Data and Applications

Homework-4

Team-50 : SYSTUM*

Prakhar Singhal: 2022111025
Chirag Dhamija: 2022101087
Mohak Somani: 2022101088
Samagra Bharti: 2022117014

November 23, 2023

Question - 1:

ISBN	Author	Genre	Publish_Date
9780545010221	J.K. Rowling	Fantasy	1997-06-26
9780439139601	J.K. Rowling	Fantasy	1998-06-02
9780439358071	J.K. Rowling	Fantasy	1999-07-08
9780061120084	Harper Lee	Fiction	1960-07-11
9780141439587	Jane Austen	Classic	1813-01-28
9781840227805	Jane Austen	Classic	1811-10-30
9780316769488	J.D. Salinger	Short Stories	1953-01-31

Title	Author
"Harry Potter and the Sorcerer's Stone"	J.K. Rowling
"Harry Potter and the Chamber of Secrets"	J.K. Rowling
"Harry Potter and the Prisoner of Azkaban"	J.K. Rowling
"To Kill a Mockingbird"	Harper Lee
"Pride and Prejudice"	Jane Austen
"Sense and Sensibility"	Jane Austen
"Nine Stories"	J.D. Salinger

A spurious tuple is a row in a database table that is created when two tables are joined incorrectly. Here are the spurious tuples that will be formed by the joining of BookInfo1 and BookInfo2 Tables. Managing the creation of spurious tuples requires establishing resilient data validation methods, guaranteeing adequate constraints, conducting routine assessments of data quality, authenticating inputs, and meticulously upholding precise documentation of database schemas and relationships. This safeguards against unintended errors or discrepancies in the stored data.

ISBN	Author	Title	Genre	Publication Date
9780439139601	J.K. Rowling	”Harry Potter and the Chamber of Secrets”	Fantasy	1998-06-02
9780439358071	J.K. Rowling	”Harry Potter and the Prisoner of Azkaban”	Fantasy	1999-07-08
9780545010221	J.K. Rowling	”Harry Potter and the Sorcerer’s Stone”	Fantasy	1997-06-26
9780439358071	J.K. Rowling	”Harry Potter and the Prisoner of Azkaban”	Fantasy	1999-07-08
9780545010221	J.K. Rowling	”Harry Potter and the Sorcerer’s Stone”	Fantasy	1997-06-26
9780439139601	J.K. Rowling	”Harry Potter and the Chamber of Secrets”	Fantasy	1998-06-02
9781840227805	Jane Austen	”Sense and Sensibility”	Classic	1811-10-30
9780141439587	Jane Austen	”Pride and Prejudice”	Classic	1813-01-28

Table 1: Spurious Tuples

Question - 2:

Attributes	O	C	T	(O,C)	(O,T)	(C,T)	(O,C,T)
O	–	YES	YES	YES	YES	YES	YES
C	NO	–	NO	NO	NO	NO	NO
T	NO	NO	–	NO	NO	NO	NO
(O,C)	YES	YES	YES	–	YES	YES	YES
(O,T)	YES	YES	YES	YES	–	YES	YES
(C,T)	YES	YES	YES	YES	YES	–	YES
(O,C,T)	YES	YES	YES	YES	YES	YES	–

Table 2: Functional Dependency Matrix

ENTRY $M[i, j]$ represents whether the Functional Dependency $i \rightarrow j$ holds or not.
O denotes Order_Id
C denotes Customer_Id
T denotes Total_Price
Functional Dependencies:

- $O \rightarrow C$ (**YES**): Each order corresponds to a single customer. Therefore, an order uniquely determines a customer (in a well-designed system).
- $O \rightarrow T$ (**YES**): Each order has a specific total price. Hence, an order uniquely determines the total price in a well-maintained database.
- $C \rightarrow O$ (**NO**): There might be multiple orders from a single customer. Hence, a customer does not uniquely determine an order in general.
- $C \rightarrow T$ (**NO**): Different orders by the same customer might have different total prices. Therefore, a customer does not uniquely determine the total price.
- $T \rightarrow O$ (**NO**): The total price does not uniquely determine an order as multiple orders might have the same total price.
- $T \rightarrow C$ (**NO**): Similar to the above explanation, different orders with the same total price may belong to different customers, so the total price does not uniquely determine a customer.
- $(O, C) \rightarrow T$ (**YES**): Considering both order and customer, for a given order by a particular customer, there is a unique total price associated.
- $(O, T) \rightarrow C$ (**YES**): If you know both the order and the total price, you can uniquely determine the customer who made that order with that total price.
- $(C, T) \rightarrow O$ (**YES**): For a particular customer with a specific total price, there is a unique order associated.
- $(O, C, T) \rightarrow$ (**YES**): With all three attributes together, i.e., order, customer, and total price, there's a unique combination that identifies a specific record or transaction in the database.

The diagonal cells indicate an attribute's dependency on itself, which is trivial in a functional dependency matrix, hence marked as "–".
All of this is assumed to be TRUE for given specific instance of the table.

Question - 3:

Warehouse_ID	**Product_ID**	**Warehouse_Location**	**Product_Name**	**Product_Category**
1	101	New York	Laptop	Electronics
1	102	New York	Printer	Electronics
2	101	Los Angeles	Laptop	Electronics
2	103	Los Angeles	Smartphone	Electronics

Table 3: Question Table

The second normal form (2NF) requires that a relation be in 1NF and that no partial dependencies exist i.e. no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
In the relation state, since WarehouseID and ProductID form the composite key, every non-candidate attribute must be dependent on (WarehouseID, and ProductID) as a whole, for the second normal form to satisfy.

But from the relation, we can clearly see that Warehouse Location is derived independently from WarehouseID as well as a similar case can be drawn for Product Name and Product Category to be derived from ProductID.

Warehouse_ID	**Warehouse_Location**
1	New York
2	Los Angeles

Table 4: Warehouse Table

Product_ID	**Product_Name**	**Product_Category**
101	Laptop	Electronics
102	Printer	Electronics
103	Smartphone	Electronics

Table 5: Product Table

Warehouse_ID	**Product_ID**
1	101
1	102
2	101
2	103

Table 6: Key Mapping Table

Question - 4:

Given Original Relation $R(A, B, C, D)$ and functional dependencies $AB \rightarrow C, C \rightarrow D, D \rightarrow B$.
Given Decomposition:$R(A, B, C, D) \rightarrow R1(A, B, C) and R2(C, D)$

Test:

- $Att(R1) \cup Att(R2) = Att(R)$ **PASSED**
- $Att(R1) \cap Att(R2) \neq \phi$ **PASSED** (common attribute = {C})
- $Att(R1) \cap Att(R2) \rightarrow Att(R1)$ or $Att(R1) \cap Att(R2) \rightarrow Att(R2)$ **FAILED** as the common attribute {C} does not form a key for either Relations.

Assuming Decomposed Relations $R1(A, B, C)$ and $R2(C, D)$ are joined on C.

The Relational Dependencies $AB \rightarrow C, C \rightarrow D$ are preserved in the join as the participating attributes for these functional dependencies are independently assorted into different relations ($R1$ and $R2$).

However, the functional dependency $D \rightarrow B$ is lost on join (which would lead to the introduction of spurious tuples in joined tables).

Alternative Decomposition: $R(A, B, C, D) \rightarrow R1(A, B, C) and R2(B, C, D)$

- $Att(R1) \cup Att(R2) = Att(R)$ **PASSED**
- $Att(R1) \cap Att(R2) \neq \phi$ **PASSED** (common attribute = {B,C})
- $Att(R1) \cap Att(R2) \rightarrow Att(R1)$ or $Att(R1) \cap Att(R2) \rightarrow Att(R2)$ **PASSED** as the common attribute {B,C} form a key for R2 relation.

Question - 5:

5.1 Introduction

This document outlines the steps for converting the given ER diagram to a relational database schema.

5.2 Steps

5.2.1 Identify Entities

The ER diagram contains the following strong entities:

- **Bank** (Code, Name, Addr)
- **Account** (Balance, Type, Acc No.)
- **Loan** (Loan Number, Amount, Type)
- **Customer** (Name, SSN, Addr, Phone Number)

5.2.2 Create Tables

Create tables for each strong entity:

- **Bank** (Code, Name, Addr)
- **Account** (Balance, Type, Acc No.)
- **Loan** (Loan Number, Amount, Type)
- **Customer** (Name, SSN, Addr, Phone Number)

5.2.3 Identify Relationships

The relationships identified are:

- **Branches** (Bank, Bank-Branch (1:N))
- **Loans** (BankBranch, Loan (1:N))
- **A-C** (Customer, Account (M:N))
- **L-C** (Loan, Customer (M:N))
- **ACCTS** (BankBranch, Account (1:N))

5.2.4 Create Foreign Keys

Introduce foreign keys for relationships:

- **Branches** (Bank references Bank)
- **Loans** (BankBranch references BankBranch, Loan references Loan)
- **A-C** (Customer references Customer, Account references Account)
- **L-C** (Loan references Loan, Customer references Customer)
- **ACCTS** (BankBranch references BankBranch, Account references Account)

5.2.5 Resolve Many-to-Many Relationships

No explicit many-to-many relationships to resolve.

5.3 Handle Weak Entities

The weak entity **Bank Branch** is represented as a table with a composite primary key:

- **BankBranch** (Addr, Branch Number)

5.3.1 Attribute Mapping

Map attributes of entities to appropriate data types:

- **Bank** (Code: INT, Name: VARCHAR, Addr: VARCHAR)
- **Account** (Balance: DECIMAL, Type: VARCHAR, Acc No.: INT)
- **Loan** (Loan Number: INT, Amount: DECIMAL, Type: VARCHAR)
- **Customer** (Name: VARCHAR, SSN: VARCHAR, Addr: VARCHAR, Phone Number: VARCHAR)
- **BankBranch** (Addr: VARCHAR, Branch Number: INT)

5.4 Conclusion

The ER diagram has been successfully converted into a relational database schema.

Stage 1: Initial Tables

```
Table Bank {
  Code INT [PK]
  Name VARCHAR
  Addr VARCHAR
}

Table Account {
  Balance DECIMAL
  Type VARCHAR
  AccNo INT [PK]
}

Table Loan {
  LoanNumber INT [PK]
  Amount DECIMAL
  Type VARCHAR
}

Table Customer {
  Name VARCHAR
  SSN VARCHAR [PK]
  Addr VARCHAR
  PhoneNumber VARCHAR
}

Table BankBranch {
```

```
  Addr VARCHAR
  BranchNumber INT [PK]
}

Ref: "Branches"."Bank" > "Bank"."Code"
Ref: "Loans"."BankBranch" > "BankBranch"."BranchNumber"
Ref: "Loans"."Loan" > "Loan"."LoanNumber"
Ref: "A-C"."Customer" > "Customer"."SSN"
Ref: "A-C"."Account" > "Account"."AccNo"
Ref: "L-C"."Loan" > "Loan"."LoanNumber"
Ref: "L-C"."Customer" > "Customer"."SSN"
Ref: "ACCTS"."BankBranch" > "BankBranch"."BranchNumber"
Ref: "ACCTS"."Account" > "Account"."AccNo"
```

Stage 2: Indexing and Constraints

```
Table Bank {
  Code INT [PK]
  Name VARCHAR
  Addr VARCHAR
  Indexes {
    (Code) [UNIQUE]
  }
}

Table Account {
  Balance DECIMAL
  Type VARCHAR
  AccNo INT [PK]
  Indexes {
    (AccNo) [UNIQUE]
  }
}

Table Loan {
  LoanNumber INT [PK]
  Amount DECIMAL
  Type VARCHAR
  Indexes {
    (LoanNumber) [UNIQUE]
  }
}

Table Customer {
  Name VARCHAR
  SSN VARCHAR [PK]
  Addr VARCHAR
  PhoneNumber VARCHAR
  Indexes {
    (SSN) [UNIQUE]
  }
}

Table BankBranch {
  Addr VARCHAR
```

```
  BranchNumber INT [PK]
  Indexes {
    (BranchNumber) [UNIQUE]
  }
}
```

```
// Final Database Schema

Table Bank {
  Code INT [PK]
  Name VARCHAR
  Addr VARCHAR
  Indexes {
    (Code) [UNIQUE]
  }
}

Table Account {
  Balance DECIMAL
  Type VARCHAR
  AccNo INT [PK]
  Indexes {
    (AccNo) [UNIQUE]
  }
}

Table Loan {
  LoanNumber INT [PK]
  Amount DECIMAL
  Type VARCHAR
  Indexes {
    (LoanNumber) [UNIQUE]
  }
}

Table Customer {
  Name VARCHAR
  SSN VARCHAR [PK]
  Addr VARCHAR
  PhoneNumber VARCHAR
  Indexes {
    (SSN) [UNIQUE]
  }
}

Table BankBranch {
  Addr VARCHAR
  BranchNumber INT [PK]
  Indexes {
    (BranchNumber) [UNIQUE]
  }
}

Table Branches {
  Bank INT [PK]
  BankBranch INT [PK]
  Indexes {
    (Bank, BankBranch) [UNIQUE]
```

```
  }
}

Table Loans {
  BankBranch INT [PK]
  Loan INT [PK]
  Indexes {
    (BankBranch, Loan) [UNIQUE]
  }
}

Table A_C {
  Customer VARCHAR [PK]
  Account INT [PK]
  Indexes {
    (Customer, Account) [UNIQUE]
  }
}

Table L_C {
  Loan INT [PK]
  Customer VARCHAR [PK]
  Indexes {
    (Loan, Customer) [UNIQUE]
  }
}

Table ACCTS {
  BankBranch INT [PK]
  Account INT [PK]
  Indexes {
    (BankBranch, Account) [UNIQUE]
  }
}

Ref: "Branches"."Bank" > "Bank"."Code"
Ref: "Branches"."BankBranch" > "BankBranch"."BranchNumber"
Ref: "Loans"."BankBranch" > "BankBranch"."BranchNumber"
Ref: "Loans"."Loan" > "Loan"."LoanNumber"
Ref: "A_C"."Customer" > "Customer"."SSN"
Ref: "A_C"."Account" > "Account"."AccNo"
Ref: "L_C"."Loan" > "Loan"."LoanNumber"
Ref: "L_C"."Customer" > "Customer"."SSN"
Ref: "ACCTS"."BankBranch" > "BankBranch"."BranchNumber"
Ref: "ACCTS"."Account" > "Account"."AccNo"
```

Final RDML Table:

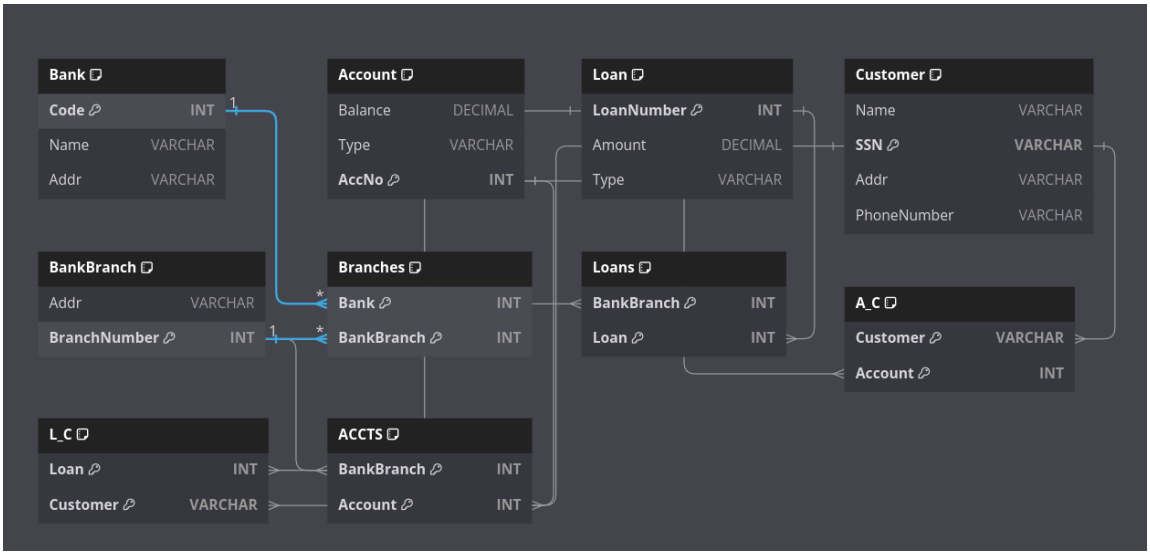

Figure 1: RDML for Given ER diagram

All the parts i.e. mapping strong entity types, weak entity types, and relationships have been already answered while creating the relational table.

Assumptions

1. SSN ID for a given customer can be a alphanumeric strings.
2. Phone number of a customer cannot exceed the limit of INT (i.e. there is a 9 digit phone number).

Hyperlink:

Hyperlink to DBML DBML on dbdiagrams.io.