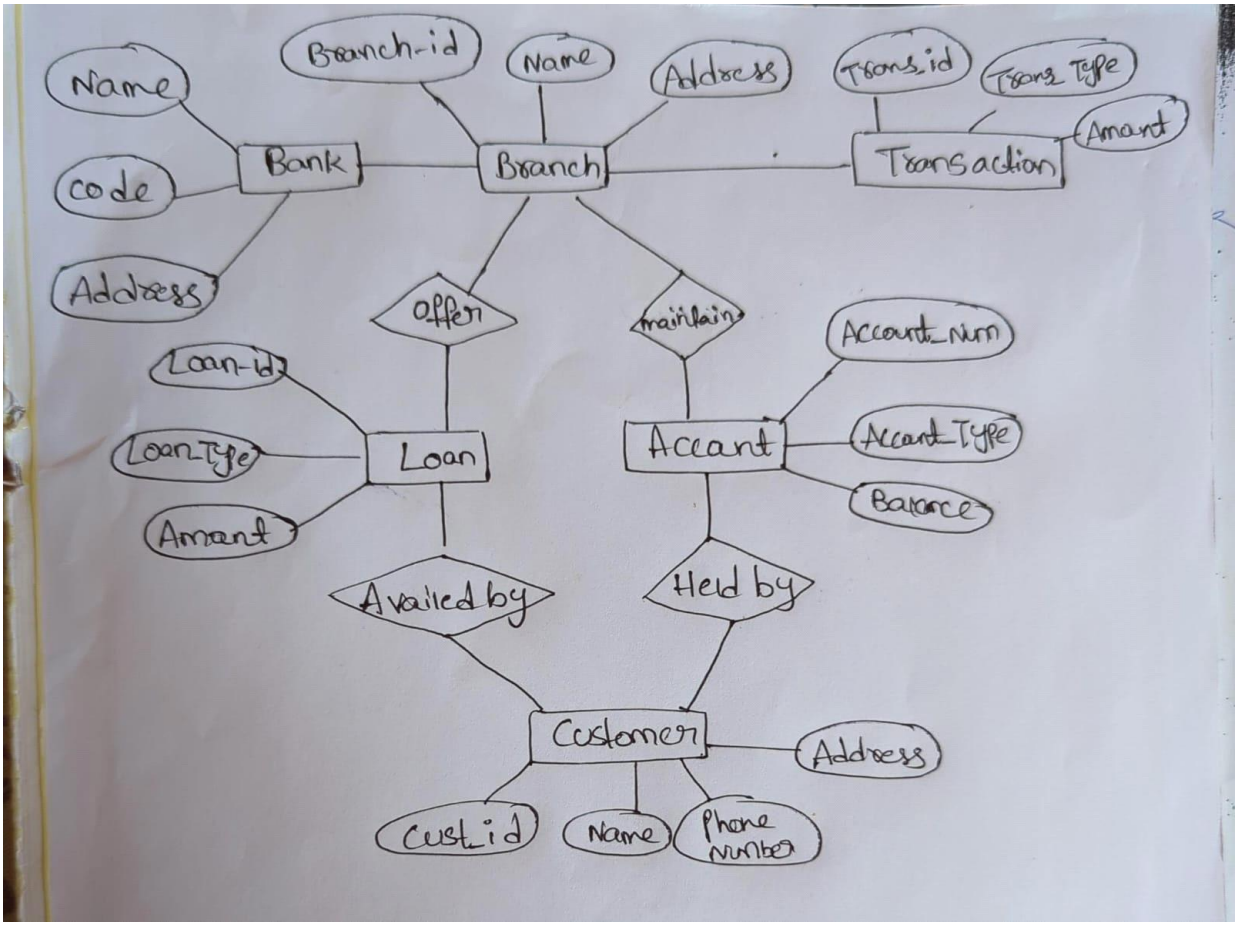
Assignment 1: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

ER diagram of Bank management System



Bank has Branches => 1 : N

One Bank can have many Branches but one Branch can not belong to many Banks, so the relationship between Bank and Branch is one to many relationship.

Branch maintain Accounts => 1 : N

One Branch can have many Accounts but one Account can not belong to many Branches, so the relationship

between Branch and Account is one to many relationship.

Branch offer Loans => 1 : N

One Branch can have many Loans but one Loan can not belong to many Branches, so the relationship between

Branch and Loan is one to many relationship.

Account held by Customers => M : N

One Customer can have more than one Accounts and also One Account can be held by one or more Customers, so the relationship

between Account and Customers is many to many relationship.

Loan availed by Customer => M : N

(Assume loan can be jointly held by many Customers).

One Customer can have more than one Loans and also One Loan can be availed by one or more Customers, so the relationship between

Loan and Customers is many to many relationship.

Assignment 3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a

transaction that includes locking and demonstrate different isolation levels to show concurrency control.

ACID Properties of Transactions

ACID is an acronym that stands for four key properties that ensure the integrity and consistency of data during

transactions in a database:

Atomicity: This property guarantees that a transaction is treated as a single, indivisible unit of work. Either all the

operations within the transaction are completed successfully (committed), or none of them are (rolled back). This prevents

partial updates that could leave the database in an inconsistent state.

Consistency: Consistency ensures that transactions only bring the database from one valid state to another. Transactions

enforce pre-defined business rules to maintain data integrity. For instance, a transaction transferring funds between accounts

wouldn't allow a negative balance in either account.

Isolation: Isolation guarantees that concurrent transactions executing at the same time do not interfere with each other's data.

Even if multiple transactions access and modify the same data, the results should be the same as if the transactions were executed sequentially.

Durability: Durability ensures that once a transaction is committed, the changes are persisted permanently in the database. Even if a system

failure occurs after the commit, the changes should not be lost. This is typically achieved by writing the changes to stable storage like a disk.

Now, let's demonstrate these properties with SQL statements:

```
-- Simulating a transaction with locking

BEGIN TRANSACTION;


-- Updating balance of two accounts

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

In this transaction, we deduct 100 units from account 1 and add 100 units to account 2. This operation should be

executed atomically. If one of the updates fails, both should be rolled back.

Now, let's demonstrate different isolation levels to show concurrency control:

```
-- Set isolation level to READ COMMITTED

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;


-- Start a transaction

BEGIN TRANSACTION;


-- Selecting balance from account 1

SELECT balance FROM accounts WHERE account_id = 1;


-- Another user updates the balance of account 1

UPDATE accounts SET balance = balance - 50 WHERE account_id = 1;


-- Selecting balance from account 1 again

SELECT balance FROM accounts WHERE account_id = 1;


-- Commit the transaction

COMMIT;
```

In this scenario, if the isolation level is set to READ COMMITTED, the second SELECT statement will fetch the

updated balance after the other user's update. This prevents dirty reads, but it may lead to non-repeatable reads.

Similarly, you can demonstrate other isolation levels like READ UNCOMMITTED, REPEATABLE READ, or SERIALIZABLE and

observe how they affect concurrency control and the visibility of data changes made by other transactions.