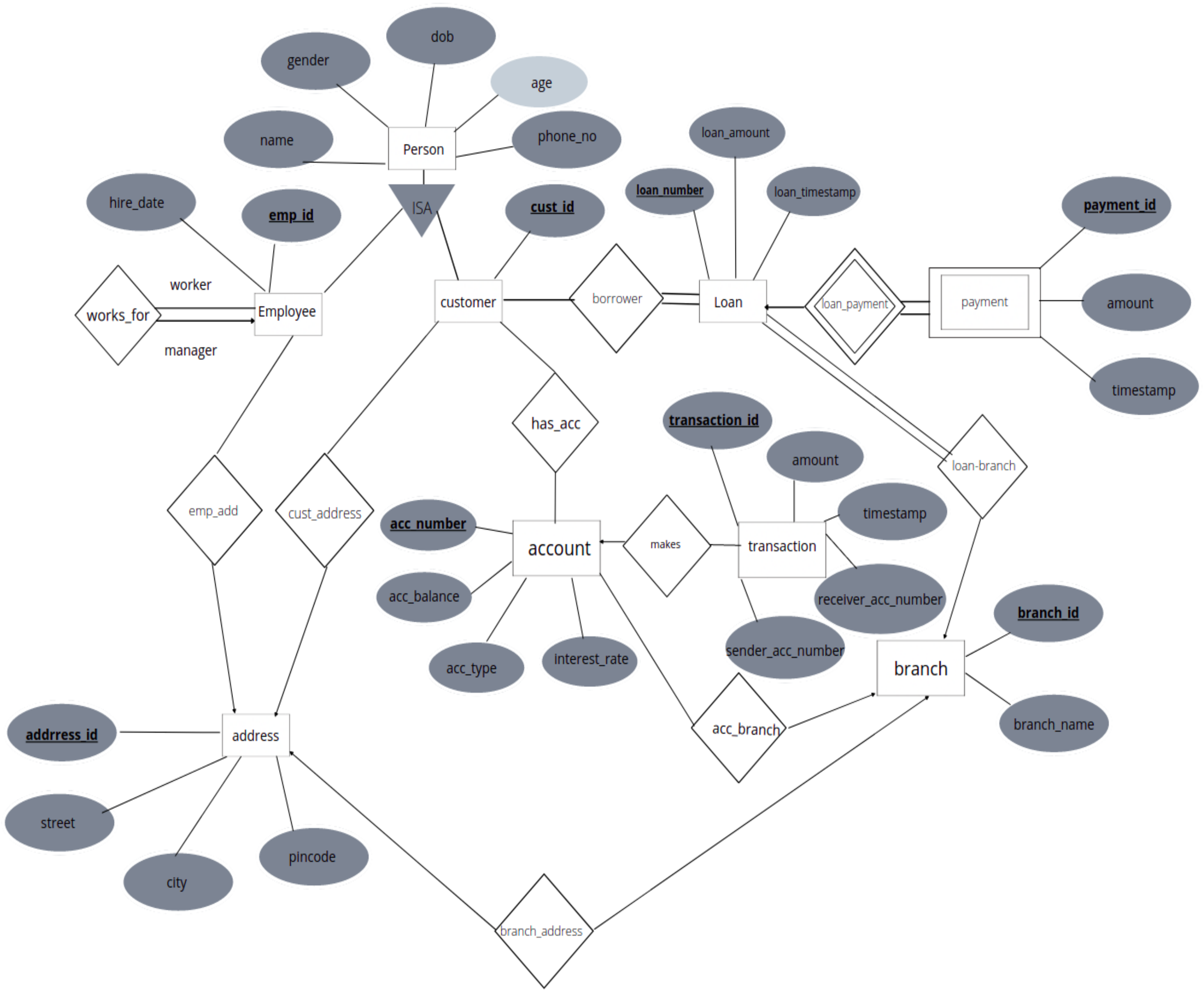
DBMS

ER diagram :



ENTITIES:

1. Person ⇒ name, gender, dob, age(derived attribute), phone_no
2. Customer ⇒ cust_id
3. Employee ⇒ emp_id, start_date
4. Address ⇒ address_id, street, city, pincode
5. Loan ⇒ loan_number, loan_amount, loan_timestamp(when the loan was taken).
6. Payment ⇒ payment_id,timestamp(when the payment is done), amount.
7. Branch => branch_id, branch_name.
8. Account ⇒ acc_number, acc_balance, acc_type, interest_rate
9. Transaction ⇒ transaction_id, sender_acc_number, receiver_acc_number, amount, timestamp.

Relations:

1. branch_Account = branch(1) — account(N). A branch can have multiple accounts.
2. Account ISA = saving, current.
3. Loan-branch = branch(1) — loan(N)
4. Depositor = customer(m) — account n
5. Borrower = customer(m) — loan N
6. Loan payment = loan(1) — payment(N)(weak entity) identifying relation.
7. Cust_banker = customer(M) — employee(N)
8. works_for(recursive relation) = emp — worker — manager — emp

Tables

1. **customer(customer+cust_address) = <u>cust_id</u>,** address_id, **gender, first_name, last_name, dob, phone_no**

   Primary_key ⇒ cust_id, foreign_kay = address_id

2. **Borrower ⇒ <u>cust_id, loan_number</u>(primary_key ⇒ {cust_id, loan_number}, foreign key ⇒ cust_id, loan_number)**
3. **Has_acc ⇒ cust_id, acc_number (both)**
4. **Account (account + acc_branch) ⇒ acc_number, branch_id, acc_balance, acc_type, interest_rate. (primary_key ⇒ acc_number, foreign_key ⇒ branch_id)**
5. **Transaction (transaction+makes) ⇒ transaction_id, sender_acc_number, receiver_acc_number, timestamp, amount. (acc_number == sender_acc_number)(primary_key ⇒ transaction_id, foreign_key ⇒ sender_acc_number, receiver_acc_number)**
6. **branch(branch+branch_address) ⇒ branch_id, address_id, branch_name,(primary_key = branch_id, foreign_key ⇒ address_id)**
7. **Address ⇒ address_id, street, city, pincode. (primary_key ⇒ address_id)**
8. **loan(loan + loan_branch) ⇒ loan_number, loan_amount, loan_timestamp, branch_id(primary_key ⇒ loan_number) (foreign_key ⇒ branch_id)**

9. **payment(payment + loan_payment) ⇒ payment_id, amount, timestamp, loan_number (primary_key ⇒ payment_id) and foreign_key(loan_number)**
10. **Employee (works for + emp_address+employee) ⇒ emp_id, manager_id, address_id, hire date, first name, last name, phone_no, gender, dob**

    primary_key(emp_id) and foreign_key (address_id)

1. customer:

cust_id	address_id	name	dob		gender	phone_no

2. Account :

acc_number	branch_id	acc_balance	acc_type	interest_rate

3. Transaction;

transaction_id	sender_acc_number	receiver_acc_number	timestamp

4. Borrower

cust_id	loan_number

5. Has_acc

cust_id	acc_number

6. Loans:

loan_number	loan_amount	loan_timestamp	branch_id

7. Address

address_id	street	city	pincode

8. Branch :

branch_id	branch_name	address_id

9. Employee:

emp _id	manager _id	address _id	hire_ date	first_name	last_name	phone _no	gender	dob	

10. Payment:

payment_id	amount	timestamp	loan_number

The commands we used

CREATE TABLE address (

address_id SERIAL PRIMARY KEY NOT NULL,

```
street VARCHAR(50) NOT NULL,
city VARCHAR(50) NOT NULL,
CONSTRAINT pincode_constraint CHECK(pincode > 99999 AND
pincode<=999999));

CREATE TABLE branch(
branch_id SERIAL PRIMARY KEY NOT NULL,
constraint f_key_address_id FOREIGN KEY (address_id) references
address(address_id))

ALTER SEQUENCE branch_branch_id_seq restart with 1000

CREATE TABLE loan(
loan_number serial primary key NOT NULL,
loan_amount NUMERIC NOT NULL CHECK(loan_amount>=0),
branch_id int,
CONSTRAINT f_key_branch_id foreign key (branch_id) references
branch(branch_id));

ALTER SEQUENCE loan_loan_number_seq restart with 123456

ALTER TABLE loan ADD COLUMN loan_timestamp timestamp DEFAULT
current_timestamp NOT NULL;
```

Kundan

```
CREATE TABLE has_acc
```

```
(cust_id INT, acc_number INT
, CONSTRAINT f_key_cust_id FOREIGN KEY (cust_id) REFERENCES customer(cust_id),
CONSTRAINT f_key_acc_number FOREIGN KEY (acc_number) REFERENCES account(acc_number)
);


CREATE TABLE borrower(cust_id INT, loan_number INT,
CONSTRAINT f_key_cust_id FOREIGN KEY (cust_id) REFERENCES customer(cust_id),
CONSTRAINT f_key_loan_number FOREIGN KEY (loan_number) REFERENCES loan(loan_number)
);


CREATE TABLE employee(
emp_id SERIAL NOT NULL PRIMARY KEY, first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL, phone_no NUMERIC CHECK((phone_no >= 1000000000) and
(phone_no <= 9999999999)) NOT NULL,
gender VARCHAR(10) CHECK(gender in ('Other', 'Male', 'Female')) NOT NULL, dob DATE CHECK(dob <
CURRENT_DATE) NOT NULL,
hire_date DATE CHECK(hire_date <= CURRENT_DATE) NOT NULL DEFAULT CURRENT_DATE,
manager_id INT, address_i


CREATE OR REPLACE PROCEDURE deposit(
receiver int,
amount NUMERIC
)
language plpgsql AS $$
BEGIN
IF NOT EXISTS(select acc_number from has_acc where receiver=acc_number)  then
Raise exception ‘Account number does not exists!’;
ELSIF amount<=0 then
RAISE EXCEPTION ‘Not a valid amount!’;
```

```
else
    UPDATE account
    SET acc_balance = acc_balance + amount
    WHERE
    acc_number = receiver;

    INSERT INTO transaction(receivers_acc_number, transaction_amount)
    VALUES (receiver, amount);
End if;
    COMMIT;
END; $$
;
```

Final deposit:

```
CREATE OR REPLACE PROCEDURE deposit(
receiver int,
amount NUMERIC
)
language plpgsql AS $$
declare stat VARCHAR;
BEGIN
    IF NOT EXISTS(select acc_number from has_acc where receiver=acc_number)
then
        Raise exception 'Account number does not exists!';
    ELSE IF amount<=0 then
        RAISE EXCEPTION 'Not a valid amount!';
    END if;

    select status into stat from account where acc_number = receiver;
    if (stat = 'Inactive') then
        Raise exception 'Entered account is not active anymore';
    else
```

```
        UPDATE account
        SET acc_balance = acc_balance + amount
        WHERE
        acc_number = receiver;

        INSERT INTO transaction(receivers_acc_number, transaction_amount)
        VALUES (receiver, amount);
    End if;

    COMMIT;
END; $$
;
```

```
CREATE OR REPLACE PROCEDURE withdraw(
account_no int,
amount NUMERIC
)
language plpgsql AS $$
DECLARE
            account_balance integer;
BEGIN
select acc_balance into account_balance  from account where acc_number = account_no;
IF NOT EXISTS(select acc_number from has_acc where account_no=acc_number)
then
Raise exception ‘Account number does not exists!’;
ELSIF amount<=0 then
RAISE EXCEPTION ‘Not a valid amount!’;
ELSIF account_balance < amount then
Raise exception ‘Not enough balance!’;
else
    UPDATE account
    SET acc_balance = acc_balance - amount
```

```
   WHERE
   acc_number = account_no;

   INSERT INTO transaction(senders_acc_number, transaction_amount)
   VALUES (account_no, amount);
End if;
   COMMIT;
END; $$
;
```

Final withdraw :

```
CREATE OR REPLACE PROCEDURE withdraw(
account_no int,
amount NUMERIC
)
language plpgsql AS $$
DECLARE
        account_balance integer;
        stat VARCHAR;
BEGIN
   select acc_balance into account_balance  from account where acc_number =
account_no;
   IF NOT EXISTS(select acc_number from has_acc where account_no=acc_number)
then
      Raise exception 'Account number does not exists!';
   ELSIF amount<=0 then
      RAISE EXCEPTION 'Not a valid amount!';
   END if;

   select status into stat from account where acc_number = account_no;
   if (stat = 'Inactive') then
      Raise exception 'Entered account is not active anymore';
   ELSIF account_balance < amount then
      Raise exception 'Not enough balance!';
   else
      UPDATE account
      SET acc_balance = acc_balance - amount
      WHERE
      acc_number = account_no;

      INSERT INTO transaction(senders_acc_number, transaction_amount)
      VALUES (account_no, amount);
   End if;
   COMMIT;
END; $$
;


CREATE OR REPLACE PROCEDURE transfer(
sender INT,
receiver INT,
amount NUMERIC
)
language plpgsql AS $$
Declare
         account_balance integer;
BEGIN
IF NOT EXISTS(select acc_number from has_acc where sender=acc_number)  then
Raise exception ‘Sender’s Account number does not exists!’;
Elsif  NOT EXISTS(select acc_number from has_acc where receiver=acc_number)
then
Raise exception ‘Receivers Account number does not exists!’;
Endif;

select acc_balance into account_balance  from account where acc_number = sender;
```

```
IF amount<=0 then
RAISE EXCEPTION ‘Not a valid amount!’;
ELSIF account_balance < amount then
Raise exception ‘Not enough balance!’;
else

   UPDATE account
   SET acc_balance = acc_balance - amount
   WHERE acc_number = sender;

   UPDATE account
   SET acc_balance = acc_balance + amount
   WHERE acc_number = receiver;

   INSERT INTO transaction (senders_acc_number, receivers_acc_number, transaction_amount)
   VALUES(sender, receiver, amount);

   COMMIT;
END;  $$
;
```

Final transfer :

```
CREATE OR REPLACE PROCEDURE transfer(
sender INT,
receiver INT,
amount NUMERIC
)
language plpgsql AS $$
Declare
        account_balance integer;
        stat VARCHAR;
BEGIN
   IF NOT EXISTS(select acc_number from has_acc where sender=acc_number)  then
```

```
      Raise exception 'Sender’s Account number does not exists!';
   Elsif  NOT EXISTS(select acc_number from has_acc where receiver=acc_number)
then
      Raise exception 'Receivers Account number does not exists!';
   End if;

   select acc_balance into account_balance  from account where acc_number =
sender;
   IF amount<=0 then
      RAISE EXCEPTION 'Not a valid amount!';
   end if;



   select status into stat from account where acc_number = sender;
   if (stat = 'Inactive') then
      Raise exception 'Entered Sender account is not active anymore';
   end if;

   select status into stat from account where acc_number = receiver;
   if (stat = 'Inactive') then
      Raise exception 'Entered Receiver account is not active anymore';
   ELSIF account_balance < amount then
      Raise exception 'Not enough balance!';
   else
      UPDATE account
      SET acc_balance = acc_balance - amount
      WHERE acc_number = sender;

      UPDATE account
      SET acc_balance = acc_balance + amount
      WHERE acc_number = receiver;
```

```
        INSERT INTO transaction (senders_acc_number, receivers_acc_number,
transaction_amount)
        VALUES(sender, receiver, amount);
    END IF;
    COMMIT;
END;  $$
;



CREATE OR REPLACE PROCEDURE loan_payment(
loan_id int,
amount NUMERIC
)
language plpgsql AS $$
BEGIN
IF NOT EXISTS(select loan_number from loan where loan_id=loan_number)  then
Raise exception ‘No loan exist with this loan id!’;
Elsif amount <= 0 then
Raise exception ‘Invalid amount’;
Else
    UPDATE loan
    SET loan_amount = loan_amount - amount
    WHERE
    loan_number = loan_id;

    INSERT INTO payment(loan_number, amount)
    VALUES (loan_id, amount);
End if;
    COMMIT;
END; $$
;
```

Final loan_payment procedure after triggers:

```
CREATE OR REPLACE PROCEDURE loan_payment(
loan_id int,
amount NUMERIC
)

language plpgsql AS $$
declare
amt NUMERIC;

BEGIN
  IF NOT EXISTS(select loan_number from loan where loan_id=loan_number)  then
    Raise exception 'No loan exist with this loan id!';
  Elsif amount <= 0 then
    Raise exception 'Invalid amount';
  END IF;

  select loan_amount into amt from loan where loan_number = loan_id;
  if amt = 0 then
    RAISE NOTICE 'Total loan amount has already been paid';
    return;
  ELsif amt < amount then
    UPDATE loan
    SET loan_amount = 0
    WHERE loan_number = loan_id;

    INSERT INTO payment(loan_number, amount)
    VALUES (loan_id, amt);
  Else
    UPDATE loan
    SET loan_amount = loan_amount - amount
    WHERE
```

```
        loan_number = loan_id;

     INSERT INTO payment(loan_number, amount)
     VALUES (loan_id, amount);
End if;
     COMMIT;
END; $$
;
```

Roles, grant permissions :

```
create role dbms_administrator with superuser encrypted password 'admin';
create user rupesh;
grant rupesh to db_administrator;
create user kundan;
grant kundan to db_administrator;
grant aditya to db_administrator
alter role rupesh login password 'admin'
alter role kundan login password 'admin'

create role manager with login encrypted password 'manager'

 grant select,update,delete,insert on
account,address,borrower,customer,employee,has_acc,loan,payment,transaction to
manager

create role cashier with login encrypted password 'cashier'
grant select,insert on transaction to cashier
grant select,update on account to cashier
```

create role accountant with login encrypted password 'accountant'

grant insert,delete,update,select on account,customer,has_acc to accountant

Triggers to be implemented

1. loan_payment = 0
2. loan_interest =

Points:

Employee salary(DONE)

Branch_id in loan already present.

Min age constraint for employee, opening account(customer),(DONE)

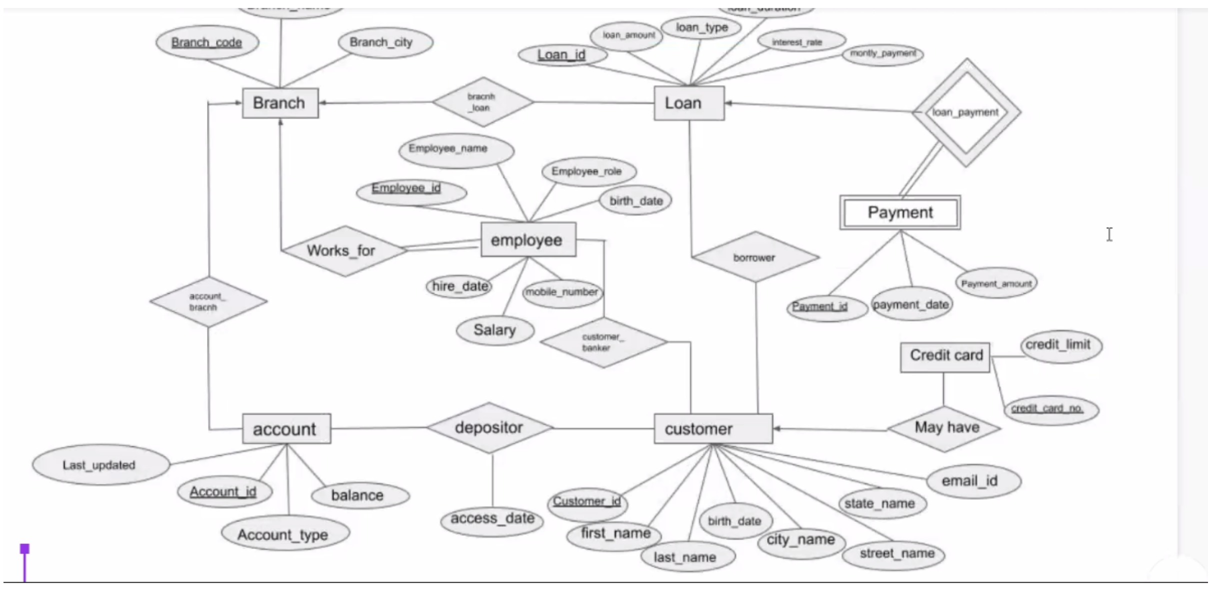
Loan interest and calculations

Customer who can be employee need to add person id to identify customer who is employee (use the natural join on the two tables that are customer and employee such that all the attributes like name,dob,address_id,phone number etc matches)(DONE)

SELECT * from customer natural join employee;

Assumption-

If the new customer comes to take a loan without an account then we have to add its cust_id in the has_acc and customer table and then allow the loan through borrower relation.



Mid term project report(pdf) should present following

1. Group name
2. name and roll of all members
3.
4. Structure of Database you are building
5. FULL functionality of it .

6. It should include a list of all tables with all constraints. Please give a good thought in design so that all constraints required on each field be specified in report
7. How constraints have been implemented either as a checksum or using some function through function ⇒ may be to use the triggers and views.
8. What kind of ROLE is there to access the data and what authentication each role has. -> Customer, Karyakarta

FUNCTIONS-

get_customer_details

```
CREATE OR REPLACE FUNCTION get_customer_details(customer_id int)
returns table(first_name varchar,last_name varchar,phone_number numeric,dob
date,street varchar,city varchar,pincode numeric )
AS $customer_details$
BEGIN
IF NOT EXISTS(select cust_id from customer where customer_id=cust_id) then
RAISE EXCEPTION ‘customer_id doesn’t exist in the database’;
else
return query
select customer.first_name, customer.last_name, customer.phone_number,
customer.dob, address.street, address.city, address.pincode
from customer, address
where customer.cust_id = customer_id and customer.address_id =
address.address_id ;


END IF;
END;
$customer_details$ LANGUAGE plpgsql;
```

Employee:

```
CREATE OR REPLACE FUNCTION get_employee_details(employee_id INT)
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, phone_no
NUMERIC, dob DATE
, manager_first_name VARCHAR, manager_last_name VARCHAR, street
VARCHAR, city VARCHAR, pincode NUMERIC)
AS $employee_details$
BEGIN
IF NOT EXISTS(select emp_id from customer where employee_id=emp_id) then
RAISE EXCEPTION ‘employee_id doesn’t exist in the database’;
Else
   RETURN QUERY select employee.first_name, employee.last_name,
employee.phone_no, employee.dob, (select employee.first_name as
manager_firstname
  from employee
  where employee.emp_id = employee.manager_id and employee.emp_id =
employee_id),
  (select employee.last_name as manager_lastname
  from employee
  where employee.emp_id = employee.manager_id and employee.emp_id =
employee_id)
  , address.street, address.city, address.pincode
from employee, address
where employee.emp_id = employee_id and employee.address_id =
address.address_id;
End if;
END;
$employee_details$ language plpgsql;
```

```
SELECT employee.first_name, employee.last_name, employee.phone_no,
employee.dob,
manager.first_name, manager.last_name, address.city, address.street,
address.pincode
FROM employee, employee AS manager, address
WHERE employee.emp_id = employee_id and manager.emp_id =
employee.manager_id and employee.address_id = employee.address_id;
```

```
CREATE OR REPLACE FUNCTION get_employee_details(employee_id INT)
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, phone_no
NUMERIC, dob DATE
, manager_first_name VARCHAR, manager_last_name VARCHAR, street
VARCHAR, city VARCHAR, pincode NUMERIC)
AS $employee_details$
BEGIN
   RETURN QUERY
                SELECT employee.first_name, employee.last_name,
        employee.phone_no, employee.dob,
        manager.first_name, manager.last_name, address.city, address.street,
        address.pincode
        FROM employee, employee AS manager, address
        WHERE employee.emp_id = employee_id and manager.emp_id =
        employee.manager_id and employee.address_id = employee.address_id;



END;
$employee_details$ language plpgsql;
```

i
FINAL get_employee_details function :

```
CREATE OR REPLACE FUNCTION get_employee_details(employee_id INT)
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, phone_no
NUMERIC, dob DATE
, manager_first_name VARCHAR, manager_last_name VARCHAR, street
VARCHAR, city VARCHAR, pincode NUMERIC)
AS $employee_details$
DECLARE
        Id INT;
BEGIN
        IF NOT EXISTS(select emp_id from employee where employee_id=emp_id)
then
RAISE EXCEPTION ‘employee_id doesn’t exist in the database’;
End if;
        SELECT manager_id INTO id FROM employee where emp_id =
        employee_id;
        If exists (SELECT manager_id  FROM employee where emp_id =
employee_id) then
        RETURN QUERY
        SELECT employee.first_name, employee.last_name, employee.phone_no,
        employee.dob,
        manager.first_name, manager.last_name, address.city, address.street,
        address.pincode
        FROM employee, employee AS manager, address
        WHERE employee.emp_id = employee_id and manager.emp_id =
        employee.manager_id and employee.address_id = address.address_id;
        Else
        RETURN QUERY


        select employee.first_name, employee.last_name, manager.first_name = null
        , manager.last_name = null , address.street, address.city  from employee,
        employee as manager, address where employee.emp_id = '1' and
        address.address_id = employee.address_id;

        End if;

END;
$employee_details$ language plpgsql;
```

Function display_account_details()

```
CREATE OR REPLACE FUNCTION
display_account_details(account_number INT)
RETURNS TABLE(acc_number INT, acc_balance NUMERIC,
interest_rate NUMERIC, branch_id INT, cust_id INT, first_name
VARCHAR, last_name VARCHAR
)
AS $account_details$
BEGIN
   RETURN QUERY select account.acc_number, account.acc_balance,
account.interest_rate, account.branch_id, customer.cust_id,
customer.first_name, customer.last_name
from customer, account, has_acc
where account.acc_number = '1234567891' and has_acc.acc_number =
account.acc_number and customer.cust_id = has_acc.cust_id;

END;
$account_details$ language plpgsql;
```

```
select * from INFORMATION_SCHEMA.role_table_grants;
psql -U accountant -d bankingsystem;
```

IMPROVEMENTS-(1st viva)
Buffer for hire date may be some days.
For transactions we have to check if amount is valid if yes then deduct else print some error message in the output.
Loan and account interest calculations.

Indexes to be implemented

Customer

1. create index cust_id_hash on customer using hash(cust_id);

2. CREATE EXTENSION btree_gin;

CREATE INDEX multi_name ON customer USING GIN (first_name,last_name);

Employee

1. create index emp_id_hash on employee using hash(emp_id);
2. CREATE INDEX multi_name_employee ON employee USING GIN (first_name,last_name);
3. create index emp_salary_index on employee using btree(emp_salary);
4. create index manager_id_index on employee using hash(manager_id);
5. create index hire_date_index on employee using btree(hire_date);

Has_acc

1. create index cust_id_has_acc on has_acc using hash(cust_id);

loan

1. create index loan_number_index on loan using hash(loan_number);
2. create index loan_amount_index on loan using btree(loan_amount);
3. create index branch_id_index on loan using hash(branch_id);

Payment

1. create index payment_index on payment using hash(loan_number);

Transaction

1. create index multi_transaction on transaction using btree(senders_acc_number,receivers_acc_number);
2. create index acc_number_index on account using hash(acc_number);
3. create index branch_id_idx on account using hash(branch_id);

4. create index acc_balance_idx on account using btree(acc_balance);

Address

1. create index address_id_index on address using hash(address_id);
2. create index city_index on address using gin(city);

3. create index pc_index on address using hash(pincode);

Borrower

1. create index cust_id_borrower on borrower using hash(cust_id);

Branch

1. create index branch_index on branch using hash(branch_id);
2. create index address_id_branch on branch using hash(address_id);

Triggers:

1. **Open_new_account (trigger inside the function) = first_name, last_anme, gender, dob, phone_number, address_id, acc_type, interest_rate, branch_id.**
2. **delete_account();**
3. **opening_of_loan();**
4. **Loan_amount ();**
5. **loan_payment();**
6.

Open_new_account:

```
CREATE OR REPLACE function create_customer(gen VARCHAR, f_name
VARCHAR, l_name VARCHAR, db DATE, ph_number NUMERIC,
str VARCHAR, ct VARCHAR, pin NUMERIC, a_type BOOLEAN, b_id INT)
RETURNS INT
AS $$

DECLARE
add_id NUMERIC;
c_id INT;
a_no INT;

BEGIN
   if(db >= CURRENT_DATE - '18 years'::interval) then
      RAISE EXCEPTION 'Invalid Age';
   ELSIF NOT EXISTS (select branch_id from branch where branch_id= b_id) then
      RAISE EXCEPTION 'Branch_id does not exist';
   END IF;

   if EXISTS (SELECT address_id from address where street = str and city = ct and
pincode = pin) then
```

```
        SELECT address_id INTO add_id from address where street = str and city = ct
and pincode = pin;
    else
        INSERT INTO address(street, city, pincode) values (str, ct, pin);
        SELECT address_id INTO add_id from address where street = str and city = ct
and pincode = pin;
    end if;


    INSERT INTO customer (gender, first_name, last_name, dob, phone_number,
address_id)
    values(gen, f_name, l_name, db, ph_number, add_id);

    if(a_type) then
        INSERT INTO account(acc_balance,interest_rate, acc_type, branch_id)
values(0, 0, 'Current', b_id);
    else
        INSERT INTO account(acc_balance,interest_rate, acc_type, branch_id)
values(0, 0.5, 'Saving', b_id);
    end if;
    select cust_id into c_id from customer
    where first_name = f_name and last_name = l_name and phone_number =
ph_number;

    select acc_number INTO a_no from account where acc_number = (select
max(acc_number) from account);

    INSERT INTO has_acc values (c_id, a_no);

    return 1;

END;
$$ language plpgsql;
```

Loan_amount calculate:

```
create or replace function calculate_amount()
returns trigger
AS $$
declare
   amt numeric;

BEGIN
   amt = new.loan_amount +
(new.loan_amount*new.loan_interest*new.loan_years)/100;
   new.loan_amount = amt;
   return new;
END;
$$ language plpgsql;

create trigger calculate_amount_trigger
before insert
on
loan
for each row
execute procedure calculate_amount();
```

Loan completion :

Changes in dbms

```
ALTER table customer add column status VARCHAR;
Alter table customer add constraint customer_status_check CHECK( status in ('Active', 'Inactive'));

Alter table customer Alter column status  set Default 'Active';
Alter table customer Alter column status  set not null;


Alter table account add column status VARCHAR Default 'Active';

Alter table customer add constraint account_status_check CHECK(status in ('Active','Inactive'));

Alter table account alter column status set not null;
```

Alter table account add constraint account_status_check CHECK(status in ('Active','Inactive'));

Final functions and procedure code is as follows

Close account procedure ⇒ not implemented in database.

```
CREATE Or REPLACE PROCEDURE close_account(
account_no int;
)
language plpgsql AS $$
declare
customer_id INT;
loan_amt NUMERIC;
stat VARCHAR;
BEGIN
   IF not EXISTS (select acc_number from account where acc_number  =
account_no) then
      Raise exception 'Account with this account number never existed';
   end if;

   select status into stat from account where acc_number = receiver;
   if (stat = 'Inactive') then
      Raise exception 'Entered account is already deactivated';
   end if;

   select cust_id into customer_id from has_acc where acc_number = account_no;
   select loan.loan_amount into loan_amt from loan, borrower
   where borrower.cust_id = customer_id and borrower.loan_number =
loan.loan_number;

   if(loan_amt > 0) then
      Raise exception 'Account can not be deactivated because of existing loan on
this account';
   end if;
```

```
    UPDATE table account set status = 'Inactive' where acc_number = account_no;
    UPDATE table customer set status = 'Inactive' where cust_id = customer_id;

    COMMIT;
END; $$
;
```

Problems of joint and merge account and loan:

1.

Rupesh

Function to delete the account which will just set the status as inactive

```
CREATE OR REPLACE function delete_customer_account(account_no integer)
RETURNS INT
AS $$


BEGIN

   if NOT EXISTS (SELECT acc_number from account where acc_number = account_no)
then
       raise exception 'Not a valid account number';
   else
       update account set status = 'Inactive' where acc_number = account_no;
                   update  account set acc_balance = 0 where acc_number = account_no;
   end if;


   return 1;

END;
$$ language plpgsql;
```

Get_account_details

```
CREATE OR REPLACE FUNCTION get_account_details(customer_id int)
returns table(account_number integer,  account_balance numeric, account_type varchar)
AS $account_details$
declare
countt integer;
BEGIN
        IF NOT EXISTS(select cust_id from customer where customer_id=cust_id) then
                RAISE EXCEPTION 'customer_id doesn’t exist in the database';

        ELSIF NOT EXISTS(select cust_id from has_acc where customer_id=cust_id) then
                RAISE EXCEPTION 'No account exist for this customer in the database';
        end if;

        select count(*) into countt from account, has_acc
        where has_acc.cust_id = customer_id and account.acc_number =
has_acc.acc_number and account.status = 'Active';

        IF (countt = 0)then
                RAISE EXCEPTION 'No active account exist in the database';
        else
        return query
                select account.acc_number, account.acc_balance, account.acc_type from
account, has_acc
                where has_acc.cust_id = customer_id and account.acc_number =
has_acc.acc_number and account.status = 'Active';
END IF;
END;
$account_details$ LANGUAGE plpgsql;
```

get_employee_details

```
CREATE OR REPLACE FUNCTION get_employee_details(employee_id INT)
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, phone_no NUMERIC, dob
DATE
, street VARCHAR, city VARCHAR, pincode NUMERIC, manager_first_name VARCHAR,
manager_last_name VARCHAR)
AS $employee_details$
BEGIN
IF NOT EXISTS(select emp_id from employee where employee_id=emp_id) then
RAISE EXCEPTION 'employee_id doesn’t exist in the database';
Else
   RETURN QUERY
                select employee.first_name, employee.last_name, employee.phone_no,
employee.dob, address.street, address.city,
                address.pincode, (select manager.first_name as manager_firstname from
employee as manager
```

```
                                                    where employee.manager_id =
manager.emp_id),
                                                    (select manager.last_name as
manager_lastname from employee as manager
                                                    where employee.manager_id =
manager.emp_id)
                from employee, address
                where employee.emp_id = employee_id and address.address_id =
employee.address_id;
End if;
END;
$employee_details$ language plpgsql;
```

Get_loan_details

```
CREATE OR REPLACE FUNCTION get_loan_details(customer_id INT)
RETURNS TABLE(loan_number integer, loan_amount numeric, branch_id integer,
loan_years integer, loan_interest numeric
                         , loan_timestamp timestamp)
AS $customer_loan_details$
declare
countt integer;

BEGIN
IF NOT EXISTS(select cust_id from customer where customer.cust_id = customer_id) then
RAISE EXCEPTION 'customer doesn’t exist in the database';
ELSIF NOT EXISTS (select cust_id from borrower where borrower.cust_id = customer_id)
then
        raise exception 'No loan exist for this customer';
END IF;
select count(*) into countt from loan, borrower
where borrower.cust_id = customer_id and loan.loan_number = borrower.loan_number and
loan.loan_amount > 0;


IF(countt = 0) then
        raise exception 'all loans are cleared for this customer';
Else
   RETURN QUERY
                select loan.loan_number, loan.loan_amount, loan.branch_id, loan.loan_years,
loan.loan_interest, loan.loan_timestamp from loan, borrower
                where borrower.cust_id = customer_id and loan.loan_number =
borrower.loan_number and loan.loan_amount > 0;
```

```
End if;
END;
$customer_loan_details$ language plpgsql;
```

Opening account

```
create or replace function open_account(f_name varchar[], l_name varchar[], gen varchar[],
dob_ date[]
                                                                    , p_number
numeric[], str varchar[], ct varchar[],
                                                                    pin integer
[],a_type varchar,  b_id integer)
returns int
as $$
declare
n integer := array_length(f_name, 1);
add_id integer[];
customer_id integer [];
a_number integer;
tempp integer;
begin
--      n = array_length(f_name, 1);
        if not exists (select branch_id from branch where branch_id = b_id) then
        raise exception 'Invalid branch id ';
        end if;
--      checking corner cases
        for i in 1..n
        loop

                if (dob_[i] >= CURRENT_DATE - '18 years'::interval) then
                raise exception 'Not eligible to open account due to age constraint';
                end if;
                if ((p_number[i] <= 999999999 OR p_number[i] > 9999999999)) then
                raise exception 'Phone number of customer is not valid';
                end if;
        end loop;

--      using the for loop here to check address exist or not
for i in 1..n
        loop
                if not exists (select address_id from address where street = str[i] and city =
ct[i] and pincode = pin[i]) then
                        insert into address (street, city, pincode) values (str[i], ct[i], pin[i]);
                        select address_id into tempp from address where street = str[i] and
city = ct[i] and pincode = pin[i];
                        add_id[i] = tempp;
                else
```

```
            select address_id into tempp from address where street = str[i] and
city = ct[i] and pincode = pin[i];
            add_id[i] = tempp;
        end if;

 end loop;



    for i in 1..n
    loop
        if not exists (select cust_id from customer where first_name = f_name[i] and
last_name = l_name[i] and dob = dob_[i] and
                    phone_number = p_number[i] and address_id =
add_id[i]) then
                    insert into customer (gender, first_name, last_name,
dob, phone_number, address_id) values
                    (gen[i], f_name[i], l_name[i], dob_[i], p_number[i],
add_id[i]);
        end if;
         select cust_id into tempp from customer where first_name = f_name[i] and
last_name = l_name[i] and dob = dob_[i] and
         phone_number = p_number[i] and address_id = add_id[i];
            customer_id[i] = tempp;
    end loop;
    if (a_type = 'Savings') then
    insert into account (acc_balance, interest_rate, acc_type, branch_id, status) values
(0, 5, a_type, b_id, 'Active');
    else
        insert into account (acc_balance, interest_rate, acc_type, branch_id, status)
values (0, 0, a_type, b_id, 'Active');
    END IF;
--  select acc_number into a_number from account where
    select acc_number INTO a_number from account where acc_number = (select
max(acc_number) from account);

    for i in 1..n
    loop
        insert into has_acc values (customer_id[i], a_number);
    end loop;

    return 1;

end;
$$ language plpgsql;
```

Sample query to run.

```
select open_account('{"Kundan", "name2"}':: varchar[], '{"Pal", "lname2"}' :: varchar[],
'{"Male", "Female"}' :: varchar[],
                                        '{"2000-12-12", "1999-10-11"}':: date[], '{1234567890,
9876543211}' :: numeric[],
                                        '{"abc", "street2"}' :: varchar[], '{"def", "city2"}' ::
varchar[], '{123456, 821109}':: integer[],
                                        'Savings', 1003);
```

open_loan();

```
create or replace function open_loan(f_name varchar[], l_name varchar[], gen varchar[],
dob_ date[]
                                                                                , p_number
numeric[], str varchar[], ct varchar[],
                                                                                pin integer [],
l_years int, l_amount numeric,b_id integer)
returns int
as $$
declare
n integer := array_length(f_name, 1);
add_id integer[];
customer_id integer [];
l_number integer;
tempp integer;
begin
--      n = array_length(f_name, 1);
        if (l_amount <=0) then
        raise exception 'Not a valid amount';
        elsif (l_years  <= 0) then
        raise exception 'not a valid time span for loan';
        end if;
        if not exists (select branch_id from branch where branch_id = b_id) then
        raise exception 'Invalid branch id ';
        end if;
--      checking corner cases
        for i in 1..n
        loop

                if (dob_[i] >= CURRENT_DATE - '18 years'::interval) then
                raise exception 'Not eligible to open account due to age constraint';
                end if;
                if ((p_number[i] <= 999999999 OR p_number[i] > 9999999999)) then
                raise exception 'Phone number of customer is not valid';
                end if;
        end loop;
```

```
--	using the for loop here to check address exist or not
for i in 1..n
	loop
		if not exists (select address_id from address where street = str[i] and city =
ct[i] and pincode = pin[i]) then
			insert into address (street, city, pincode) values (str[i], ct[i], pin[i]);
			select address_id into tempp from address where street = str[i] and
city = ct[i] and pincode = pin[i];
			add_id[i] = tempp;
		else
			select address_id into tempp from address where street = str[i] and
city = ct[i] and pincode = pin[i];
			add_id[i] = tempp;
		end if;

 end loop;



	for i in 1..n
	loop
		if not exists (select cust_id from customer where first_name = f_name[i] and
last_name = l_name[i] and dob = dob_[i] and
					phone_number = p_number[i] and address_id =
add_id[i]) then
					insert into customer (gender, first_name, last_name,
dob, phone_number, address_id) values
					(gen[i], f_name[i], l_name[i], dob_[i], p_number[i],
add_id[i]);
		end if;
		  select cust_id into tempp from customer where first_name = f_name[i] and
last_name = l_name[i] and dob = dob_[i] and
		  phone_number = p_number[i] and address_id = add_id[i];
			customer_id[i] = tempp;
	end loop;

	insert into loan (loan_amount, branch_id, loan_years, loan_interest) values
(l_amount, b_id, l_years, 8);
--	select acc_number into a_number from account where
	select loan_number INTO l_number from loan where loan_number = (select
max(loan_number) from loan);

	for i in 1..n
	loop
		insert into borrower values (customer_id[i], l_number);
	end loop;

	return 1;
```

```
end;
$$ language plpgsql;

select open_loan('{"Kundan", "name3"}':: varchar[], '{"Pal", "lname3"}' :: varchar[], '{"Male",
"Female"}' :: varchar[],
                                        '{"2000-12-12", "1999-10-11"}':: date[], '{1234567890,
9876543210}' :: numeric[],
                                        '{"abc", "street3"}' :: varchar[], '{"def", "city3"}' ::
varchar[], '{123456, 821107}':: integer[],
                                        10, 1000000, 1003);
```

Trigger for max_minlimit

```
-- trigger for loan
create or replace function check_amount()
returns trigger
AS $$
declare
   amount numeric;
        acc integer;
BEGIN

        amount = new.acc_balance;
        acc = new.acc_number;
        if(amount <= 5000) then
--              raise notice 'Your account has reached minimum balance limit ';
                raise notice 'Your account has reached minimum balance limit for account
number :', acc;
        end if;
        if(amount > 500000) then
                raise notice 'Your account has reached maximum balance limit for account
number :', acc;

        end if;
   return new;
END;
$$ language plpgsql;

create trigger warn_max_min_amount
after update
on
account
for each row
execute procedure check_amount();
```

Implementing create_account

Array of customer details.
We have to create a single account.
Arr1 = normal
Arr more = joint

We will check the age and phone number of all customers.
We will check whether all the addresses are existing gor not and if not then we will add them to the address table.
We will check the branch id

If existing then we will store the add_id in array
If not existing then we will add this address to the table and then fetch the recently added address id.

Customer
Now we have to insert the customer details one by one.
If a customer already exists then we do not need to add it again.

Has_acc
We have to insert all the customer id corresponding to the same acc_number.

Account
Only one entry will ,be added with 0 balance and b_id

has
Opening loan();
Array of customer details.
We have to create a loan
Arr1 = normal
Arr more = joint

We will check the age and phone number of all customers.
We will check whether all the addresses are existing gor not and if not then we will add them to the address table.
We will check the branch id
Insert into borrower
And insert into the customer.

Assumption
1. There can be customer without account or loan
2. But if a customer exists then he has taken either an account or loan in the present or past.

3. Account and loan are totally separate
4. To delete an account we are not considering a loan.
5. We allow only max 2 active loan from the bank for a particular customer.(trigger)
6. We allow only max 2 active accounts for a customer.(trigger)

Things to be done

1. Trigger

2. Make views according to the number of roles.
3. Some part of partial indexing to be done.

Hire_employee function

```
create or replace function hire_employee(f_name varchar,l_name varchar,p_no numeric,gen
varchar,db date,manag_id int,sal numeric,srt varchar,ct varchar,pc numeric)
returns int
as $$
declare
a_id int;
begin
if (p_no<1000000000 or p_no > 9999999999) then
raise exception 'Not a valid phone number!';
elsif (db >= current_date - '18 years' :: interval) then
raise exception 'Age of the employee is less than 18 years';
end if;
if not exists(select * from address where street=srt and city=ct and pc=pincode) then
insert into address (street,city,pincode) values(srt,ct,pc);
end if;
select address_id into a_id from address where street=srt and city=ct and pc=pincode;
insert into employee
(first_name,last_name,phone_no,gender,dob,emp_salary,manager_id,address_id)
values(f_name,l_name,p_no,gen,db,sal,manag_id,a_id);
return 1;end;
$$
language plpgsql;
```

-- transfer procedure
call transfer(1234567906,1234567893,10000);
call transfer(1234567906,12345789,1);
call transfer(1234567906,1234567896,30000);
call transfer(1234567894,1234567892,10000);

```
select * from transaction;
select * from account;
select * from payment;

-- deposit procedure
call deposit(1234567892,1000);
call deposit(1234567899,1000);

-- withdraw procedure
call withdraw(1234567892,1000);

-- loan_payment procedure
call loan_payment(123458,20000);
call loan_payment(1234565,1);
call loan_payment(1234566,180000000);

--taking a loan
select open_loan('{"Mark",
"angelina"}'::varchar[],'{"Henry","jolie"}'::varchar[],
'{"Male","Female"}'::varchar[], '{"1984-12-12", "1985-12-4"}'::date[],
'{1234567899, 9874563210}'::numeric[], '{"cement road", "Near
temple"}'::varchar[]
                        , '{"kochi", "kerela"}'::varchar[], '{123654,
326548}'::integer[], 3, 300000, 1000);

select * from loan;
select * from customer;
select * from borrower;
-- open account
select open_account('{"Jack", "Eminem"}':: varchar[], '{"Hanma",
"war"}' :: varchar[], '{"Male", "Male"}' :: varchar[],
                                    '{"1998-12-12", "1999-10-11"}':: date[],
'{1234967890, 9816543211}' :: numeric[],
                                    '{"kothri circle", "Ambedkar"}' ::
varchar[], '{"Guna", "Chomu"}' :: varchar[], '{193456, 121109}'::
integer[],
                                    'Savings', 1006);
```

```
-- delete customer account
select delete_customer_account(1234567898);

-- get employee details
select * from get_employee_details(6);

-- get customer details
select * from get_customer_details(12353);

-- get loan details
select * from get_loan_details(12343);

-- get account details
select * from get_account_details(12348);

-- add new employee
select hire_employee('James', 'cavill', 6541239870, 'Male',
'1974-07-01', 1, 65000, 'Near Dominos', 'Palghat', 654789);
select * from employee;

-- trigger for loan
select open_loan('{"Marco",
"angel"}'::varchar[],'{"Henry","jolie"}'::varchar[],
'{"Male","Female"}'::varchar[], '{"1984-12-12", "1985-12-4"}'::date[],
'{1234567899, 9874563210}'::numeric[], '{"cement road", "Near
temple"}'::varchar[]
                          , '{"kochi", "kerela"}'::varchar[], '{123654,
326548}'::integer[], 3, 300000, 1000);
select * from loan;

-- trigger for max min amount
call transfer(1234567891,1234567892,10000);

-- trigger for loan completion
call loan_payment(1234564,100)

-- to check the permissions on a particular table
```

```
SELECT grantee, privilege_type,table_name
FROM information_schema.role_table_grants
WHERE grantee='recruitment_manager';
```

```
call open_loan('{"Mark", }','Henry', 'Male', '2018-12-12', 1234567899, '46', 'kochi',
123654, 3, 300000, 1000)
```