NORMALIZATION:

Normal Forms

1st Normal Form (1NF)

To ensure that all attributes in a table are atomic, we take the following steps:

- **Atomic Attributes**: All attributes must contain indivisible values. If an attribute is composite (having multiple components), each component should be included in the table
- **Multivalued Attributes**: For attributes that can have multiple values, a separate table is created to store these values, with a reference to the main table.
- **Uniqueness**: Each attribute in a table must have a unique name, and there should be no repeated tuples (rows).

By ensuring atomicity, handling multivalued attributes properly, and maintaining uniqueness, we achieve 1NF.

2nd Normal Form (2NF)

To achieve 2NF, we must eliminate any partial dependencies:

- **Partial Dependency**: A relation is in 2NF if no non-prime attribute is functionally dependent on a proper subset of any candidate key. In our case, all relations are already in 1NF, and since there are no partial dependencies identified among non-prime attributes, we can conclude that the relational schema is in 2NF.

3rd Normal Form (3NF)

To ensure that our schema is in 3NF, we need to eliminate transitive dependencies:

- **Transitive Dependency**: A relation is in 3NF if no non-prime attribute is transitively dependent on a candidate key.

To convert this into 3NF, we create a separate relation (table) and primary key and `some` additional attributes. The original `attribute` tables will then reference this new table attribute.

Thus, after addressing transitive dependencies, the relational schema is now in 3NF.

Boyce-Codd Normal Form (BCNF)

To check for BCNF, we ensure that:

- For every functional dependency in the schema, the left-hand side must be a superkey. After converting the schema into 3NF, we find that there are no functional dependencies where the left-hand side is not a superkey. Hence, we conclude that the relational schema is in BCNF.

⇒ Normalization is the process of organizing the attributes and relations of a database to minimize redundancy and dependency. It typically involves dividing large tables into smaller ones and defining relationships between them.

Here's how to normalize each of the tables based on their functional dependencies, ensuring they meet the criteria for at least the First (1NF), Second (2NF), and Third Normal Forms (3NF):

IMPLEMENTATION ON OUR FUNCTIONAL DEPENDENCIES

==

1. Users Table

Original Functional Dependencies:

- UserID → PhoneNumber, Username, ProfilePicture, Status, LastSeen

Normalization Steps:

- **1NF**: Ensure all attributes have atomic values. All attributes are already atomic.
- **2NF**: There are no partial dependencies since `UserID` is the primary key.
- **3NF**: There are no transitive dependencies. All attributes depend only on `UserID`.

Normalized Table:

- **Users**:
  - `UserID` (PK)
  - `PhoneNumber`
  - `Username`
  - `ProfilePicture`
  - `Status`
  - `LastSeen`

2. Contacts Table

Original Functional Dependencies:

- ContactID → UserID, ContactUserID, Nickname, Blocked
- UserID, ContactUserID → Nickname, Blocked

Normalization Steps:

- **1NF**: Ensure all attributes are atomic. All attributes are already atomic.
- **2NF**: No partial dependencies exist, as `ContactID` is the only primary key.
- **3NF**: There is a transitive dependency since `Nickname` and `Blocked` depend on both `UserID` and `ContactUserID`.

Normalized Table:

- **Contacts**:
  - `ContactID` (PK)
  - `UserID` (FK)
  - `ContactUserID` (FK)
- **ContactDetails**:
  - `UserID` (FK)
  - `ContactUserID` (FK)
  - `Nickname`
  - `Blocked`

3. Messages Table

Original Functional Dependencies:

- MessageID → SenderID, ReceiverID, Content, MediaURL, Timestamp, Status
- SenderID, ReceiverID, Timestamp → Content, MediaURL, Status

Normalization Steps:

- **1NF**: All attributes have atomic values.
- **2NF**: No partial dependencies exist.
- **3NF**: There is a transitive dependency since `Content`, `MediaURL`, and `Status` depend on `SenderID`, `ReceiverID`, and `Timestamp`.

Normalized Table:

- **Messages**:
  - `MessageID` (PK)

    - `SenderID` (FK)
    - `ReceiverID` (FK)
    - `Timestamp`
- **MessageDetails**:
    - `MessageID` (FK)
    - `Content`
    - `MediaURL`
    - `Status`

4. Groups Table

Original Functional Dependencies:

- GroupID → GroupName, GroupPicture, AdminID, CreationDate

Normalization Steps:

- **1NF**: All attributes are atomic.
- **2NF**: No partial dependencies exist.
- **3NF**: No transitive dependencies.

Normalized Table:

- **Groups**:
    - `GroupID` (PK)
    - `GroupName`
    - `GroupPicture`
    - `AdminID` (FK)
    - `CreationDate`

5. GroupMembers Table

Original Functional Dependencies:

- GroupMemberID → GroupID, UserID, JoinDate, Role
- GroupID, UserID → JoinDate, Role

Normalization Steps:

- **1NF**: All attributes are atomic.
- **2NF**: No partial dependencies exist.
- **3NF**: There are no transitive dependencies.

Normalized Table:

- **GroupMembers**:
    - `GroupMemberID` (PK)
    - `GroupID` (FK)
    - `UserID` (FK)
    - `JoinDate`
    - `Role`

6. Statuses Table

Original Functional Dependencies:

- StatusID → UserID, Content, MediaURL, ExpirationTime
- UserID, ExpirationTime → Content, MediaURL

Normalization Steps:

- **1NF**: All attributes are atomic.
- **2NF**: No partial dependencies exist.
- **3NF**: There is a transitive dependency as `Content` and `MediaURL` depend on both `UserID` and `ExpirationTime`.

Normalized Table:

- **Statuses**:
    - `StatusID` (PK)
    - `UserID` (FK)
    - `ExpirationTime`
- **StatusDetails**:
    - `StatusID` (FK)
    - `Content`
    - `MediaURL`

7. Calls Table

Original Functional Dependencies:

- CallID → CallerID, ReceiverID, CallType, CallStartTime, CallEndTime, CallDuration
- CallerID, ReceiverID, CallStartTime → CallType, CallEndTime, CallDuration

Normalization Steps:

- **1NF**: All attributes are atomic.
- **2NF**: No partial dependencies exist.

- **3NF**: There are transitive dependencies since `CallType`, `CallEndTime`, and `CallDuration` depend on `CallerID`, `ReceiverID`, and `CallStartTime`.

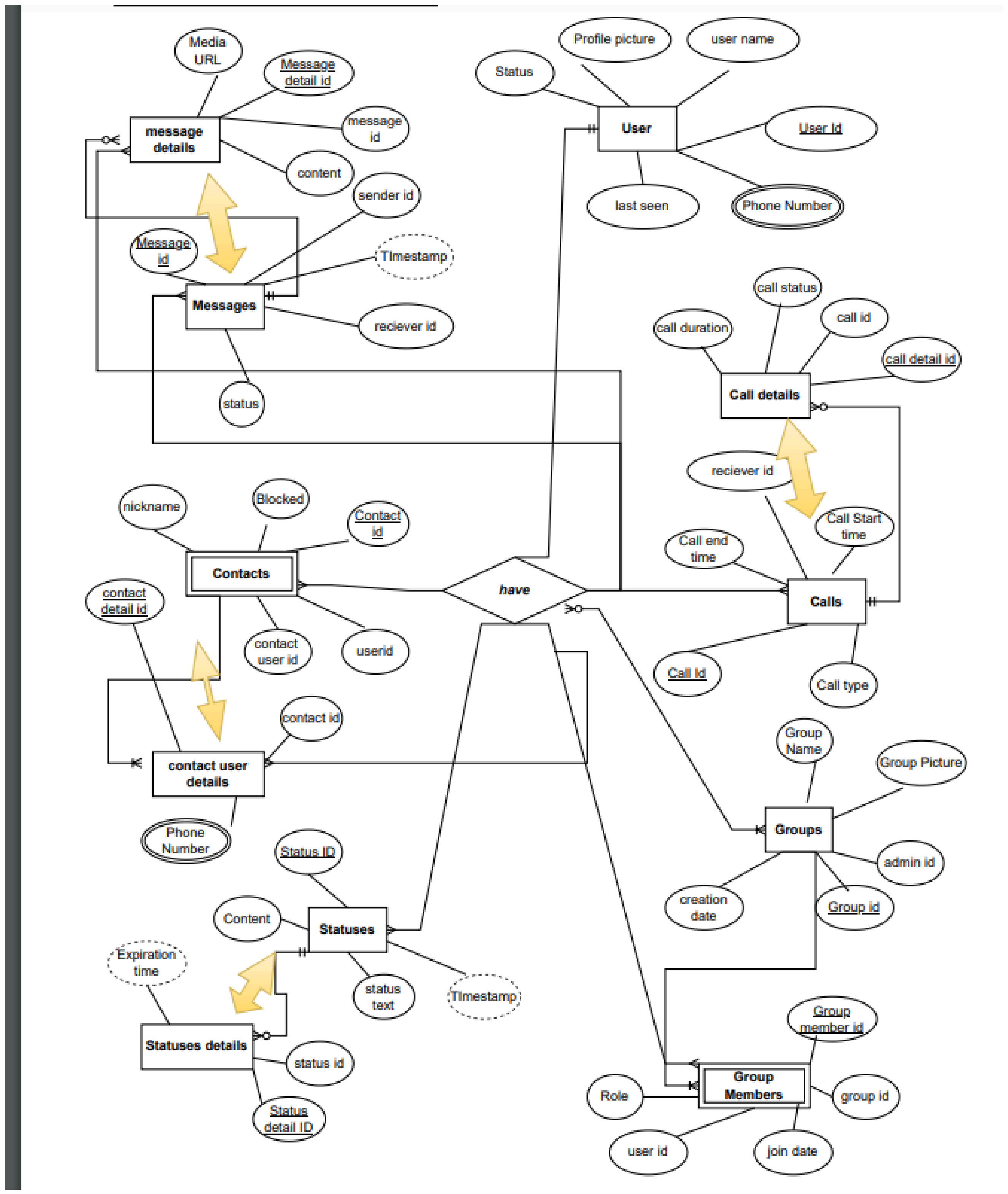
Normalized Table:

- **Calls**:
    - `CallID` (PK)
    - `CallerID` (FK)
    - `ReceiverID` (FK)
    - `CallStartTime`
- **CallDetails**:
    - `CallID` (FK)
    - `CallType`
    - `CallEndTime`
    - `CallDuration`

Summary of Normalization

1. **Users**:
    - Remains as is (1NF, 2NF, 3NF).
2. **Contacts**:
    - Split into `Contacts` and `ContactDetails` to eliminate transitive dependency.
3. **Messages**:
    - Split into `Messages` and `MessageDetails` to eliminate transitive dependency.
4. **Groups**:
    - Remains as is (1NF, 2NF, 3NF).
5. **GroupMembers**:
    - Remains as is (1NF, 2NF, 3NF).
6. **Statuses**:
    - Split into `Statuses` and `StatusDetails` to eliminate transitive dependency.
7. **Calls**:
    - Split into `Calls` and `CallDetails` to eliminate transitive dependency.

<u>ER DIAGRAM AFTER NORMALIZATION:</u>



EXPLAINATION OF NORMALIZATION:

1. **Users ↔ Contacts**

- Relation: A user can have multiple contacts (one-to-many).

- Foreign Key: `UserID` in Contacts refers to `UserID` in Users.
- Foreign Key: `ContactUserID` in Contacts refers to `UserID` in Users (self-referencing for contact lists).

2. **Contacts ↔ ContactDetails**

- Relation: A contact can have multiple phone numbers (one-to-many).
- Foreign Key: `ContactID` in ContactDetails refers to `ContactID` in Contacts.

3. **Users ↔ Messages**

- Relation: A user can send and receive multiple messages (one-to-many for both sender and receiver).
- Foreign Key: `SenderID` and `ReceiverID` in Messages refer to `UserID` in Users.

4. **Messages ↔ MessageDetails**

- Relation: A message can have multiple parts or details (e.g., text, media) (one-to-one or one-to-many depending on design).
- Foreign Key: `MessageID` in MessageDetails refers to `MessageID` in Messages.

5. **Groups ↔ Users**

- Relation: A user can be part of multiple groups, and a group can have multiple users (many-to-many).
- Foreign Key (in intermediate table GroupMembers): `UserID` in GroupMembers refers to `UserID` in Users.
- Foreign Key (in intermediate table GroupMembers): `GroupID` in GroupMembers refers to `GroupID` in Groups.

6. **Groups ↔ GroupMembers**

- Relation: A group can have multiple members (one-to-many).
- Foreign Key: `GroupID` in GroupMembers refers to `GroupID` in Groups.

7. **Users ↔ Statuses**

- Relation: A user can create multiple statuses (one-to-many).
- Foreign Key: `UserID` in Statuses refers to `UserID` in Users.

8. **Statuses ↔ StatusDetails**

- Relation: A status can have additional details like expiration time (one-to-one or one-to-many).
- Foreign Key: `StatusID` in StatusDetails refers to `StatusID` in Statuses.

9. **Users ↔ Calls**

- Relation: A user can initiate or receive multiple calls (one-to-many for both caller and receiver).
- Foreign Key: `CallerID` and `ReceiverID` in Calls refer to `UserID` in Users.

10. **Calls ↔ CallDetails**

- Relation: A call can have additional details like duration, status (one-to-one or one-to-many).
- Foreign Key: `CallID` in CallDetails refers to `CallID` in Calls.

Conclusion

The normalization process enhances data integrity and reduces redundancy by ensuring that each attribute in a table depends only on the primary key. This helps maintain a well-structured database that can be efficiently queried and managed.