**Fundamentals of Database Systems**

LAB REPORT # 12 (Experiment 9)

**NAME:** Muhammad Uzman Khan

**ROLL No. :** 19L-1359

**Introduction:**

To start with, this experiment provides us with the opportunity to uncover fresh insights into Erwin. Erwin is a widely used data modeling tool that assists organizations in designing, visualizing, and managing their data. Data modeling is the process of creating a conceptual representation of data, which helps to understand complex data structures, identify potential issues, and ensure data consistency. Erwin offers a broad range of features and tools that enable users to build and maintain data models for various applications and databases. With Erwin, users can easily create and modify data models using drag-and-drop functionality, collaborate with team members, and generate comprehensive reports to document their progress.

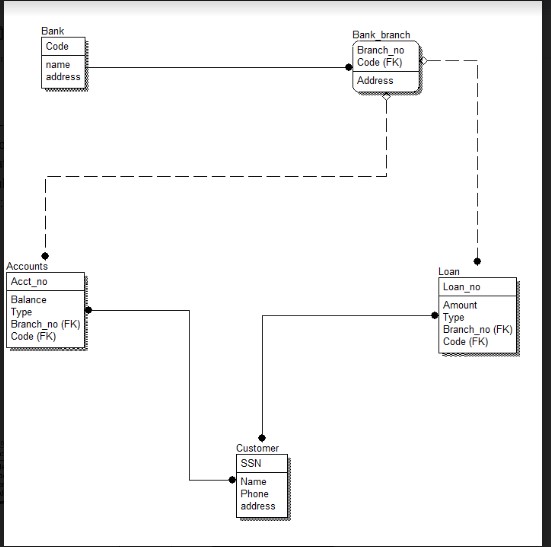
**Objective:**

The aim of this experiment is

* to gain knowledge about Erwin
* and become acquainted with its functioning.

**Design:**

To begin the experimental procedure, we must acquire the essential Erwin software, which is necessary for conducting the experiment. Once we have obtained the software, we will begin by opening it and creating a new file to start our experimentation. Then, we will choose the Logical model option, which will bring up a new project window and provide us with an ideal environment to conduct our experiments. To create an entity, we will click on the entity icon located in the Erwin software toolbar. When we click the icon, a cursor will appear on the blank sheet, and we will position it appropriately. Once positioned, the entity shape will appear on the blank sheet, and we will proceed to name the entity and define the primary attribute or key, which is a vital aspect of the entity's functionality. To achieve this, we will press the tab key, which will move the cursor to the primary attribute row, where we will enter the primary key name. Once defined, we will press the tab key again to proceed to the non-prime attributes section, where we will enter relevant attribute names for the entity we are creating. After entering the attribute names, we will press the enter key to complete the entity creation process. This intricate process will enable us to create an entity using the Erwin software, allowing us to conduct our experiment accurately and efficiently.

Our subsequent objective is to establish a relationship between the two entities, which can be achieved through two types of relationships: identifying and non-identifying. It is crucial to understand the differences between the two relationships. In 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 example, a customer's order record can be identified by the customer's primary key. In contrast, 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 is not part of the child table's primary key. For instance, 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. Once we understand the different types of relationships, we will link our entities accordingly. Additionally, we will use the many-to-many cardinality ratio in our lab tasks. We will execute the task below systematically and efficiently in our lab.

**Issues:**

There were **no issues** encountered during this lab session.

**Conclusion:**

Based on the output received, we can conclude that a proper understanding of Erwin can be achieved by implementing it in the creation of a model.

**Application:**

Triggers have various applications, such as:

• Ensuring data accuracy: Erwin ensures that data is correct, follows rules and regulations, and maintains consistency.

• Analyzing data: Erwin assists in creating data models that are user-friendly and allow for data analysis to find patterns.

• Developing software: Erwin facilitates the design of software components that handle data-related tasks, such as data storage and retrieval.

• Creating and modifying databases: Erwin simplifies the process of designing and modifying databases in a quick and efficient manner.

**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)

