LAB COMP3161: Database Management Systems

Data Requirements for a Bank Database

- The Bank is organized into branches. Each branch is located in a particular city and is identified by a unique name. The bank monitors the assets of each branch.

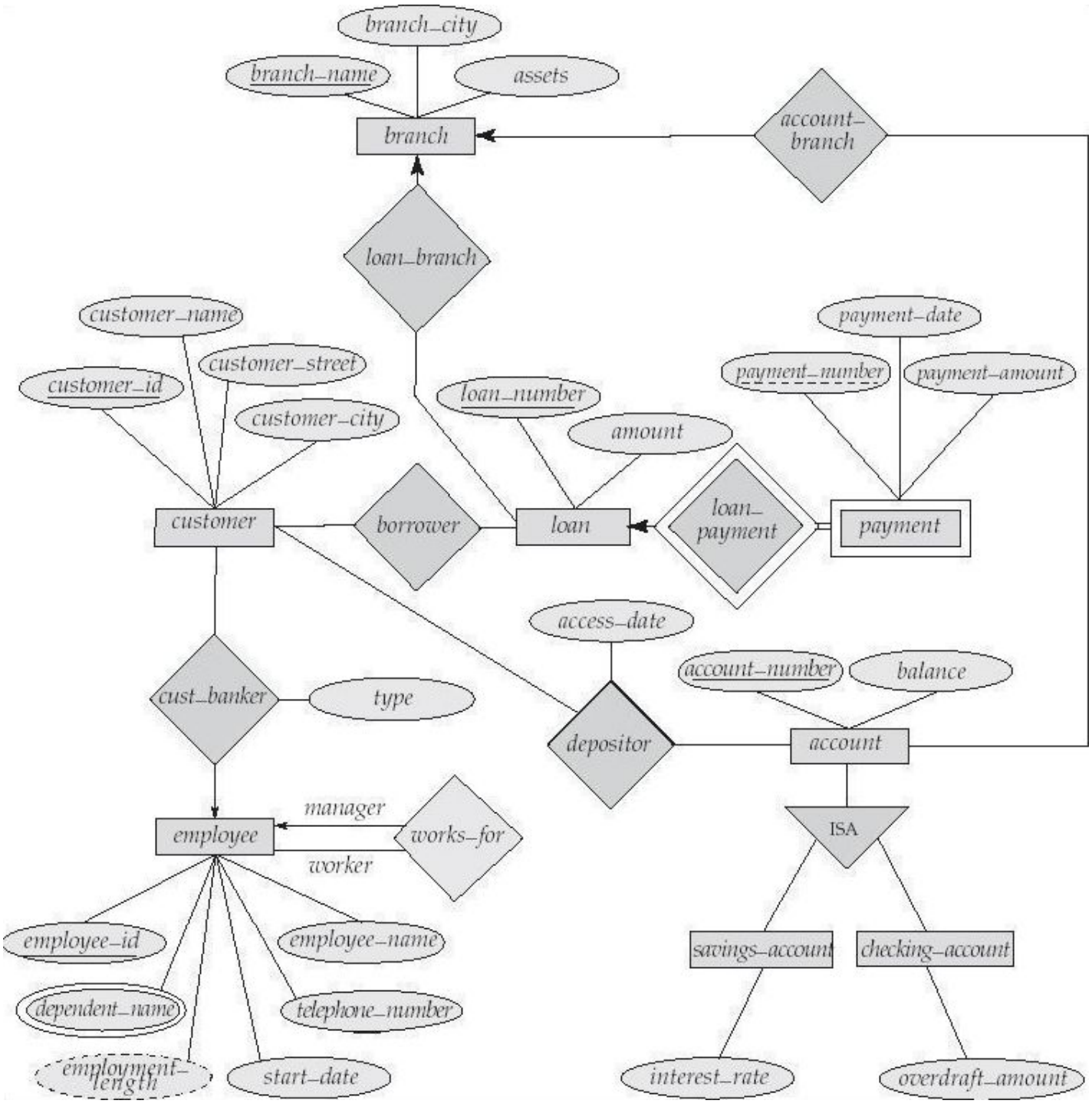
- Bank customers are identified by their *TRN*. The bank stores each customer's name and the street and the city where the customer lives. Customers may have accounts and can take out loans. A customer may be associated with a particular banker, who may act as loan officer or a personal banker for that customer.

- Bank employees are identified by their *employee_id* values. The bank administration stores the name and telephone number of each employee, the names of the employee's dependents, and the *employee_id* number of the employee's manager. The bank also keeps a track of the employee's start date, and thus, length of employment.

- The bank offers two types of accounts – savings and checking accounts. Accounts can be held by more than one customer, and a customer can have more than one account. Each account is assigned a unique account number. The bank maintains a record of each customer's account balance and the most recent date on which the account was accessed by each customer holding the account. In addition, each savings account has an interest rate and overdrafts are recorded for each checking account.

- A loan originates at a particular branch and can be held by one or more customers. A loan is identified by a unique loan number. For each loan, the bank keeps track of the loan amount and the loan payments. Although a loan payment number does not uniquely identify a particular payment among those for all the bank's loans, a payment number does identify a particular payment for a specific loan. The date and amount are recorded for each payment.

E-R Diagram


branch_city
branch_name
assets
branch
account_branch
loan_branch
customer_name
customer_street
customer_id
customer_city
loan_number
amount
payment_date
payment_number
payment_amount
customer
borrower
loan
loan_payment
payment
access_date
account_number
balance
cust_banker
type
depositor
account
manager
employee
works_for
worker
ISA
employee_id
employee_name
dependent_name
telephone_number
employment_length
start_date
savings_account
checking_account
interest_rate
overdraft_amount

Relational Schema for Banking Enterprise

NB: You need to check all the schemas and ensure that all the necessary data is represented from the ER Diagram

- Schemas derived from a strong entity;

  branch(branch_name, branch_city, assets)
  customer(customer_id, customer_name, customer_street, customer_city)
  loan(loan number, amount)
  account(account_number, balance)
  employee(employee_id, employee_name, telephone_number, start_date)

- Schemas derived from a weak entity set

  p*ayment(loan_number, payment_number, payment_date, payment_amout)*

- Schemas derived from a multivalued attribute:

  d*ependent_name(employee_id, d_name)*

We need not represent the derived attributes. They are defined in views or a specially defined function.

- Schemas derived from a relationship set involving strong entity sets

  a*ccount_branch(account_number, branch_name)*
  loan_branch(loan_number, branch_name)
  borrower(customer_id, loan_number)
  depositor(customer_id, account_number)
  cust_banker(customer_id, employee_id, type)
  works_for(worker-employee_id, manage_employee_id) note that the relationship is a one to many and so a candidate key is that of the 'many' entity

Tables derived from relationships between a weak and its corresponding dominant entity are redundant.

- Schemas derived from ISA relationship: (We are assuming that there are accounts that are neither savings nor checking accounts)

  s*avings_account(account_number, interest_rate)*
  checking-account(account-number, overdraft-amount)

Exercise1.1
Using SQL create the tables for **customer**, **loan** and **employee** in your database.

Exercise 1.2
Create the tables for the entire banking database as specified above. Be sure to state all keys and enforce integrity constraints.

Exercise 1.3
On updating the account number field, in the account table, all occurrences of account number in the account_branch table should also be updated. How can this integrity constraint be enforced?

Exercise1.4
Enter some sample/test data in the database. Try and include values that are relevant for the queries that are asked below.

SQL – QUESTIONS

-

Using the relations of our bank database, write SQL expressions to define the following:

1) The names and addresses of all customers who have an account with the bank, but do not have a loan.

2) The name and average account balance of every customer of the Rock Ridge Branch.

3) Give all borrowers a 10% deduction on their loan balances.

4) Create a view listing account numbers, balances and customer names of all depositors at the Downtown branch.

5) Alter the account table so that we can capture the status (active, inactive) of an account.

6) Set the status field of all depositors, who has less than $100 in their account, to inactive and those with balance greater than or equal to $100 to active.

One way to achieve this is to execute two update queries. However there is an easier way, using case statements.

Update relation
Set attribute1 = **case**
when predicate **then** <update rule>
when predicate **then** <update rule>
…
else <update rule>
end

7) Find all branches where the total account deposit is less than the average total account deposit at all branches.

8) Delete the records of all accounts with balances below the average at the bank.

9) Find the names of all branches that have greater assets than all branches located in Brooklyn.

10) Find the average account balance of those branches where the average account balance is greater than $1500.

** END **