IE 6700: DATA MANAGEMENT FOR ANALYTICS

USE CASE: MILESTONE #2

Enhancing customer understanding and Service Improvement for a Banking System

---

GROUP NO. 12

GROUP MEMBERS:

MALLIKA GAIKWAD

SHARVARI PRAVIN DESHPANDE

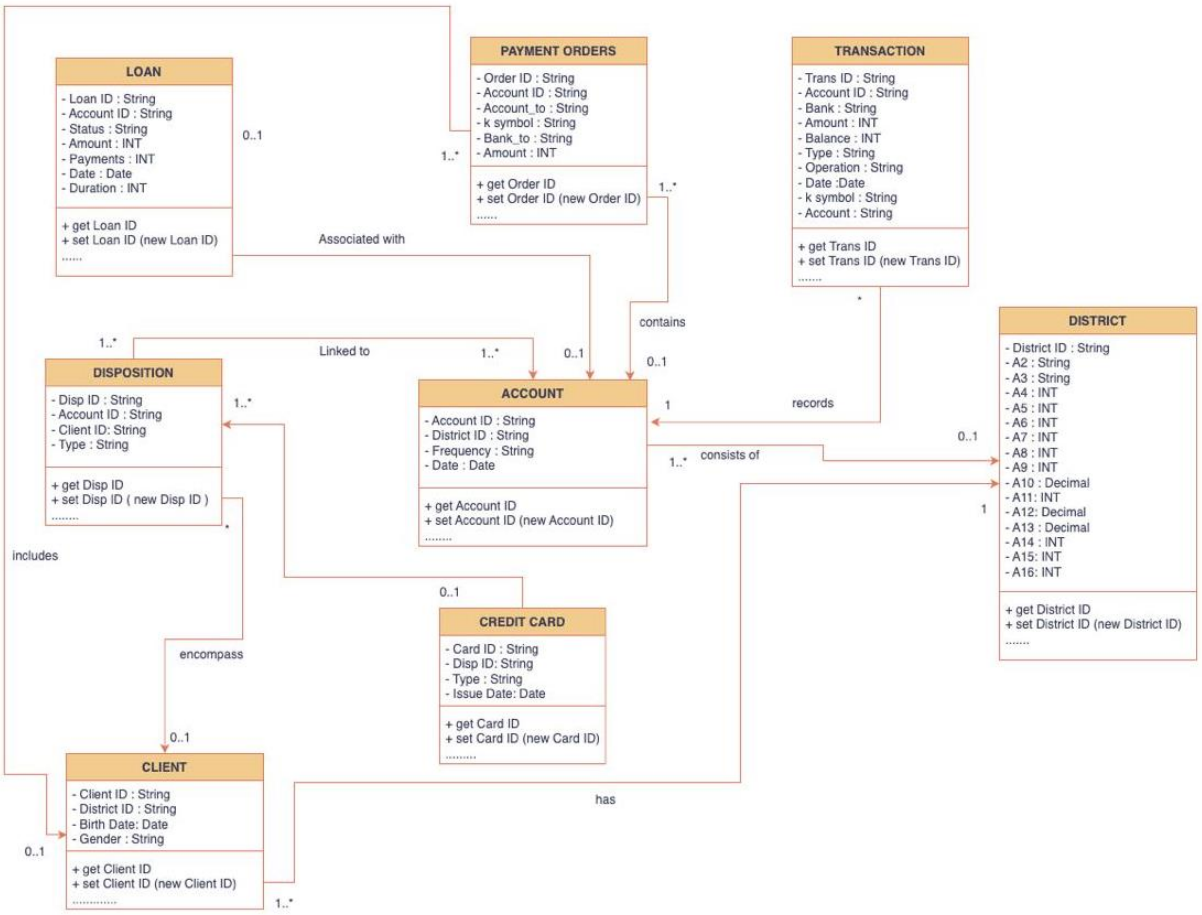
PROBLEM STATEMENT:

Objective:

The primary objective of this project is to empower the banking system with precise insights into customer behavior, preferences, and risk profiles. By utilizing advanced analytical techniques, the bank aims to refine its understanding of clients and formulate specific strategies for service improvement. The bank seeks a robust and well-structured approach to ensure convincing results.

A COMPLETE CONCEPTUAL DATA MODEL USING UML CLASS DIAGRAM INCLUDING SALIENT METHODS

Below is the UML class diagram for the problem statement given.



The above UML class diagram consists of the Class name (entity type), Variables (attribute types) and the methods.

- +: Denotes a public method. It means that the method is accessible from outside the class.
- -: Denotes a private method. It means that the method is only accessible from within the class itself.
- #: Denotes a protected method. It means that the method is accessible within the class itself and by its subclasses.

These notations help to convey the visibility, name, parameters, and return type of the methods in a UML class diagram.

A RELATIONAL MODEL ACCURATELY MAPPED FROM THE CONCEPTUAL MODEL

Step 1 : Mapping Entity types

The first step to map a conceptual data model to a relational model is to map each entity type into a relation. Simple attributes here have been mapped directly. The key attribute types of pf the entity type has been set as the primary key of the relation and is indicated with an underline.

LOAN (LoanID, AccountID, Status, Amount, Payments, Date, Duration)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

PAYMENT_ORDER (OrderID, ksymbol, AccountID, Bank_to, Account_to, Amount)

TRANSACTION (TransID, Bank, Amount, Balance, AccountID, Type, Operation, Date, ksymbol, Account)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

DISPOSITION (DispID, AccountID, ClientID, Type)

CREDIT_CARD (CardID, IssueDate, DispID, Type)

DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)

Step 2 : Mapping Relationship Types

Once we have mapped the entity types, we can continue with the relationship types. The mapping depends upon the degree and cardinalities of the relationships.

Mapping a Binary 1:1 Relationship

Let us consider the 1:1 relationship type. A loan is associated with at most one account (Loan to Account) So there exists a 1:1 relationship between loan and account entity type.

LOAN is existence-dependent on ACCOUNT.

ACCOUNT (AccountID, DistrictID, Frequency, Date)

LOAN (LoanID, AccountID, Status, Amount, Payments, Date, Duration)

So there are two ways to map the relationship type.

OPTION 1 : Add a Foreign key ‘LoanID’ to the ACCOUNT relation, which refers to the primary key ‘LoanID’ in LOAN as follows :

Note : Foreign keys have been depicted in italics

ACCOUNT (AccountID, DistrictID, Frequency, Date, *LoanID*)

LOAN (LoanID, AccountID, Status, Amount, Payments, Date, Duration)

This foreign key can be NULL, since every account is not necessarily associated with a loan.

OPTION 2 :

ACCOUNT (AccountID, DistrictID, Frequency, Date)

LOAN (LoanID, *AccountID*, Status, Amount, Payments, Date, Duration)

In this option, we include a foreign key **AccountID** in the LOAN table, referring to the primary key **AccountID** in the ACCOUNT table. This foreign key should be declared as NOT NULL, since every loan should have exactly one corresponding account.

Now, let's analyze the cardinalities for each option:

Option 1:

- Can an account have zero loans? Yes, this is possible if the LoanID in the ACCOUNT table is NULL.
- Can an account have more than one loan? No, since we're assuming that an account can have at most one loan in a 1:1 relationship.
- Can a loan be associated with zero accounts? Yes, if the AccountID in the LOAN table doesn't match any AccountID in the ACCOUNT table.
- Can a loan be associated with more than one account? No, since we're assuming that a loan can be associated with at most one account in a 1:1 relationship.

Out of the four cardinalities, only two are supported in Option 1.

Option 2:

- Can an account have zero loans? Yes, this is possible if the AccountID in the LOAN table doesn't match any AccountID in the ACCOUNT table.
- Can an account have more than one loan? No, since we're assuming that an account can have at most one loan in a 1:1 relationship.
- Can a loan be associated with zero accounts? No, every loan should be associated with exactly one account in this option.
- Can a loan be associated with more than one account? No, since we're assuming that a loan can be associated with at most one account in a 1:1 relationship.

Out of the four cardinalities, three are supported in Option 2.

Option 2 is preferred as it closely models a 1:1 relationship between ACCOUNT and LOAN.

ACCOUNT (AccountID, DistrictID, Frequency, Date)

LOAN (LoanID, *AccountID*, Status, Amount, Payments, Date, Duration)

In the above mapping, there is some loss of semantics that needs to be considered.

Loss of Nullability Information:
In Option 1, where **LoanID** is added to the **ACCOUNT** table, a **NULL** value in the **LoanID** column could indicate an account without a loan. However, it doesn't provide information about the reason why there is no loan associated. It could be due to the absence of a loan application, a rejected application, or a different reason.

Loss of Loan Status and Duration Information:
Since the **STATUS** and **DURATION** attributes are part of the **LOAN** table, in both options, an account without a loan loses information about potential loan status and duration.

Potential Ambiguity:
In Option 1, if a **LoanID** is present in the **ACCOUNT** table, it might be assumed that there is a corresponding loan. However, this may not always be the case, potentially leading to misinterpretation.

Limited Flexibility in Option 2:
In Option 2, if the same account needs to be associated with multiple loans (which may be rare but possible), the model will not allow it. This limitation could be relevant in certain business scenarios.

Mapping a Binary 1:N Relationship type
Let us consider the 1:N relationship type. A client can have multiple dispositions (Client to Disposition)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

DISPOSITION (DispID, AccountID, ClientID, Type)

We can again explore two options to establish the relationship type in the relational model.

OPTION 1:

CLIENT (ClientID, DistrictID, Birthdate, Gender)

DISPOSITION (DispID, AccountID, *ClientID*, Type)

In this option, we include a foreign key **ClientID** in the **DISPOSITION** table, referring to the primary key **ClientID** in the **CLIENT** table. This foreign key should be declared as NOT NULL to ensure that every disposition is associated with exactly one client.

OPTION 2:

CLIENT (ClientID, DistrictID, Birthdate, Gender, *DispID*)

DISPOSITION (DispID, AccountID, ClientID, Type)

In this option, we include a foreign key **DispID** in the **CLIENT** table, referring to the primary key **DispID** in the **DISPOSITION** table. This foreign key can be NULL since not every client may have a corresponding disposition.

Now, let's analyze the cardinalities for each option:

Option 1:

- Can a client have more than one disposition? Yes, this is the case for clients who have multiple accounts, resulting in multiple dispositions.
- Can we guarantee that every client has at least one disposition? Yes, since the foreign key **ClientID** in the **DISPOSITION** table is declared as NOT NULL, ensuring that every disposition is associated with exactly one client.

Out of the four cardinalities, all four are supported in Option 1.

Option 2:

- Can a client have more than one disposition? Yes, this is the case for clients who have multiple accounts, resulting in multiple dispositions.
- Can we guarantee that every client has at least one disposition? No, this is not guaranteed in Option 2. If a client has no associated accounts, the **DispID** in the **CLIENT** table will be NULL.

Out of the four cardinalities, three are supported in Option 2.

Option 1 is preferred as it accurately models the 1:N relationship between **CLIENT** and **DISPOSITION**, ensuring that every client has at least one disposition. It also supports the possibility of a client having multiple dispositions.

CLIENT (ClientID, DistrictID, Birthdate, Gender)

DISPOSITION (DispID, AccountID, *ClientID*, Type)

Below can be the possible loss of semantics:
Loss of Explicit NULL Information:
While Option 1 enforces that every disposition must be associated with a client, it doesn't explicitly represent cases where a disposition might be temporarily without a client (for instance, when a new client is being registered).

Potential Inconsistencies in Option 2:

In Option 2, if the same disposition needs to be associated with multiple clients (which may be rare but possible), the model will not allow it. This limitation could be relevant in certain business scenarios.

Implicit Assumptions:
Both options assume that the relationship between clients and dispositions is strictly one-to-many. If the business logic changes in the future to allow for more complex relationships, this model may not accommodate it.

Loss of Flexibility in Option 2:
In Option 2, if a disposition needs to be temporarily without a client (perhaps during a transition period), this option will not allow for it.

Similarly, we can create relational mapping for the entities below as well.

A client can have multiple payment orders (Client to payment order)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

PAYMENT_ORDER (OrderID, ksymbol, AccountID, Bank_to, Account_to, Amount)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

PAYMENT_ORDER (OrderID, *ClientID*, ksymbol, AccountID, Bank_to, Account_to, Amount)

An account can have multiple transactions (Account to transaction)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

TRANSACTION (TransID, Bank, Amount, Balance, AccountID, Type, Operation, Date, ksymbol, Account)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

TRANSACTION (TransID, Bank, Amount, Balance, *AccountID*, Type, Operation, Date, ksymbol, Account)

An account can have multiple payment orders (Account to payment order)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

PAYMENT_ORDER (OrderID, ksymbol, AccountID, Bank_to, Account_to, Amount)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

PAYMENT_ORDER (OrderID, ksymbol, *AccountID*, Bank_to, Account_to, Amount)

A district can have multiple accounts (District to account)

DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)

ACCOUNT (AccountID, *DistrictID*, Frequency, Date)

Step 3: Mapping a binary M:N Relationship type

M:N relationship types can be mapped by introducing a new relationship type as R. The primary key of R is a combination of foreign keys referring to the primary keys of the relations corresponding to the participating entity types. The attribute types of the M:N relationship type can also be added to R.

A client can have multiple accounts and an account can have multiple clients (Client to account through disposition)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

To create a M:N (many-to-many) mapping between the **CLIENT** and **ACCOUNT** relationships, we need to introduce a new relation to map this M:N relationship. Let's call this new relation INCLUDES.

INCLUDES (ClientID, AccountID, DistrictID, Birthdate, Gender, Frequency, Date)

- ClientID and AccountID are primary keys referencing to both the tables and hence NOT NULL

Hence, the final relations that we get are:

LOAN (LoanID, AccountID, Status, Amount, Payments, Date, Duration)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

PAYMENT_ORDER (OrderID, ksymbol, AccountID, Bank_to, Account_to, Amount)

TRANSACTION (TransID, Bank, Amount, Balance, AccountID, Type, Operation, Date, ksymbol, Account)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

DISPOSITION (DispID, AccountID, ClientID, Type)

CREDIT_CARD (CardID, IssueDate, DispID, Type)

DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

LOAN (LoanID, *AccountID*, Status, Amount, Payments, Date, Duration)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

DISPOSITION (DispID, AccountID, *ClientID*, Type)

CLIENT (ClientID, DistrictID, Birthdate, Gender)

PAYMENT_ORDER (OrderID, *ClientID*, ksymbol, AccountID, Bank_to, Account_to, Amount)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

TRANSACTION (TransID, Bank, Amount, Balance, *AccountID*, Type, Operation, Date, ksymbol, Account)

ACCOUNT (AccountID, DistrictID, Frequency, Date)

PAYMENT_ORDER (OrderID, ksymbol, *AccountID*, Bank_to, Account_to, Amount)

DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)

ACCOUNT (AccountID, *DistrictID*, Frequency, Date)

INCLUDES (ClientID, AccountID, DistrictID, Birthdate, Gender, Frequency, Date)

RELATIONAL MODEL MUST BE NORMALIZED UPTO (ATLEAST) 3.5 NF FORM

To normalize the above relations, let's consider each relation seperately

1. LOAN (LoanID, AccountID, Status, Amount, Payments, Date, Duration)

The above relation is already in the first normal form (1NF) as the attributes are atomic and not multi-valued.

The second normal form (2 NF) states that the relation should be in 1NF and ensures that no partial dependencies exist; all non-key attributes are fully functionally dependent on the primary key.

Based on the attributes, we can identify the below functional dependencies :

- LoanID → {AccountID, Status, Amount, Payments, Date, Duration}
- AccountID → {Status, Amount, Payments, Date, Duration}

Since, there are two non-trivial functional dependencies, we'll break the 'LOAN' into two smaller relations.

RELATION 1: LOAN_PARTIAL (LoanID, *AccountID*, Status)

RELATION 2: LOAN_DETAILS (AccountID, Amount, Payments, Date, Duration)

The third normal form (3NF) states that, it should be in 2NF and ensures that there are no transitive dependencies between non-key attributes and the primary key.

There are no transitive dependencies in LOAN_PARTIAL and LOAN_DETAILS.

Both relations are in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

Since we've already reached BCNF and there are no join dependencies or multivalued dependencies, the relations are in 3.5NF.

FINAL RELATIONS :

LOAN_PARTIAL (LoanID, *AccountID*, Status)

LOAN_DETAILS (AccountID, Amount, Payments, Date, Duration)

2. ACCOUNT (AccountID, DistrictID, Frequency, Date)

1NF

The relation is already in 1NF because it contains only atomic values.

2NF

The relation is already in 2NF because there are no partial dependencies.

3NF

The relation is already in 3NF because there are no transitive dependencies.

BCNF

The relation is in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

FINAL RELATION :

ACCOUNT (AccountID, DistrictID, Frequency, Date)

3.PAYMENT_ORDER (OrderID, ksymbol, AccountID, Bank_to, Account_to, Amount)

The above relation is already in the first normal form (1NF) as the attributes are atomic and not multi-valued.

The second normal form (2 NF) states that the relation should be in 1NF and ensures that no partial dependencies exist; all non-key attributes are fully functionally dependent on the primary key.

Following functional dependencies can be identified :

- OrderID → {ksymbol, AccountID, Bank_to, Account_to, Amount}
- AccountID → {ksymbol, OrderID, Bank_to, Account_to, Amount}

Since there are two non-trivial functional dependencies, we'll break the **PAYMENT_ORDER** relation into two smaller relations:
RELATION 1: PAYMENT_PARTIAL (OrderID, ksymbol, *AccountID*)
RELATION 2: PAYMENT_DETAILS (AccountID, Bank_to, Account_to, Amount)

The third normal form (3NF) states that, it should be in 2NF and ensures that there are no transitive dependencies between non-key attributes and the primary key.

There are no transitive dependencies in PAYMENT_PARTIAL and PAYMENT_DETAILS.

Both relations are in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

Since we've already reached BCNF and there are no join dependencies or multivalued dependencies, the relations are in 3.5NF.

FINAL RELATIONS :
PAYMENT_PARTIAL (OrderID, ksymbol, *AccountID*)
PAYMENT_DETAILS (AccountID, Bank_to, Account_to, Amount)

4.TRANSACTION (TransID, Bank, Amount, Balance, AccountID, Type, Operation, Date, ksymbol, Account)

The above relation is already in the first normal form (1NF) as the attributes are atomic and not multi-valued.

The second normal form (2 NF) states that the relation should be in 1NF and ensures that no partial dependencies exist; all non-key attributes are fully functionally dependent on the primary key.

Following functional dependencies can be identified :

- TransID → {Bank, Amount, Balance, AccountID, Type, Operation, Date, ksymbol, Account}
- AccountID → {Bank, Amount, Balance, Type, Operation, Date, ksymbol, Account}

Since there are two non-trivial functional dependencies, we'll break the **TRANSACTION** relation into two smaller relations:

RELATION 1: TRANSACTION_PARTIAL (TransID, Bank, Amount, Balance, *AccountID*)

RELATION 2: TRANSACTION_DETAILS (AccountID, Type, Operation, Date, ksymbol, Account)

The third normal form (3NF) states that, it should be in 2NF and ensures that there are no transitive dependencies between non-key attributes and the primary key.

There are no transitive dependencies in PAYMENT_PARTIAL and PAYMENT_DETAILS.

Both relations are in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

Since we've already reached BCNF and there are no join dependencies or multivalued dependencies, the relations are in 3.5NF.

FINAL RELATION :

TRANSACTION_PARTIAL (TransID, Bank, Amount, Balance, *AccountID*)

TRANSACTION_DETAILS (AccountID, Type, Operation, Date, ksymbol, Account)

5.CLIENT (ClientID, DistrictID, Birthdate, Gender)

1NF

The relation is already in 1NF because it contains only atomic values.

2NF

The relation is already in 2NF because there are no partial dependencies.

3NF

The relation is already in 3NF because there are no transitive dependencies.

BCNF

The relation is in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

FINAL RELATION:

CLIENT (ClientID, DistrictID, Birthdate, Gender)

6.DISPOSITION (DispID, AccountID, ClientID, Type)

The above relation is already in the first normal form (1NF) as the attributes are atomic and not multi-valued.

The second normal form (2 NF) states that the relation should be in 1NF and ensures that no partial dependencies exist; all non-key attributes are fully functionally dependent on the primary key.

Following functional dependencies can be identified :

- DispID → {AccountID, ClientID, Type}
- (AccountID, ClientID) → Type

Since there are two non-trivial functional dependencies, we'll break the **DISPOSITION** relation into two smaller relations:

RELATION 1: DISP_PARTIAL (DispID, *AccountID*, ClientID)

RELATION 2: DISP_DETAILS (AccountID, ClientID, Type)

The third normal form (3NF) states that, it should be in 2NF and ensures that there are no transitive dependencies between non-key attributes and the primary key.

There are no transitive dependencies in PAYMENT_PARTIAL and PAYMENT_DETAILS.

Both relations are in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

Since we've already reached BCNF and there are no join dependencies or multivalued dependencies, the relations are in 3.5NF.

FINAL RELATION:

DISP_PARTIAL (DispID, *AccountID*, ClientID)

DISP_DETAILS (AccountID, ClientID, Type)

7.CREDIT_CARD (CardID, IssueDate, DispID, Type)

The above relation is already in the first normal form (1NF) as the attributes are atomic and not multi-valued.

The second normal form (2 NF) states that the relation should be in 1NF and ensures that no partial dependencies exist; all non-key attributes are fully functionally dependent on the primary key.

Following functional dependencies can be identified :

- CardID → {IssueDate, DispID, Type}
- DispID → {CardID, IssueDate, Type}

Since there are two non-trivial functional dependencies, we'll break the **CREDIT_CARD** relation into two smaller relations:

RELATION 1: CARD_PARTIAL (CardID, IssueDate, *DispID*)

RELATION 2: CARD_DETAILS (DispID, Type)

The third normal form (3NF) states that, it should be in 2NF and ensures that there are no transitive dependencies between non-key attributes and the primary key.

There are no transitive dependencies in PAYMENT_PARTIAL and PAYMENT_DETAILS.

Both relations are in BCNF because, for every non-trivial functional dependency X → Y, X is a super key.

Since we've already reached BCNF and there are no join dependencies or multivalued dependencies, the relations are in 3.5NF.

FINAL RELATION:

CARD_PARTIAL (CardID, IssueDate, *DispID*)

CARD_DETAILS (DispID, Type)

8.DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)

1NF

The relation is already in 1NF because it contains only atomic values.

2NF

The relation is already in 2NF because there are no partial dependencies.

3NF

The relation is already in 3NF because there are no transitive dependencies.

BCNF

The relation is in BCNF because, for every non-trivial functional dependency $X \rightarrow Y$, X is a super key.

FINAL RELATION:

DISTRICT (DistrictID, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16)
