Fundamentals of Database Systems

LAB REPORT # 12 (Experiment 9)

STUDENT'S NAME: Muhammad Hassham

ROLL No. : 19L-1380

SECTION: EE-8A-1

Introduction:

To begin with, Through this experiment, we have the opportunity to discover new insights about Erwin. Erwin is a popular data modeling tool utilized by organizations for data design, visualization, and management. Data modeling involves developing a conceptual portrayal of data, aiding in comprehending intricate data structures, recognizing probable problems, and guaranteeing consistency in data. Erwin presents an extensive range of features and tools that empower users to construct and administer data models for diverse applications and databases. With Erwin, users can conveniently design and edit data models through drag-and-drop functionality, work together with team members, and generate comprehensive reports to document their progress.

Objective:

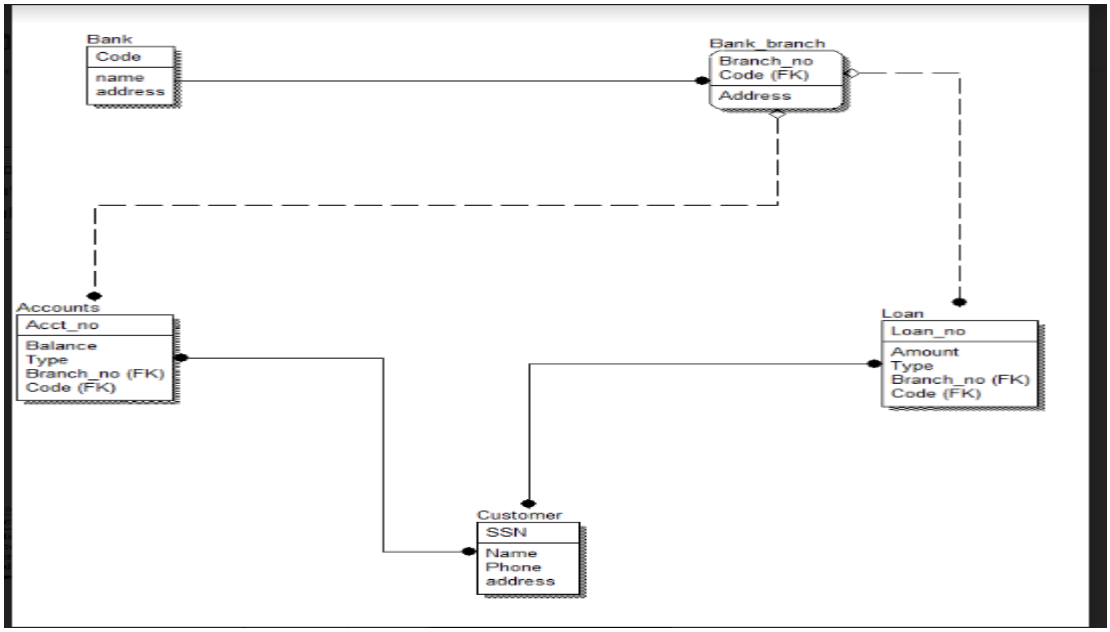
The aim of this experiment is to gain knowledge about Erwin and become acquainted with its functioning.

Design:

To commence the experimental procedure, we will need to procure the Erwin software which is an indispensable tool for conducting the said experiment. Once the software is obtained, we will initiate the process by opening the software and creating a brand-new file to commence our experimentation. Subsequently, we will select the Logical model option which will bring forth a new project window, providing us with an ideal environment to perform our experiments. Now, to create an entity, we will need to click on the entity icon situated in the toolbar of the Erwin software. Once the entity icon is clicked, a cursor will be displayed on the blank sheet, and we will position the cursor accordingly. Upon positioning the cursor, the entity shape will be generated on the blank sheet. At this point, we will proceed to name the entity and define the primary attribute or key, which is a crucial aspect of the entity's functioning. We will accomplish this task by pressing the tab key which will shift the cursor to the primary attribute row. Here, we will enter the name of the primary key. Once the primary key is defined, we will press the tab key once again to proceed to the non-prime attributes. In the non-prime attribute section, we will enter the attribute names which are relevant to the entity we are creating. After entering the attribute names, we will press the enter key, which will culminate in the completion of our entity creation process. Overall, this intricate procedure, comprising various steps, will allow us to create an entity using the Erwin software, which will enable us to perform our experiment with ease and precision.

Moving forward, our subsequent objective is to establish a relationship between the two entities, which can be accomplished through two types of relationships, i.e., non-identifying and identifying relationships. It is imperative to understand the differences between the two relationships. In the case of an identifying relationship, the child table's existence is dependent on the parent table's existence, and the child table's primary key is also a foreign key that refers to the parent table's primary key. For instance, a

customer's order record can be identified by the customer's primary key. On the other hand, in a non-identifying relationship, the child table can exist independently of the parent table. The child table's foreign key refers to the parent table's primary key, but it does not form part of the child table's primary key. For example, a sales representative's commission record can be linked to a sales territory, but the commission record can still exist even if the sales territory record is deleted. After comprehending the different types of relationships, we will proceed to link our entities accordingly. Furthermore, we will also utilize the many-to-many cardinality ratio in our lab tasks. In essence, the task for our lab is illustrated below, and we will execute it in a systematic and efficient manner.



Issues:

There were no issues encountered during this lab session.

Conclusion:

In the light of output obtained, Hence in the end we will conclude that we will learn about the Erwin understand it properly by implementing it in the creation of model.

Application:

The applications of triggers are as follows.

- **Making sure data is correct:** Erwin makes sure that data is accurate, consistent, and follows rules and laws.
- **Analyzing data**: Erwin helps people create models of data that are easy to use for looking at data and finding patterns.
- **Creating software**: Erwin helps people design the parts of software that deal with data, like storing and retrieving information.

- **Designing databases:** Erwin helps people create and change databases quickly and easily

Post Lab

1)

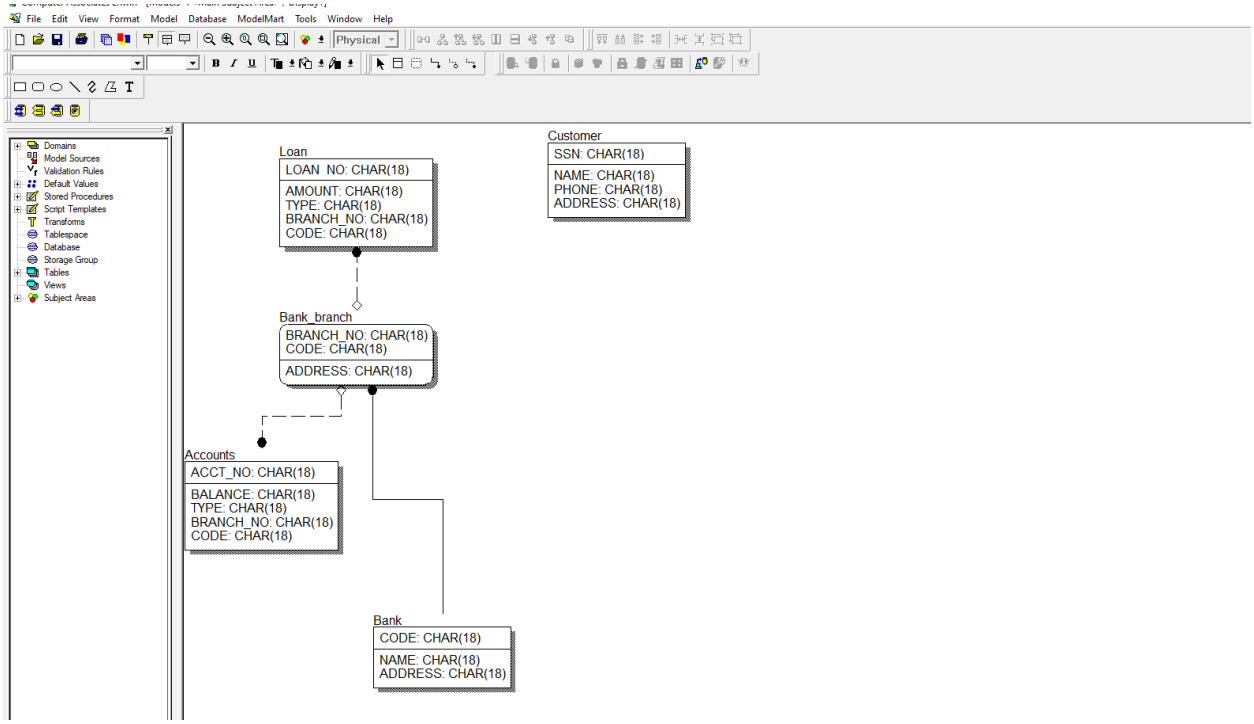
```
CREATE TABLE Accounts (
    Acct_no          CHAR(18) NOT NULL,
    Balance          CHAR(18),
    Type             CHAR(18),
    Branch_no        CHAR(18),
    Code             CHAR(18)
);

CREATE UNIQUE INDEX XPKAccounts ON Accounts
(
    Acct_no                  ASC
);


ALTER TABLE Accounts ADD PRIMARY KEY (Acct_no);
CREATE TABLE Bank (
    Code             CHAR(18) NOT NULL,
    name             CHAR(18),
    address          CHAR(18)
);

CREATE UNIQUE INDEX XPKBank ON Bank
(
    Code                     ASC
);


ALTER TABLE Bank
    ADD PRIMARY KEY (Code);
CREATE TABLE Bank_branch (
    Branch_no        CHAR(18) NOT NULL,
    Code             CHAR(18) NOT NULL,
    Address          CHAR(18)
);

CREATE UNIQUE INDEX XPKBank_branch ON Bank_branch
(
    Branch_no                ASC,
    Code                     ASC
);
ALTER TABLE Bank_branch ADD PRIMARY KEY (Branch_no, Code);
CREATE TABLE Customer (
    SSN              CHAR(18) NOT NULL,
    Name             CHAR(18),
```

```
    Phone           CHAR(18),
    address         CHAR(18)
);

CREATE UNIQUE INDEX XPKCustomer ON Customer
(
    SSN                     ASC
);
ALTER TABLE Customer ADD PRIMARY KEY (SSN);
CREATE TABLE Loan (
    Loan_no         CHAR(18) NOT NULL,
    Amount          CHAR(18),
    Type            CHAR(18),
    Branch_no       CHAR(18),
    Code            CHAR(18)
);

CREATE UNIQUE INDEX XPKLoan ON Loan
(
    Loan_no                 ASC
);
ALTER TABLE Loan
    ADD PRIMARY KEY (Loan_no);
ALTER TABLE Accounts
    ADD FOREIGN KEY (Branch_no, Code)
                REFERENCES Bank_branch
                ON DELETE SET NULL;
ALTER TABLE Bank_branch ADD FOREIGN KEY (Code) REFERENCES Bank ON DELETE RESTRICT;
ALTER TABLE Loan ADD FOREIGN KEY (Branch_no, Code) REFERENCES Bank_branch ON DELETE SET
NULL;
create trigger tI_Accounts after INSERT on Accounts
  REFERENCING NEW AS NEW for each row mode db2sql
   update Accounts
    set
     Branch_no = NULL,
     Code = NULL
    where
     not exists (
      select * from Bank_branch
       where
        new.Branch_no = Bank_branch.Branch_no and
        new.Code = Bank_branch.Code
     )
create trigger tU_Accounts after UPDATE on Accounts
    for each row mode db2sql
   update Accounts
    set
     Branch_no = NULL,
     Code = NULL
    where
     not exists (
      select * from Bank_branch
       where
        new.Branch_no = Bank_branch.Branch_no and
        new.Code = Bank_branch.Code
     )
create trigger tD_Bank after DELETE on Bank
  REFERENCING OLD AS OLD for each row mode db2sql
  WHEN (0 < (select count(*) from Bank_branch where Bank_branch.Code =
```

```
old.Code))
 BEGIN ATOMIC
    SIGNAL SQLSTATE '75001' ('Cannot DELETE Bank because Bank_branch
exists.');
 END
create trigger tU_Bank after UPDATE on Bank
    for each row mode db2sql
  WHEN (0 < ((select count(*) from Bank where Bank.Code <> old.Code))
AND
    (0 < (select count(*) from Bank_branch where Bank_branch.Code =
old.Code)))
 BEGIN ATOMIC
      SIGNAL SQLSTATE '75001' ('Cannot UPDATE Bank because Bank_branch
exists.');
 END
create trigger tD_Bank_branch after DELETE on Bank_branch
  REFERENCING OLD AS OLD for each row mode db2sql
   update Accounts
    set
      Branch_no = NULL,
      Code = NULL
    where
      Accounts.Branch_no = old.Branch_no and
      Accounts.Code = old.Code
create trigger tD_Bank_branch2 after DELETE on Bank_branch
  REFERENCING OLD AS OLD for each row mode db2sql
   update Loan
    set
      Branch_no = NULL,
      Code = NULL
    where
      Loan.Branch_no = old.Branch_no and
      Loan.Code = old.Code
create trigger tI_Bank_branch after INSERT on Bank_branch
  REFERENCING NEW AS NEW for each row mode db2sql
  WHEN ((0 = (select count(*) from Bank where new.Code = Bank.Code))
   )
 BEGIN ATOMIC
      SIGNAL SQLSTATE '75001' ('Cannot INSERT Bank_branch because Bank
does not exist.');
 END
create trigger tU_Bank_branch after UPDATE on Bank_branch
    for each row mode db2sql
  WHEN (0 < (select count(*) from Bank_branch where
Bank_branch.Branch_no <> old.Branch_no or Bank_branch.Code <> old.Code))
    update Accounts
     set
       Branch_no = NULL,
       Code = NULL
     where
       Accounts.Branch_no = old.Branch_no and
       Accounts.Code = old.Code
 create trigger tU_Bank_branch2 after UPDATE on Bank_branch
    for each row mode db2sql
  WHEN (0 < (select count(*) from Bank_branch where
Bank_branch.Branch_no <> old.Branch_no or
                                    Bank_branch.Code
<> old.Code))
    update Loan
```

```
      set
        Branch_no = NULL,
        Code = NULL
      where
        Loan.Branch_no = old.Branch_no and
        Loan.Code = old.Code
create trigger tU_Bank_branch3 after UPDATE on Bank_branch
   for each row mode db2sql
  WHEN (0 = ((select count(*) from Bank where new.Code = Bank.Code))
    )
BEGIN ATOMIC
     SIGNAL SQLSTATE '75001' ('Cannot UPDATE Bank_branch because Bank
does not exist.');
END
create trigger tI_Loan after INSERT on Loan
  REFERENCING NEW AS NEW for each row mode db2sql
   update Loan
    set
      Branch_no = NULL,
      Code = NULL
    where
      not exists (
        select * from Bank_branch
          where
            new.Branch_no = Bank_branch.Branch_no and
            new.Code = Bank_branch.Code
      )
create trigger tU_Loan after UPDATE on Loan
   for each row mode db2sql
  update Loan
    set
      Branch_no = NULL,
      Code = NULL
    where
      not exists (
        select * from Bank_branch
          where
            new.Branch_no = Bank_branch.Branch_no and
            new.Code = Bank_branch.Code
      )
```

PART-2)


File Edit View Format Model Database ModelMart Tools Window Help
Physical
Domains
Model Sources
Validation Rules
Default Values
Stored Procedures
Script Templates
Transforms
Tablespace
Database
Storage Group
Tables
Views
Subject Areas
Loan
LOAN_NO: CHAR(18)
AMOUNT: CHAR(18)
TYPE: CHAR(18)
BRANCH_NO: CHAR(18)
CODE: CHAR(18)
Customer
SSN: CHAR(18)
NAME: CHAR(18)
PHONE: CHAR(18)
ADDRESS: CHAR(18)
Bank_branch
BRANCH_NO: CHAR(18)
CODE: CHAR(18)
ADDRESS: CHAR(18)
Accounts
ACCT_NO: CHAR(18)
BALANCE: CHAR(18)
TYPE: CHAR(18)
BRANCH_NO: CHAR(18)
CODE: CHAR(18)
Bank
CODE: CHAR(18)
NAME: CHAR(18)
ADDRESS: CHAR(18)