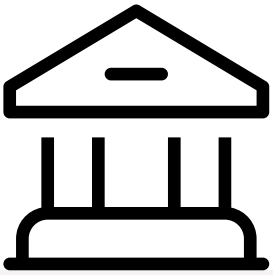
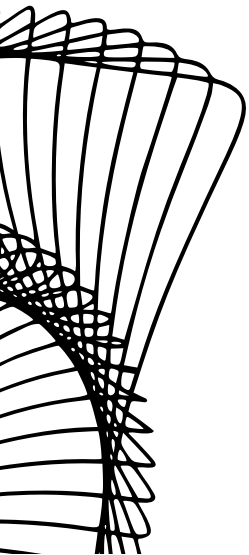
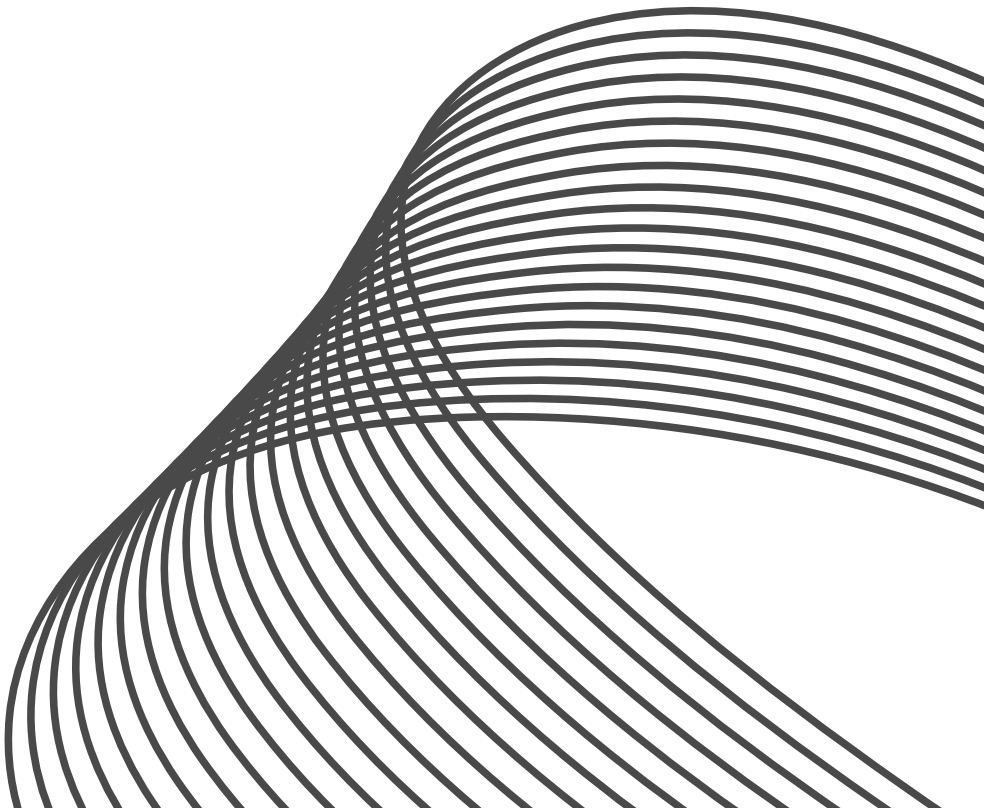


DBMS PROJECT

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

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

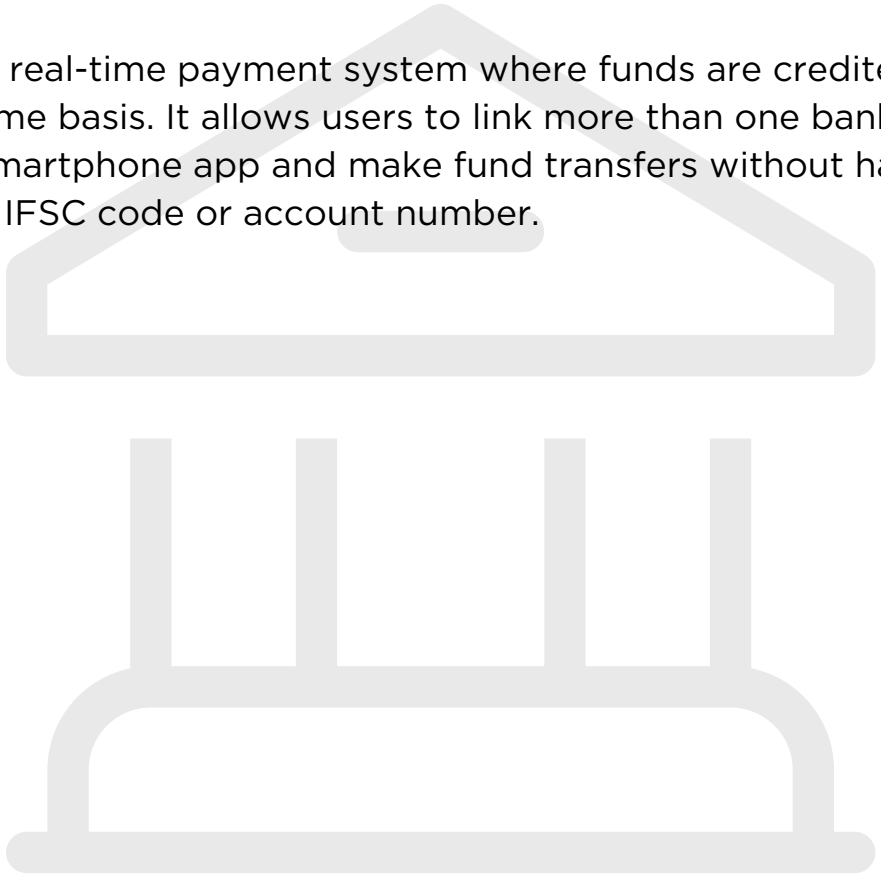
- **BRANCH, DEPARTMENT, EMPLOYEES, CUSTOMER**
  - This segment deals with all branches of a bank along with details of departments that a particular branch contains. The records of employees working in particular department of a branch and records of customer is also maintained. The customer is one who 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.

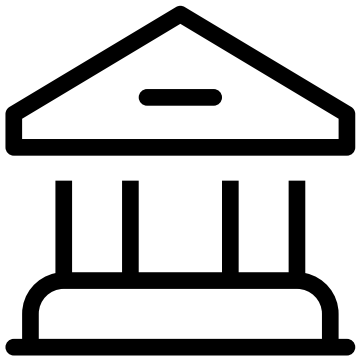


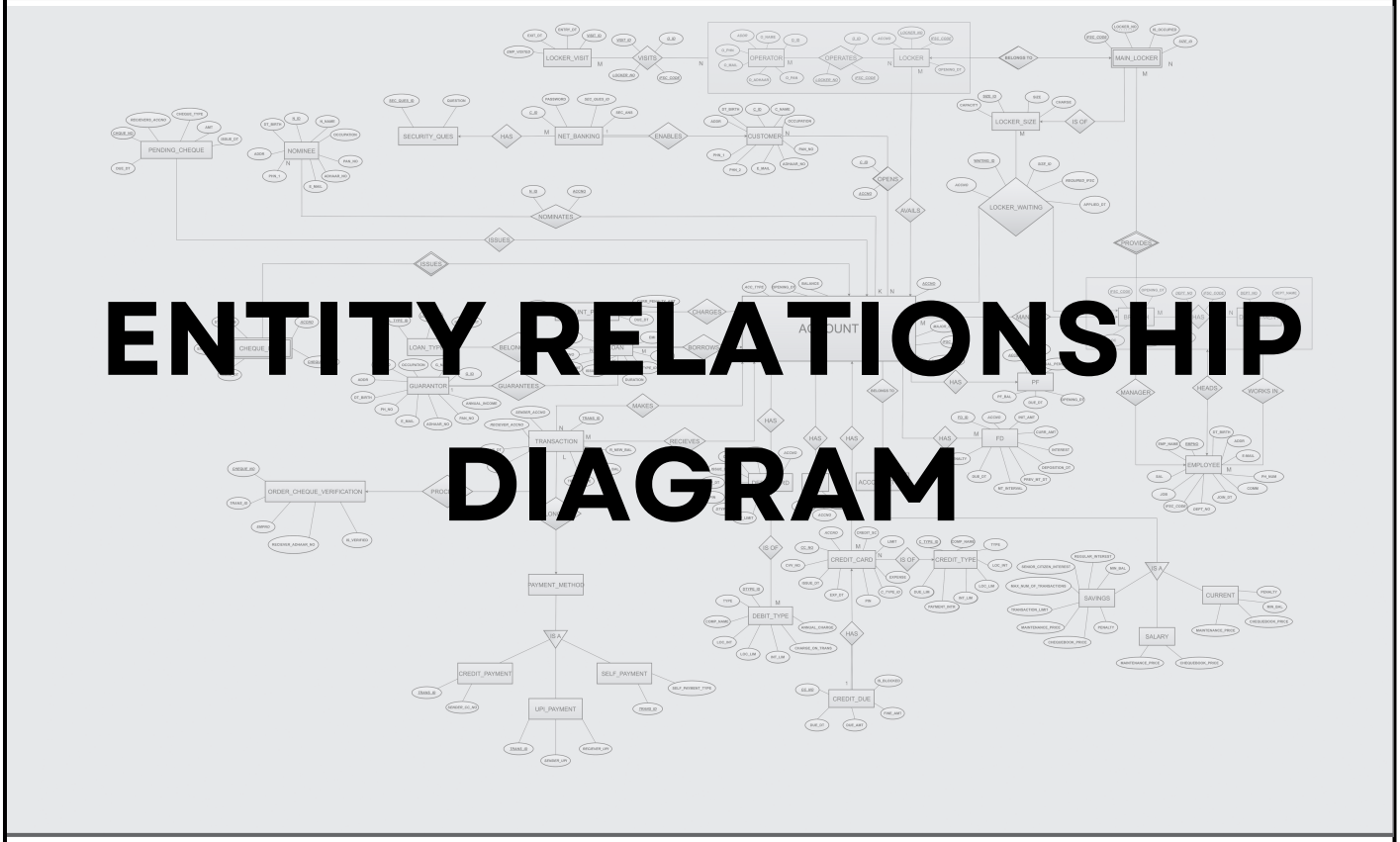
ASSUMPTIONS

- 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.

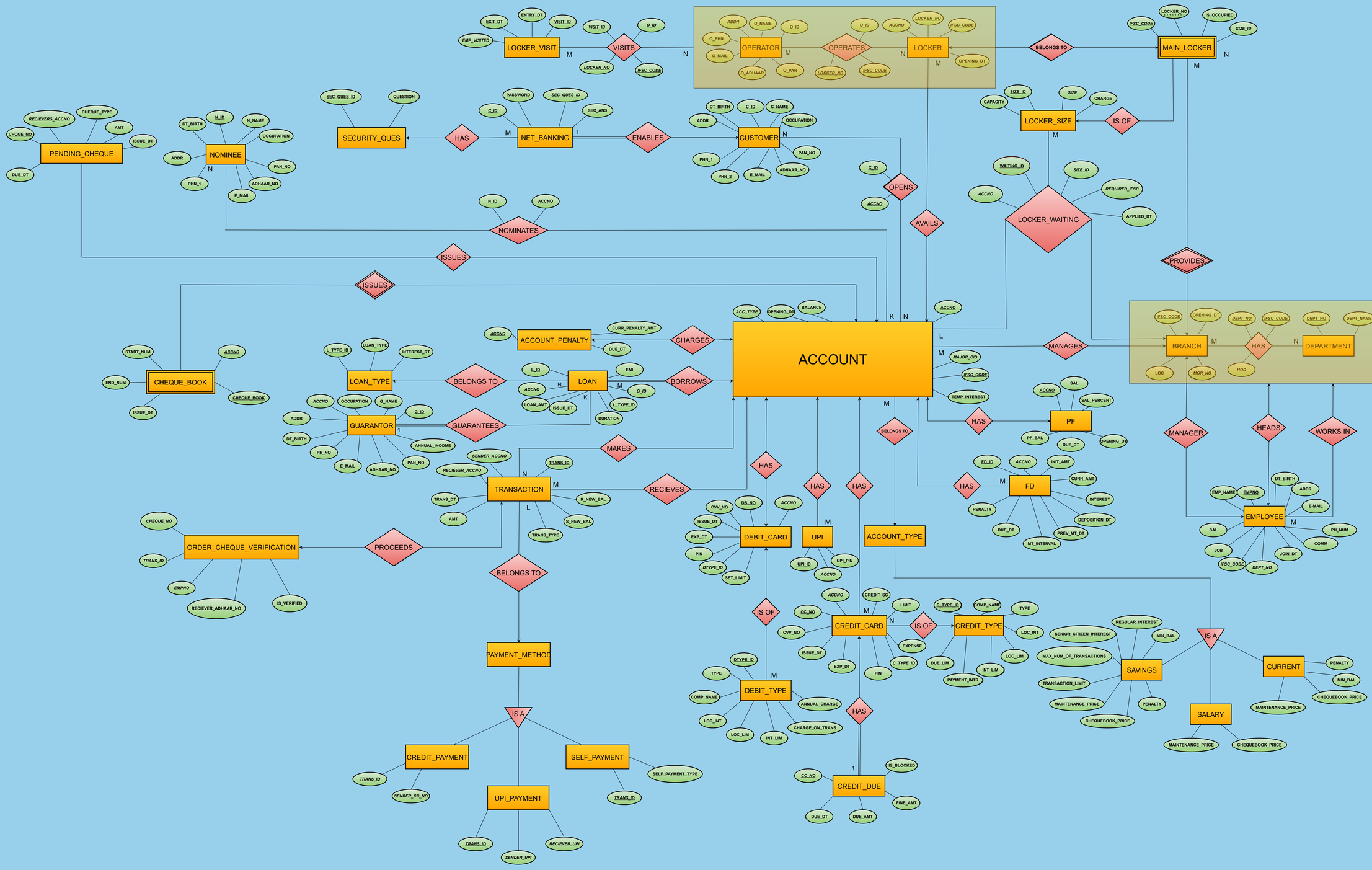
- Interest on FD will be deposited in FD depending upon maturity interval prescribed.
- If someone wants a loan from the bank then the person must have an account in the bank and such person can issue a number of loans and of types prescribed, Interest depends upon loan type.
- Every loan holder must 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.





ENTITY RELATIONSHIP DIAGRAM


LOCKER_VISIT
EXIT_DT
ENTRY_DT
VISIT_ID
EMP_VISITED
VISITS
VISIT_ID
O_ID
LOCKER_NO
IFSC_CODE
OPERATOR
ADDR
O_NAME
O_ID
O_PHN
O_MAIL
O_ADHAAR
O_PAN
OPERATES
O_ID
ACCNO
LOCKER_NO
IFSC_CODE
LOCKER_NO
IFSC_CODE
LOCKER
OPENING_DT
BELONGS TO
MAIN_LOCKER
LOCKER_NO
IS_OCCUPIED
IFSC_CODE
SIZE_ID
LOCKER_SIZE
SIZE_ID
CAPACITY
SIZE
CHARGE
IS OF
PENDING_CHEQUE
RECIEVERS_ACCNO
CHEQUE_TYPE
AMT
CHQUE_NO
ISSUE_DT
DUE_DT
NOMINEE
N_ID
N_NAME
DT_BIRTH
OCCUPATION
ADDR
PAN_NO
PHN_1
ADHAAR_NO
E_MAIL
SECURITY_QUES
SEC_QUES_ID
QUESTION
HAS
NET_BANKING
PASSWORD
SEC_QUES_ID
C_ID
SEC_ANS
ENABLES
CUSTOMER
DT_BIRTH
C_ID
C_NAME
ADDR
OCCUPATION
PAN_NO
PHN_1
PHN_2
E_MAIL
ADHAAR_NO
OPENS
C_ID
ACCNO
AVAILS
LOCKER_WAITING
WAITING_ID
ACCNO
SIZE_ID
REQUIRED_IFSC
APPLIED_DT
NOMINATES
N_ID
ACCNO
ISSUES
ISSUES
PROVIDES
ACCOUNT
ACC_TYPE
OPENING_DT
BALANCE
ACCNO
MANAGES
BRANCH
IFSC_CODE
OPENING_DT
LOC
MGR_NO
HAS
DEPT_NO
IFSC_CODE
HOD
DEPARTMENT
DEPT_NO
DEPT_NAME
CHEQUE_BOOK
START_NUM
ACCNO
END_NUM
CHEQUE_BOOK
ISSUE_DT
LOAN_TYPE
L_TYPE_ID
LOAN_TYPE
INTEREST_RT
BELONGS TO
ACCOUNT_PENALTY
ACCNO
CURR_PENALTY_AMT
DUE_DT
CHARGES
LOAN
L_ID
ACCNO
LOAN_AMT
ISSUE_DT
EMI
G_ID
L_TYPE_ID
DURATION
BORROWS
GUARANTOR
ACCNO
OCCUPATION
G_NAME
ADDR
G_ID
DT_BIRTH
PH_NO
E_MAIL
ADHAAR_NO
PAN_NO
ANNUAL_INCOME
GUARANTEES
MAKES
HAS
BELONGS TO
PF
ACCNO
SAL
SAL_PERCENT
PF_BAL
DUE_DT
OPENING_DT
HAS
FD
FD_ID
ACCNO
INT_AMT
CURR_AMT
INTEREST
DEPOSITION_DT
PREV_MT_DT
MT_INTERVAL
PENALTY
DUE_DT
MANAGER
HEADS
WORKS IN
EMPLOYEE
EMP_NAME
EMPNO
DT_BIRTH
ADDR
E-MAIL
SAL
PH_NUM
JOB
COMM
JOIN_DT
IFSC_CODE
DEPT_NO
TRANSACTION
SENDER_ACCNO
RECIEVER_ACCNO
TRANS_ID
TRANS_DT
AMT
R_NEW_BAL
S_NEW_BAL
TRANS_TYPE
RECIEVES
ORDER_CHEQUE_VERIFICATION
CHEQUE_NO
TRANS_ID
EMPNO
RECIEVER_ADHAAR_NO
IS_VERIFIED
PROCEEDS
BELONGS TO
PAYMENT_METHOD
IS A
CREDIT_PAYMENT
TRANS_ID
SENDER_CC_NO
UPI_PAYMENT
TRANS_ID
SENDER_UPI
RECIEVER_UPI
SELF_PAYMENT
TRANS_ID
SELF_PAYMENT_TYPE
DEBIT_CARD
CVV_NO
DB_NO
ACCNO
ISSUE_DT
EXP_DT
PIN
DTYPE_ID
SET_LIMIT
UPI
UPI_ID
UPI_PIN
ACCNO
ACCOUNT_TYPE
IS OF
DEBIT_TYPE
DTYPE_ID
TYPE
COMP_NAME
LOC_INT
LOC_LIM
INT_LIM
ANNUAL_CHARGE
CHARGE_ON_TRANS
CREDIT_CARD
ACCNO
CREDIT_SC
LIMIT
CC_NO
CVV_NO
ISSUE_DT
EXP_DT
PIN
EXPENSE
C_TYPE_ID
IS OF
CREDIT_TYPE
C_TYPE_ID
COMP_NAME
TYPE
LOC_INT
LOC_LIM
INT_LIM
DUE_LIM
PAYMENT_INTR
HAS
CREDIT_DUE
CC_NO
IS_BLOCKED
FINE_AMT
DUE_DT
DUE_AMT
IS A
SAVINGS
REGULAR_INTEREST
SENIOR_CITIZEN_INTEREST
MIN_BAL
MAX_NUM_OF_TRANSACTIONS
TRANSACTION_LIMIT
MAINTENANCE_PRICE
PENALTY
CHEQUEBOOK_PRICE
IS A
CURRENT
PENALTY
MIN_BAL
CHEQUEBOOK_PRICE
MAINTENANCE_PRICE
SALARY
MAINTENANCE_PRICE
CHEQUEBOOK_PRICE

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);

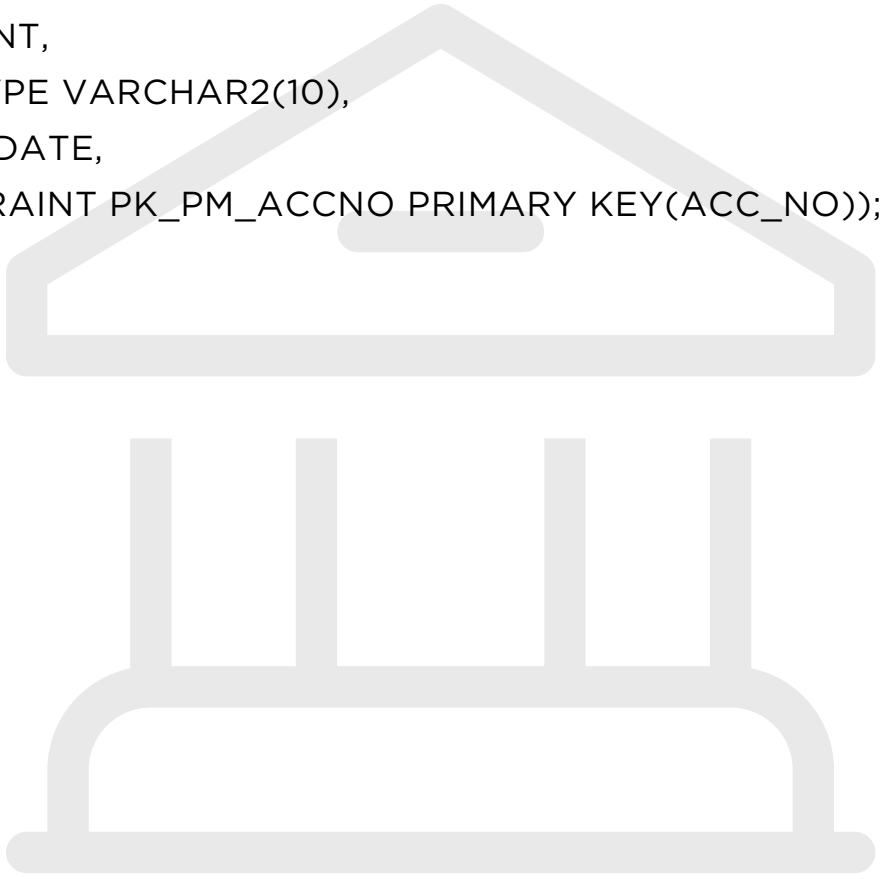
- CREATE TABLE LOC_OPR (
  O_ID NUMBER(11) CONSTRAINT FK_OID_LOCOPR REFERENCES OPERATOR(O_ID),
  IFSC_CODE VARCHAR2(8),
  LOCKER_NO NUMBER(3),
  CONSTRAINT FK_IFSC_LNO_LOCOPR FOREIGN KEY(IFSC_CODE, LOCKER_NO) REFERENCES LOCKER(IFSC_CODE, LOCKER_NO),
  CONSTRAINT PK_OID_IFSC_LNO_LOCOPR PRIMARY KEY(O_ID, 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_TRAND_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 CREDITDUEDATE(), 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 CREDITDUEDATE(); 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 SUCCESSFULL OR UNSUCCESSFULL INSERT QUERY ON TRANSACTION

```
CREATE OR REPLACE PROCEDURE PERFORM_TRANSACTION
   IS
   RES NUMBER(1);
TYPE1 TRANSACTION.TYPE%TYPE;
   BEGIN

   DBMS_OUTPUT.PUT_LINE('CHOSE 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_PROC;
/

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,NULL,NULL,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;

 INSERT INTO TRANSACTION VALUES
 ( TRANS_ID, SENDER_ACCNO,R1.RECEIVER_ACCNO, SYSDATE, R1.AMT,
 'OC', NULL,NULL, 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;

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

3)GENERATES TRANSACTION ID IN PATTERN : DATE ON WHICH TRANSACTION PERFORMED(DDMMYYYY) + NO OF SUCCESSFULL TRANSACTION ON THAT DAY.

```
CREATE OR REPLACE FUNCTION GET_TR_ID
   RETURN NUMBER
IS
TEMPTID VARCHAR2(38);
NOTD NUMBER(30);
TODAY_DATE USEFULL.DATE1%TYPE;

BEGIN

SELECT DATE1 INTO TODAY_DATE FROM USEFULL WHERE DID=1;
IF(TO_DATE(SYSDATE,'DD-MM-YYYY')-TODAY_DATE>0) THEN
   UPDATE USEFULL SET DATE1=TO_DATE(SYSDATE,'DD-MM-YYYY')  WHERE DID=1;
   UPDATE USEFULL SET RELATT=0  WHERE DID=1;
   TODAY_DATE:=TO_DATE(SYSDATE,'DD-MM-YYYY');
END IF;

SELECT RELATT INTO NOTD FROM USEFULL WHERE DID=1;

IF(NOTD<99999999999999999999999999999) THEN
   NOTD:=NOTD+1;
   TEMPTID:=TO_CHAR(TODAY_DATE,'DDMMYYYY')||NOTD;
END IF;
DBMS_OUTPUT.PUT_LINE(TEMPTID);
RETURN TO_NUMBER(TEMPTID,'99999999999999999999999999999999999999');
END;
/
```

4)FUNCTION TO CALCULATE THE PENALTY FOR THE OVERLIMIT EXPENSES OF A CREDIT CARD :

```
CREATE OR REPLACE FUNCTION GENERATEDUEAMOUNTPENALTY(
ABOVELIMIT IN NUMBER
)
RETURN NUMBER
IS
 OVERLIMITINTEREST CONSTANT NUMBER(2, 1) := 2.5;
 MINIMUM CONSTANT NUMBER(3) := 600;
 FINE NUMBER(20, 2);
BEGIN
 FINE := (ABOVELIMIT * OVERLIMITINTEREST) / 100;
```

```
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.BALALNCE < 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.BALALNCE < 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,ER
R);
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 NECCESSARY 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;
     UPDATECREDITONTRANSACTION(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