NEW.AMT THEN
    :NEW.SENDER_INSTANT_BAL:=SBAL-:NEW.AMT;
  ELSE
    RAISE_APPLICATION_ERROR(-20000,'NOT ENOUGH FUNDS');
  END IF;
ELSIF(:NEW.TYPE='C') THEN
  SELECT BALANCE INTO SBAL FROM ACCOUNT WHERE ACC_NO=:NEW.SENDER_ACC;
  :NEW.SENDER_INSTANT_BAL:=SBAL;
END IF;

IF(:NEW.RECEIVER_ACC IS NOT NULL) THEN
  SELECT BALANCE INTO RBAL FROM ACCOUNT WHERE ACC_NO=:NEW.RECEIVER_ACC;
  :NEW.RECEIVER_INSTANT_BAL:=RBAL+:NEW.AMT;
END IF;
END T1;
```





**7) AFTER INSERT ON TRANSACTION : ON SUCCESSFUL TRANSACTION IT WILL INCREMENT DEDUCT AMOUNT FROM USER AND TRANSFER IT TO RECEIVER'S ACCOUNT AND ALSO PERFORM SOME NECESSARY TASKS ON SUCCESSFUL TRANSACTION.**

CREATE OR REPLACE TRIGGER T2 AFTER INSERT ON TRANSACTION

FOR EACH ROW

DECLARE

R1 ACCOUNT%ROWTYPE;

CC CREDIT.CREDIT\_CARD\_NO%TYPE;

BEGIN

CASE :NEW.TYPE

WHEN 'ET' THEN

DELETE FROM PENDING\_EMI WHERE L\_ID =

TO\_NUMBER(:NEW.MOTTO,'9999999999');

UPDATE TRANSACTION SET MOTTO='EMI\_PAY' WHERE TRANS\_ID=:NEW.TRANS\_ID;

WHEN 'OC' THEN

UPDATE ORDER\_CHECK\_VERIFICATION SET TRANS\_ID=:NEW.TRANS\_ID WHERE  
CHECQE\_NO=TO\_NUMBER(:NEW.MOTTO,'999999');

UPDATE TRANSACTION SET MOTTO='OC\_CHECQE' WHERE  
TRANS\_ID=:NEW.TRANS\_ID;

ELSE

DBMS\_OUTPUT.PUT\_LINE("");

END CASE;

IF(:NEW.TYPE='C') THEN

SELECT SENDER\_CC INTO CC FROM CREDIT\_PAY WHERE  
TRANS\_ID=:NEW.TRANS\_ID;

UPDATE CREDIT ON TRANSACTION(CC, :NEW.AMT);

ELSIF (:NEW.SENDER\_ACC IS NOT NULL) THEN

UPDATE ACCOUNT SET BALANCE=:NEW.SENDER\_INSTANT\_BAL WHERE  
ACC\_NO=:NEW.SENDER\_ACC;

END IF;

IF (:NEW.RECEIVER\_ACC IS NOT NULL) THEN

UPDATE ACCOUNT SET BALANCE=:NEW.RECEIVER\_INSTANT\_BAL WHERE  
ACC\_NO=:NEW.RECEIVER\_ACC;

END IF;

UPDATE USEFULL SET RELATT=RELATT+1 WHERE DID=1;

END;

/



**THE END**