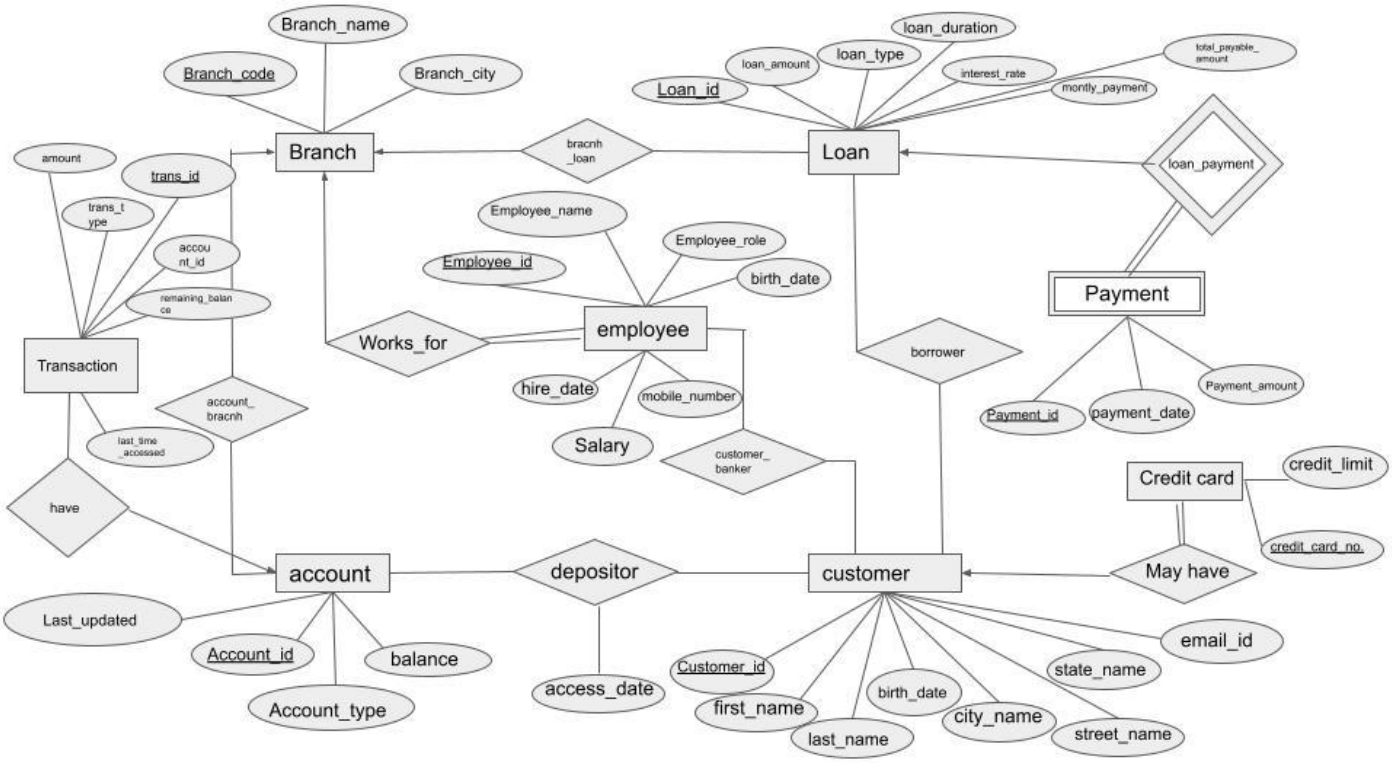
CS3120 Database Management Systems Laboratory

Banking System Database

Group number - 14

Group members:
111901032 Mohammed Abdul Azeem
111901052 Vikas Naik Dharavath
111901015 Bandi Kaushik

ER DIAGRAM :



Functionality:

The bank is organised into various branches. Each **branch** is located in a particular city and has a unique name. Each branch has a particular branch code.
The **customers** of the bank are uniquely identified by their *customer_id* values. The bank has information about each **customer's name, street** and **city** where the customer lives. Customers may have **accounts** and they may take **loans**. A customer may be associated with a particular bank **employee**.
Employees of the bank are identified by employee id values. The bank has information like the **name of the employee, telephone number, names of the employees 's dependent, role of the employee in the bank**. Each employee works for a particular bank and each branch may

have many employees. The bank also keeps track of the *employee's hire_date* and we can get employment *experience* for the employee.
Bank offers accounts to the customers. Accounts can be held by more than one customer and one customer can have more than one account. Each account has a unique *account_number* and the bank maintains information about the record of the account's balance and the recent date on which the account was accessed by the customer holding the account.
A loan originates at a particular branch and can be held by one or more customers. A loan has a unique loan number. For each loan, the bank has information about the *loan_amount* and the loan payments. Although a loan payment does not uniquely identify a particular payment among those for all the bank's loans, a payment number will identify a particular payment for a specific loan. The date and amount are noted for each payment. A customer may have a **credit card** offered by the bank.A customer can do transactions,withdrawals,deposits and take loans.

Cardinality Constraints:

A customer can have multiple accounts and an account can be held by multiple customers. Cardinality is M:N.
Customers are allowed to take loans from the bank and a loan can be held by one or more customers. Cardinality is M:N.
Each loan can be paid back with multiple payments having payment numbers. Cardinality is 1:N.
A customer can be an employee of a bank. Cardinality is M:N.
Branch can have multiple accounts. Cardinality is 1:N.
A branch of the bank will give loans to the customers. Cardinality is 1:N.
A customer may have multiple credit cards. Cardinality is 1:N.

The entities and attributes for the entities are as follows:

ENTITY	ATTRIBUTES
Branch	Branch_code, Branch_name,Branch_city
Customer	Customer_id, first_name, last_name, birth_date, city_name, street_name, state_name, email_id
loan	Loan_id, loan_amount, loan_type, loan_duration_months, interest_rate, montly_payment, total_payable_amount, date_of_creation
Employee	Employee_id, Employee_name, mobile_number, Employee_role, birth_date, hire_date, Salary
Payment	Payment_id, payment_date, Payment_amount
Account	Account_id, balance, Account_type, Last_update

Credit card	credit_card_no.,credit_limit
Transaction	Trans_id, amount, trans_type, remaining_balance

ER DIAGRAM TO RELATIONAL MODEL:

Entity Tables:

BRANCH

Branch_code	Branch_name	Branch_city

CUSTOMER

Customer_id	First_name	last_name	Birth_date	city_name	street_name	state_name	email_id

LOAN

Loan_id	Loan_amount	Loan_type	Loan_duration	Interest_rate	Monthly_payment	Branch_code	total_payable_account	Date_of_creation

EMPLOYEE

Employee_id	Employee_name	Mobile_number	Employee_role	Birth_date	Hire_date	Salary	Branch_code

PAYMENT

Payment_id	Payment_date	Payment_amount	Loan_id

ACCOUNT

Account_id	Balance	Account_type	Last_update	Branch_code

CREDIT_CARD

Credit_card_number	Credit_limit	cutomer_id

TRANSACTIONS

trans_id	account_id	amount	trans_type	remaining_balance

Relationship Tables:

DEPOSITOR

Customer_id	Account_id	Access_date

BORROWER

Customer_id	Loan_id

CUSTOMER_BANKER

Employee_id	Customer_id

Constraints:

- Branch:
  Branch_code - PRIMARY KEY
  Branch_name - UNIQUE
  Branch_city - NOT NULL

- Loan:
  Loan_id - PRIMARY KEY
  Loan_amount - CHECK(loan_amount > 0)
  Loan_type - NOT NULL
  Loan_duration_months - NOT NULL
  Interest_rate - NOT NULL
  Monthly_payment - NULL
  Branch_code - FOREIGN KEY
  Total_payable_amount- NULL
  Date_of_creation - NULL

- Customer;
  Customer_id - PRIMARY KEY
  First_name - NOT NULL
  Last_name - NOT NULL
  Birth_date - NOT NULL
  Street_name - NOT NULL
  City_name - NOT NULL
  State_name - NOT NULL
  Email_id - NOT NULL

- Employee:
  Employee_id - PRIMARY KEY
  Employee_name - UNIQUE
  Employee_role - NOT NULL
  Salary - CHECK(salary > 0)
  Telephone_number - UNIQUE
  Birth_date - NOT NULL

Hire_date - NOT NULL
Branch_code - FOREIGN KEY

- Payment:
  Payment_id - PRIMARY KEY
  Payment_amount - NOT NULL
  Payment_date - NOT NULL
  Loan_id - FOREIGN KEY

- Account:
  Account_id - PRIMARY KEY
  Account_type - NOT NULL
  Balance - NULL
  Last_updated - NOT NULL
  Branch_code - FOREIGN KEY

- Transactions:
  Trans_id - PRIMARY KEY
  account_id - FOREIGN KEY
  amount - CHECK(amount>0),
  trans_type - NOT NULL CHECK(trans_type= 'Deposit' or trans_type='Withdrawal' OR trans_type='Amt_sent' OR trans_type='Amt_received')
  remaining_balance - NOT NULL
  date_time_trans - NOT NULL

- Credit_card:
  Credit_card_number - PRIMARY KEY
  Credit_limit - NOT NULL
  Customer_id - FOREIGN KEY

- Depositor:
  Customer_id - NOT NULL, FOREIGN KEY ,PRIMARY KEY
  Account_id - PRIMARY KEY, FOREIGN KEY, PRIMARY KEY
  Access_date - NOT NULL

- Borrower:
  Customer_id - PRIMARY KEY, FOREIGN KEY, NOT NULL
  Loan_id - FOREIGN KEY, PRIMARY KEY, NOT NULL

- Customer_banker:
  Employee_id - PRIMARY KEY, FOREIGN KEY, NOT NULL
  Customer_id - FOREIGN KEY , NOT NULL, PRIMARY KEY

Procedures:

List of all procedures with brief explanation :

->deposit() : Procedure to deposit amount. (It takes account_id and amount to deposit as argument)
It takes the amount to be deposited and *account_id* and initially it updates the balance column in the account table and then lists the change in the transaction table and inserts a new record in the transaction table and finally updates the *access_date* to current_date in the depositor table.

->withdrawal() : Procedure to withdraw amount. (It takes account_id and amount to withdraw as argument)
It takes amount and *account_id* and initially it checks if the withdrawal amount is greater than the balance and if it's greater than the account balance, it raises a notice 'Insufficient balance in the account' else it updates the balance column in account table then inserts the new record in transaction table and updates the access_date to current_date in depositor table.

->transfer() : Procedure for amount transfer between two accounts(It takes the sender's account_id and receiver's account_id and amount_to_transfer as argument).
It takes the account_id of sender,receiver and the amount to be transferred then if the sender balance is less than the transfer amount it raises a notice saying 'Insufficient balance in sender's account' else it updates the balance column for sender's account table and then it insert new record in transactions table to reflect this transaction and then it updates the balance column of receiver and adds new tuple in transactions table and update the access_date for sender's details in depositor table.

->create_new_account : Procedure for creating an account in the bank for a customer who is opening an account for the first time in the bank.
It takes the person's first name, last name, D.O.B,street name,city name,email address and account_type. It inserts the values into the customer table then in the account table(this also includes creation of random account_id) .As the bank's condition is to keep a minimum balance of 5000 for Checking account and 3000 for Savings account there is a if statement which states, If the account_type is 'Zero Balance Account' then the changes won't be reflected in the transaction table as we don't deposit money in a Zero Balance Account. Else if account_type other than 'Zero Balance Account' is given then we insert this change in the transaction table where we generate a transaction_id and update the remaining_balance field and amount_field with the minimum money required during the account creation with the help of the trigger named cal_bal().

->create_another_account : Procedure for creating another account for the existing customer who already has an account.
In account type we have only 3 types of account and customers can have only one account in either of the types so maximum number of accounts a customer can have is 3. This function takes customer_id and account_type as input. Then it checks whether the given account type exists for the given account_id with if statement, if there exists any account with the account type mentioned it will give an error if not it will insert the data into the account table(this includes creation of random account_id). As the bank's condition is to keep a minimum balance of 5000 for Checking account and 3000 for Savings account there is a if statement which states, If the account_type is 'Zero Balance Account' then the changes won't be reflected in the transaction table as we don't

deposit money in a Zero Balance Account. Else if account_type other than ‘Zero Balance Account’ is given then we insert this change in the transaction table where we generate a transaction_id and update the remaining_balance field and amount_field with the minimum money required during the account creation with the help of the trigger named cal_bal().

->create_loan : Procedure for creating loan.
It takes customer_id,loan amount,loan_type,loan_duration and interest_rate as arguments and then it inserts the values into the loan table (here loan_id is generated randomly) and inserts values into the borrower table to maintain consistency of the data. The cal_emi() trigger is used to calculate loan data and to insert the data in the relevant tables.

Functions:

List of all the functions:

->find_balance() : This function is used to find the balance of the customer in his account with customer_id as input and gives the balance and account_type as output. If the customer has multiple accounts then it will give accounts types and its balances.

->loan_paid_to_date() : This function is used to find the percentage of loan payment completed till the last payment date. Here the input is customer_id and output is percentage of loan payment completed till last payment date.

->account_count() : This function is used to count the total number of accounts a customer has by taking customer_id as argument. We used group by and having clauses to achieve this function.

->account_to_cc() : This function is used to give the details like credit card number and credit limit of a customer with the given account_id if it exists. Here the input is account_id and output is credit card number and credit limit for that. If there is no credit card number and credit limit for that account_id the output is NULL.

Triggers:

List of triggers used:

->cal_bal(): Trigger for calculating the balance while creating a new account.
When a customer is created or new customer details added into the database using the procedure “create_new_account” or “create_another_account” in the process of insertion of data into the account table, this trigger gets activated and calculates the minimum balance according to the account type mentioned and inserts the calculated balance in the balance column of account table. Then, the calculated amount is inserted in the amount and remaing_balance field of the transaction table.

->emi_cal() : Trigger for calculating monthly payment for a loan.
This Trigger is used to calculate the monthly payment for a loan if a customer has taken a loan.

We are using the cal_emi() function for this trigger. Whenever a new record enters the loan table with the help of create_loan procedure, we insert the loan_id and the other relevant data in their corresponding field in the loan table. Then this cal_emi() trigger will calculate the monthly payment of the role with the help of the formula **EMI = P × r × (1 + r)n/((1 + r)n - 1)** where '**P**' is the principal loan amount which is given as input and '**r**' is interest rate per month which is calculated as interest_rate(input)/(12*100). This EMI value is then inserted into the monthly_payment column of the loan table. In this way when a new loan is created, we calculate the monthly_payment automatically using cal_emi trigger. The necessary data is inserted in the borrower table also to maintain data consistency.

->total_amt_cal(): Trigger used for calculating Total_payable_amount.
This trigger is used to calculate the total payable amount at the end of loan tenure. We are using the function cal_total_amt() for this trigger. Whenever a new record enters the loan table with the help of create_loan procedure, we calculate the monthly_payment using the emi_cal() trigger and then use that monthly_payment to calculate the *Total_payable_amount* using the formula *monthly_payment * loan_duration_month*. This calculated value is then inserted into the *total_payable_amount* field of the loan table. The relevant data is also inserted into the borrowers table to maintain data consistency.

Roles:

In this database I have created a Schema called banking and revoked the public and revoked all on this database from the public so that permissions and privileges can be applied to the roles. There are mainly three types of roles in this database:

1) Manager: This role has all access to the database created also the superuser. So this role has access and privileges to SELECT, INSERT, UPDATE and DELETE commands in the database on all tables.
2) Officer: As this role is an employee in a bank the main job is to manipulate the data. So this role has access and privileges to SELECT, INSERT, UPDATE and DELETE commands created in the database on the tables account, borrower, credit_card, customer, depositor, loan, payment, transactions. They can also do usage on the sequences.
3) Customer: This role only can use the SELECT command on the view customer_info which can show only basic info about the customer. And customers can only read the data from the customer_info.

Indices:

For the database the basic indexes used are

- inx_customer_c_id -> index on table customer for column customer_id
- inx_account_a_id -> index on table account for column account_id
- inx_account_balance -> index on table account for column balance
- inx_credit_card_c_id -> index on table credit card for column credit_card_number
- inx_employee_e_id -> index on table employee for column employee_id
- inx_employee_salary -> index on table employee for column salary
- inx_loan_amount -> index on table loan for column amount
- inx_loan_l_id -> index on table loan for column loan_id

For the database the additional indexes used are

- c_inx_employee -> index on table employee for column salary, employee_role
    - Query:
        - SELECT * FROM employee WHERE employee_role='Manager' AND salary>50000;
        - SELECT * FROM employee WHERE employee_role='Officer' AND salary<50000;
- c_inx_loan -> index on table loan for column loan_amount, loan_type
    - Query:
        - SELECT * FROM loan WHERE loan_amount>500000 AND loan_type='Equipment loan';
        - SELECT * FROM loan WHERE loan_amount<1000000 AND loan_type='Home loan';
- multi_inx_loan_b -> index on table loan for column loan_amount and branch_code
    - Query:
        - SELECT * FROM loan WHERE branch_code='HYD0001' AND loan_amount>500000;
        - SELECT * FROM loan WHERE branch_code='WAR0025' AND loan_amount>1000000;

Views:

-> customer_info() :

View is created with the columns first_name and last_name from table customer as full_name, birth date from customer table, account_type from account table and street name, city name, state from customer table.

Here the view is created to get the basic customer information separately so that we can access select only option on this view to the role customer. This will give access for the users under customer role to read their basic details.

-> customer_loan() :

View is created with columns first_name and last_name from table customer as full_name, loan_id and loan_amount from the table loan and percentage of loan completed is from the function loan_paid_to_date().

Here the view is created to get the information of the customers who took the loan and also the to know about the status of the loan i.e. to know about the percentage of of loan completed by the customer.