DBMS PROJECT
BANKING SYSTEM

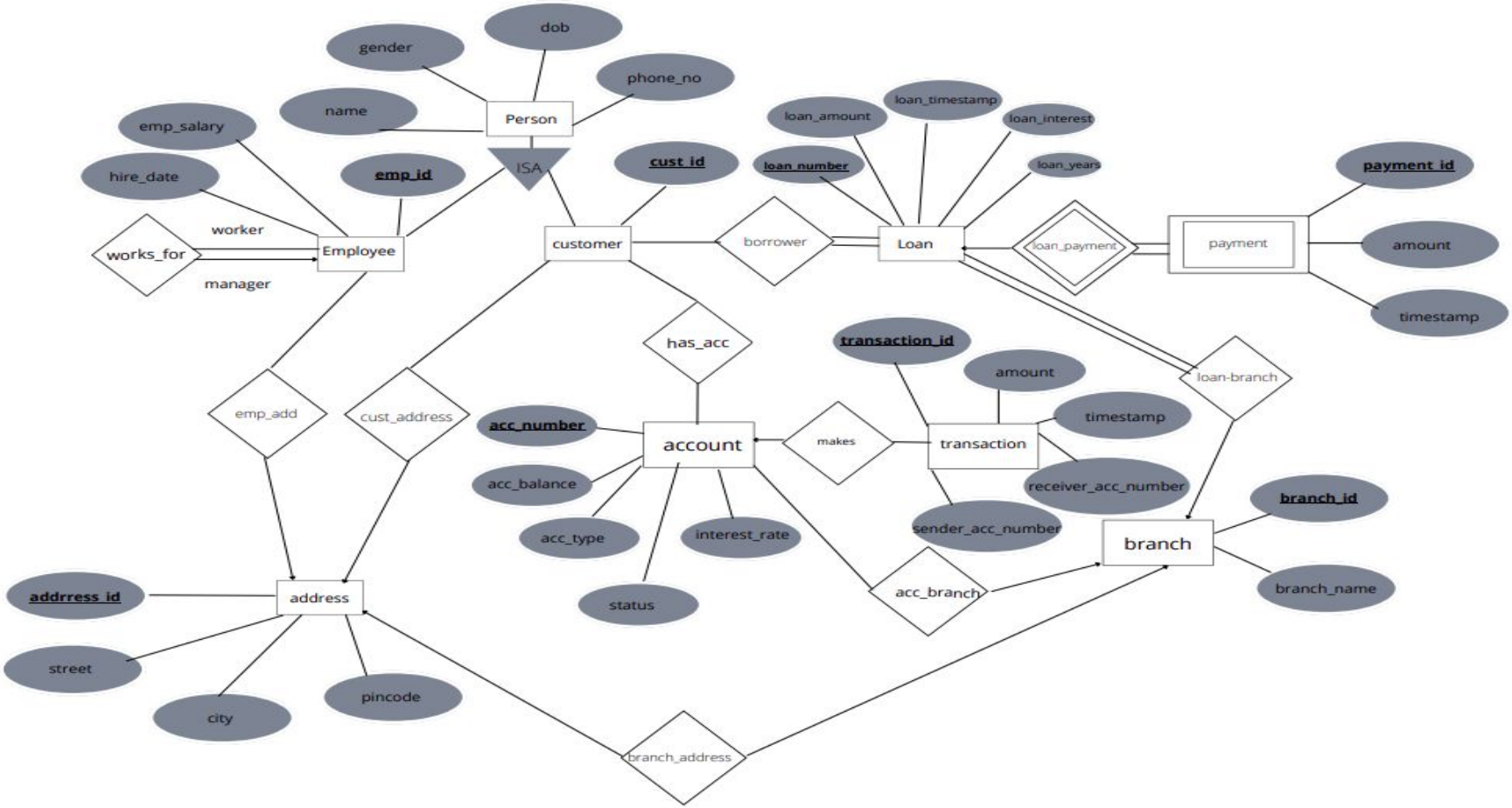
A report by : Group number 4
111901028 Kundan Pal
111901004 Aditya Agarwal
111901045 Rupesh Kumar

Description of Banking System

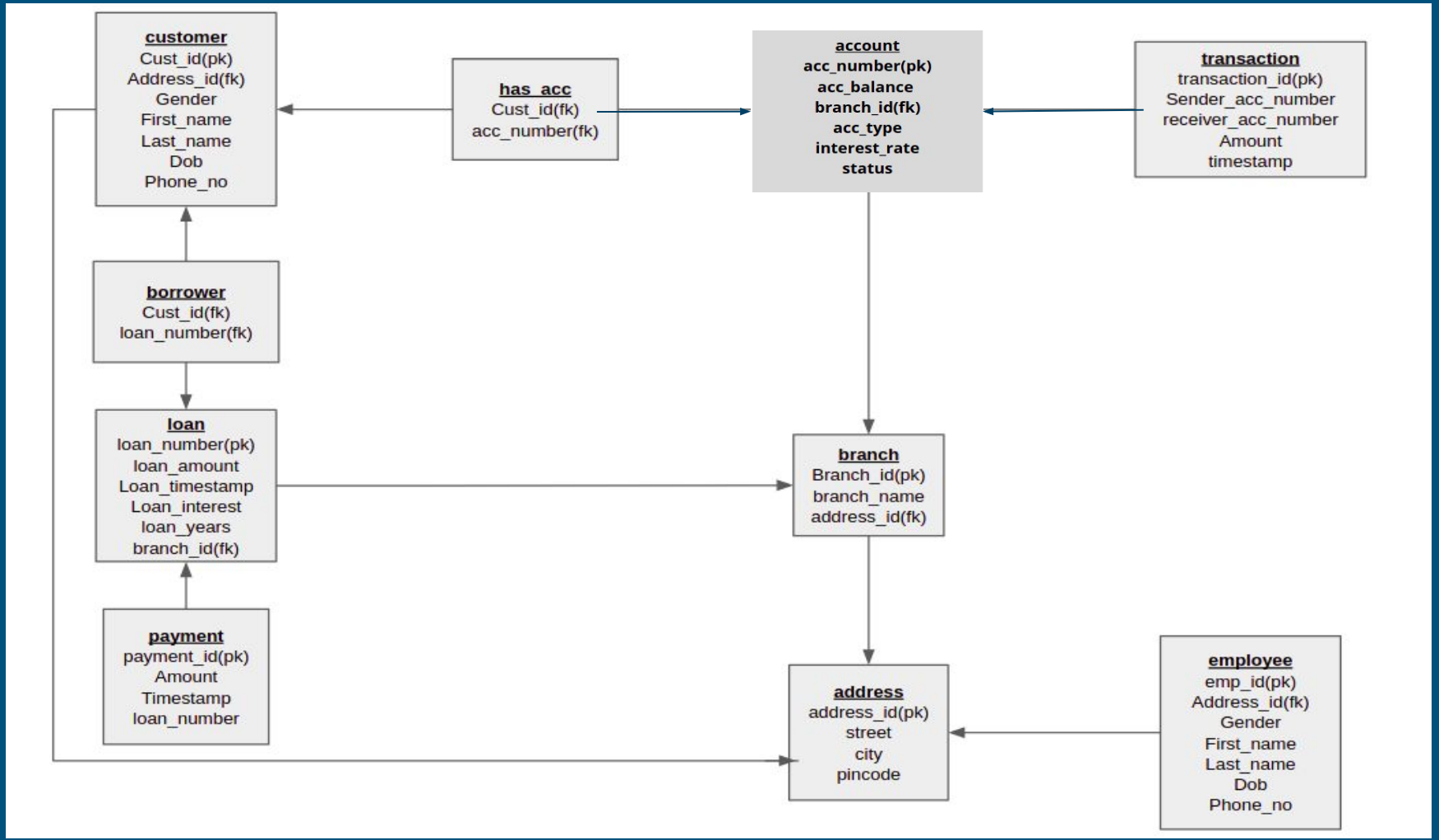
1. The bank is divided into branches. Each branch is located in particular city and is identified by unique id and their branch name.
2. Bank customers are identified by their customer id values.The bank stores each customer's name, gender, dob thus age ,phone number, street and the city where the customer lives.Customer have accounts and can take loans.The status of the account is also maintained whether the account is active or inactive.Moreover all the information is still present even if the account is deactivated.
3. Bank employees are identified by their employee id values. The bank administration stores the phone number of each employee, name, dob, gender, street and the city where the employee lives,the employee id of employee's manager, hire date and thus the length of employment.
4. The bank offers two types of account - saving and current account.Account can be held by more than one customer and one customer can held more than one account.Each account is assigned a unique account number. The bank maintains a record of account balance and the most recent date on which any kind of transaction has occurred via the account. In addition each account has an interest rate on savings account.
5. A loan originates at particular branch and can be held by more than one customers. A loan is identified by a unique loan number for each loan the bank keep track of loan amount and the loan installments paid towards a particular loan.
6. All the information regarding the transactions of accounts is maintained which consists unique_id, transaction amount, timestamp, and the senders id and receivers id.

7. The information regarding withdrawal and deposit of money by the account holder is also stored.

ER Diagram


dob
gender
phone_no
name
Person
emp_salary
hire_date
emp_id
ISA
cust id
loan_amount
loan_timestamp
loan_interest
loan number
loan_years
payment_id
worker
works_for
Employee
customer
borrower
Loan
loan_payment
payment
amount
manager
timestamp
has_acc
transaction_id
amount
loan-branch
emp_add
cust_address
acc_number
account
makes
transaction
timestamp
acc_balance
receiver_acc_number
sender_acc_number
branch_id
acc_type
interest_rate
branch
addrress id
address
acc_branch
branch_name
status
street
city
pincode
branch_address

Schema Diagram


customer
Cust_id(pk)
Address_id(fk)
Gender
First_name
Last_name
Dob
Phone_no
has_acc
Cust_id(fk)
acc_number(fk)
account
acc_number(pk)
acc_balance
branch_id(fk)
acc_type
interest_rate
status
transaction
transaction_id(pk)
Sender_acc_number
receiver_acc_number
Amount
timestamp
borrower
Cust_id(fk)
loan_number(fk)
loan
loan_number(pk)
loan_amount
Loan_timestamp
Loan_interest
loan_years
branch_id(fk)
branch
Branch_id(pk)
branch_name
address_id(fk)
payment
payment_id(pk)
Amount
Timestamp
loan_number
address
address_id(pk)
street
city
pincode
employee
emp_id(pk)
Address_id(fk)
Gender
First_name
Last_name
Dob
Phone_no

Functionality

1. Opening account ⇒ We can open the account for the customer. The customer can open multiple accounts. We have implemented the function for opening account which is mentioned in the later slides.

2. Taking loan⇒ You can also take loan from this bank for specific time period with a specific interest rate. A customer can have multiple loans in different branches of the banks.

3. Transaction between two accounts⇒ You can also transfer the money from one account to the other account the details of which will be stored into the transaction table.

4. Enrollment of new employee⇒ in this we add the information of the new employee that comes into the bank. We ask the details of the employee and then add them to the database. There is a function dedicated to this.

Functionality

1. Manager⇒ The employee will be assigned a new manager will be stored in the employee table only.

2. Deposit/withdraw ⇒ The account holder can also deposit and withdraw the money and hence this transaction will be inserted in the transaction table.

3. Installments ⇒ If loan is being payment in installments then it will be added to the payments table with the details.

4. Updating details ⇒ Any update can be made like updating the personal details, name, address and so on. Then the changes and insertion is new entries will be made as per the requirement. There are functions and procedures are defined.

Explanation of Relations

1. Works-for(employee to employee)⇒ this is a many to one recursive relation between the employee to employee. This because manager is itself a employee. And every employee will have only one manager. Every manager will look after multiple employees hence this will be many to one relation.

2. Has_acc(customer to account) ⇒ this will be many to many relation as each customer can have more than one account and also one account can be opened by multiple customers combined which we also called as joint account.

3. Borrower(customer to loan) ⇒ this is again a many to many relation as a single customer can take multiple loans and particular loan can be taken by multiple customers if they have joint account. The participation loan side will be total participation as each loan will be associated with at least one customer.

4. Loan_payment(loan to payment) ⇒ This will be a one to many relation because many installments can be made with the single loan account and each payment/installment will be done by only one loan account hence this relation will be one to many relation and this is identifying relation as well.

5. loan_branch(loan to branch) ⇒ this is a many to one relation as multiple loans can be taken from a single branch and each loan will be taken from only single branch only hence this will be many to one relation with total participation from loan side.

6. Makes(account to transaction) ⇒ This is a one to many relation again as each account can do multiple transactions and one particular transaction can be done by only one account hence one to many.

Explanation of Relations

7. Cust_address(customer to address) ⇒ This is many to one relation as each customer will have only one address and multiple customer can belongs to same address.

8. Emp_address(employee to address) ⇒ This is many to one relation as each employee will have only one address and multiple employees can belongs to same address.

9. Branch_address (branch to address) ⇒ this is one to one function as each branch will have only one address and at one address there can be only one branch.

Tables(relations)

account

acc_number	acc_balance	interest_rate	acc_type	status	branch_id

address

addess_id	street	city	pincode

borrower

cust_id	loan_number

branch

branch_id	branch_name	address_id

customer

cust_id	gender	first_name	last_name	dob	phone_ numer	address_id

employee

emp_id	first_ name	last_ name	phone_n o	gender	dob	hire_ date	manager_ id	address_id

has_acc

cust_id	acc_id

loan

loan_number	loan_amount	branch_id	loan_year	loan_interest	loan_timestamp

payment

payment_id	amount	payment_timestamp	loan_number

transaction

transaction_id	transaction_ timestamp	senders_acc_ number	receivers_acc_ number	transaction_amount

Tables with constraints

1. **Branch ⇒ branch_id (not null), address_id, branch_name (not null).**

   (primary_key ⇒ branch_id)

   (foreign_key ⇒ address_id)

2. **Address ⇒ address_id (not null), street (not null), city (not null), pincode (not null).**

   (primary_key ⇒ address_id)

   The pincode is a 6 digit number.Hence this constraint is enforced on the attribute pincode.Thus pincode will be greater than 99999 and will be less than or equal to 999999.

3. **Loan ⇒ loan_number(not null), loan_amount(not null), loan_timestamp(not null), branch_id,loan_years(not null),loan_interest(not null)**

   (primary_key ⇒ loan_number)

   (foreign_key ⇒ branch_id)

   The loan_ amount will be greater than 0.

   The loan_interest is a percentage hence will range from 0 to 100.

   The loan_years are the number of years thus will be integer and greater than 0.

4. **Account⇒ acc_number(not null), branch_id, acc_balance(not null), acc_type(not null), interest_rate(not null),status(not null).**

   (primary_key ⇒ acc_number)

   (foreign_key ⇒ branch_id)

   acc_balance should be non-negative value.

   There are only two types of account or acc_type which are savings and current account,

   The interest_rate is a percentage value hence should be greater than or equal to 0 and less than or equal to 100.

   status to check if the account is active or inactive.It can take two values active or inactive.

5. **Customer ⇒ cust_id (not null), address_id (not null), gender (not null), first_name (not null), last_name (not null), dob (not null), phone_no (not null)**

   (primary_key ⇒ cust_id)

   (foreign_key ⇒ address_id)

   The age should be at least of 18 years thus the dob should be before current_date-18 years.

   The gender will be only male, female, other.

   The phone_no signifies phone number hence a 10 digit number so will range from 1000000000 to 9999999999.

6. **Employee** ⇒ **emp_id(not null), manager_id, address_id, hire date(not null), first name(not null), last name(not null), phone_no(not null), gender(not null), dob(not null)**

(primary_key ⇒ emp_id)

(foreign_key ⇒ address_id)

emp_id of a tuple will not be equal to manager_id as the person can't be manager of himself/herself.

Age of a person should be 18 years thus the accepted dob will be dob - 18 years.

gender can take only three values which are (male, female and other)

hire_date should be less than current date.

phone_no signifies phone number hence a 10 digit number so will range from 1000000000 to 9999999999.

7. **Payment(weak entity) ⇒ payment_id(not null), amount(not null), payment_timestamp(not null), loan_number (not null)**

(primary_key ⇒ payment_id)

(foreign_key ⇒ loan_number)

The installments towards loan payment are added thus the amount will be greater than 0.

Default for payment_timestamp is current_timestamp.

8. **Borrower** ⇒ cust_id (not null), loan_number (not null)

(primary_key ⇒ {cust_id, loan_number}

(foreign key ⇒ cust_id, loan_number)

9. **Has_acc** ⇒ cust_id (not null), acc_number (not null)

(primary _key ⇒ {cust_id, acc_number}

(foreign key ⇒ cust_id, acc_number)

10. **Transaction** ⇒ transaction_id (not null), sender_acc_number, receiver_acc_number, transaction_timestamp (not null), transaction_amount (not null)

(primary_key ⇒ transaction_id)

(foreign_key ⇒ sender_acc_number, receiver_acc_number)

transaction_amount should be greater than 0.

Procedures

1. **Transfer** : For account “A” to “B” transaction we are defining procedure “transfer”. This function takes sender’s account number, receiver‘s account number and amount as input. In this we first deduct given amount from sender’s account. If this query is successful then we add this amount to receiver’s account. After these queries are successfully executed we insert details of this transaction to transaction table. In transaction table we insert sender’s and receiver’s account number and given amount.

2. **Deposit :** This procedure handles the scenario when a customer tries to deposit money in his account. “Deposit” procedure takes account number of the customer and the amount of deposit. In procedure first we add amount to the given account number. Further we add details of this transaction to the transaction table. In transaction table we add receiver’s account number and the given amount. Here sender’s account number will be null as it is assumed we a customer is depositing money to his account.

3. **Withdraw :** This procedure handles the scenario when a customer tries to withdraw money from his account. “withdraw” procedure takes account number of the customer and the amount he want to withdraw. In procedure first we deduct given amount to the given account number. If this query is successful then we add details of this transaction to the transaction table. In transaction table we add sender’s account number and the given amount. Here receiver’s account number will be null as it is assumed we a customer is withdrawing money from his account only.

a.

4. **Loan_payment :** This is used for updating the installment of a loan.Here the input is the installment amount and the loan number.The loan_amount is decreased in the loan table and its noted in the payment table as a log for paying the loan installment.

5. **open_loan :** This procedure is used to give loan to the customer. This procedure will take loan_amount, loan_interest and loan_years as input.Moreover it could be of a single person or a joint loan.

6. **open_account** : This is used for opening an account in the bank.Here we take the details of the account holder.His/her details are reflected in the customer table.Then the entry is made in the has_acc table.Then his entry is made in the account table corresponding to the account number that the person gets.Moreover it could be of a single person or a joint account.

7. **delete_customer_account** : This procedure will close the account in which we have to make the status as inactive simply.The information is stored for future references.

Functions

1. get_customer_details:This function will display the customer details if given the customer id. This will print like first_name, last_name,address, phone_number and so on.

2. get_employee_details :This function will display the employee details by inputting the employee id.

3. get_account_details:This function will display the details of the customer giving account number. So this function will take account number as input.

4. get_loan_details : This will display the loan details taken by the customer if given the customer id.

5. Add_new_employee : this function will take the employee details of the new employee and insert into the table. And we will assign him a manager.

Triggers

1. calculate_amount_trigger : When the insertion is done only the principal amount is inserted in the loan table for loan_amount.This trigger inserts the total amount that is principal + simple interest before insertion that is calculated using the loan_years and loan_interest via the formula simple interest=(principal amount * interest * years)/100.

2. warn_max_min_amount : This trigger is used to check whether after the transaction a particular account is satisfying the minimum balance and maximum balance criteria or not. If it is not followed then it will show the message on the terminal giving warning. So this trigger is done after updation of account or account balance. We have assumed that the minimum balance should be 5000 and the maximum balance is 500000.

3. loan_completion :This trigger is used to print the message to terminal if total loan is amount is paid. This will be called on every updation of the loan_amont in the loan table and hence once the loan_amount reaches 0 then it will call this trigger and the message will be printed on the terminal.

Indexing

1. **Account**

i) acc_balance = we have implemented btree indexing on this because we can get lot of range based queries like as follows

Find the details accounts whose account balance is between 20000 to 50000.

Select * from account where acc_balance >= 20000 and acc_balance <= 50000

ii) acc_number = we have used the hash index for this attribute as we get the point queries for searching the account details given the account number like as follows

Find the account details whose account_number is 123456.

Select * from account where acc_number = '123456'

iii) branch_id = we have used the hash index on this because again we can get the queries which involve finding the account details in a particular branch_id. One example of such frequent query is as follows:

Find the details of all account which have opened in branch_id = 1

Select * from account where branch_id = 1;

Indexing

One query that would be very frequent will be the accounts that exceed the maximum limit of acc_balance and other the accounts that are below the minimum limit of acc_balance so to prompt them at regular intervals for that as it will cost additional charges.So we will create partial indexing on query below-

select * from account where the acc_balance < 5000 or acc_balance >500000;

2. Address

i)address_id = we have used the hash index because the most frequent queries will be the point queries like as follows
Find the details of particular address whose address_id = 2
Select * from address where address_id = 2;

ii)pincode = we have used the hash index because the most frequent queries will be the point queries like as follows
Find the address details having pincode = 821108
Select * from address where pincode = 821108;

iii) street, city = we have used the multicolumn indexing in the street and city. Here the entries are strings hence we have Implemented the gin indexing. It will be helpful in the following queries-
Find the address details with street = ‘abc’ and city = ‘def’
Select * from address where street = ‘abc’ and city = ‘def’;

Indexing

3. borrower

i)cust_id = in this we have implemented the hash indexing as the queries will be the point queries. For example
Find the loan_numbers of the customer whose cust_id = 2
Select loan_number from borrower where cust_id = 2

4. branch

i)address_id = we have used the hash index on this because again we can have point queries like as follows
Find the address of the branch with address _id = 1
Select * from address from address, branch where branch_id = 1 and branch_id.address_id = address.address_id
ii)branch_id = used hash index with the same logic.

5. customer

i)cust_id = used the hash index as it will get the point queries frequently like
Find the details of the customer with cust_id = 1
Select * from customer where cust_id = 1;

6. employee

i)emp_id = we have used the has because it can have the point query like as follows
Find the employee details with emp_id = 1
Select * from employee where emp_id = 1;

ii)emp_salary = we have used the btree indexing because the query will be the range based query like as follows
Find the details of the employee whose salary is greater than 20000 and less than 50000.
Select * from employee where emp_salary>20000 and emp_salary < 50000

Indexing

iii)hire_date = we have used btree again because it will have range based queries more frequently like mentioned below.
Find the details of the employee who were hired between '2000-01-01' and '2002-01-01';
iv)Manager_id = we have used the hash index for this case because we can get the point query as follows
Find the details of the employee who have manager as manager_id = 1
select * from employee where manager_id = 1

7. Has_acc

i)Cust_id = we have used the hash index as the point based query will be more frequent as follows
Find the details of the account of a customer with cust_id = 1;
Select * from has_acc, account where cust_id = 1 and has_acc.acc_number = account.acc_number

8. Loan

i)branch_id = we have used the hash_index as the most frequent queries will be the point based queries for branch_id . example is :
Find the details of the loan taken from branch_id = 1
Select * from loan where branch_id = 1
ii)loan_amount = we have used the btree indexing because the most frequent queries can be range based queries like as follows
Find the details of the loan with having active amount greater 10000
select * from loan where loan_amount > 10000
iii)loan_number = we have used the hash index because the most frequent queries will be point based as shown below
Find the details of the loan with loan_number = 123
Select * from loan where loan_number = 123

Indexing

9. payment

i)loan_number = we have implemented the indexing on the loan_number because the most frequent query will be to find the transactions of the loan_number from payment table then it is nothing but a point query as mentioned below

Find details of the payment transactions of a loan_number = 123

select * from payment where loan_number = 123;

10. transaction

i)senders_acc_number , receivers_acc_number = we have implemented the multicolumn indexing using the btree because we can get the following queries

Find the details of the transactions which has been done from sender's account to the receiver's account.

Select * from transactions where senders_acc_number = '1' and receivers_acc_number = '2'

Views

We have implemented the views for each of the role as each role will have different views according to their work.

1. Cashier's View:
   According to the job of cashier that is take care of accounts.Hence cashier's view will include the account table.Moreover necessary privileges are granted like select and update on the account table.

2. Accountant's View:
   According to the job of accountant that is take care of account of the customers and act as an intermediate between the bank and the customer.Hence accountant will have to the view that include the customer and account table with certain privileges.Moreover the accountant will also be able to show the transactions involving a particular account.Hence will have select privileges on a view comprising of transaction table.

3. Recruitment Manager's View:
   According to the job of Recruitment manager that is take care of all the recruitments in the bank.Thus will have view of employee table. Moreover all the privileges will be granted on the employee table to recruitment manager.

Roles and Authorization

1. Db_administrator : The superuser of the database. He will have access to the whole database.
2. Cashier : Cashier will have to perform account “A” to ”B” transaction, money deposit and money withdrawal. So cashier will have permission to select, update data from an account.
3. Accountant : accountant in the bank keep track of transactions of customer’s account. When new customer want to open/close an account he/she will come to the accountant or update details regarding the customer. So accountant will have select, insert, update on the customer and the account table. Accountant will also have permission to select on transaction table so he/she can print the transaction details of a customer’s account.
4. Recruitment Manager:Recruitment Manager will take care of the recruitment in the bank.When new employee comes his/her details etc are added or updation in details of an employee.