

Alpha Release 2

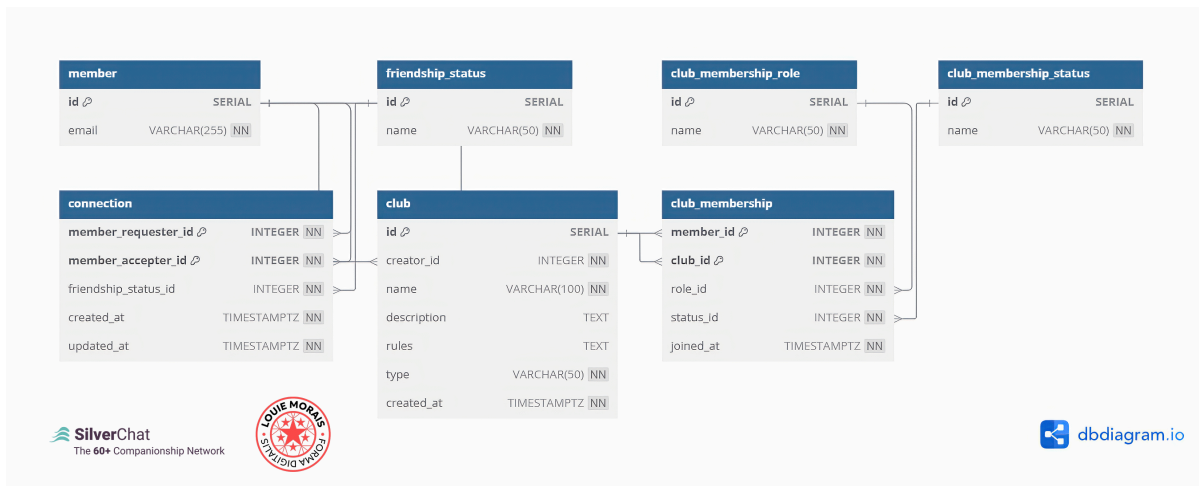
Database Schema 2: Member Networking

- Database Schema 2: Member Networking
 - 1. Member Networking Tables
 - 1.1. Lookup Tables
 - 1.2. Core Data Tables
 - 2. SQL Representation & Implementation Notes
 - 2.1. SQL Syntax
 - 2.2. DBML Syntax
 - 3. SilverChat Project Documentation
 - 4. External Sources
 - 4.1. Database Design for Social Networks (Schemas/Diagrams/ERMs/How Tos)
 - 4.2. Database GitHub Repos
 - 4.3. Database Infrastructure for Social Networks
 - 4.4. Database Design Software
 - 4.5. Facebook Reference

1. Member Networking Tables

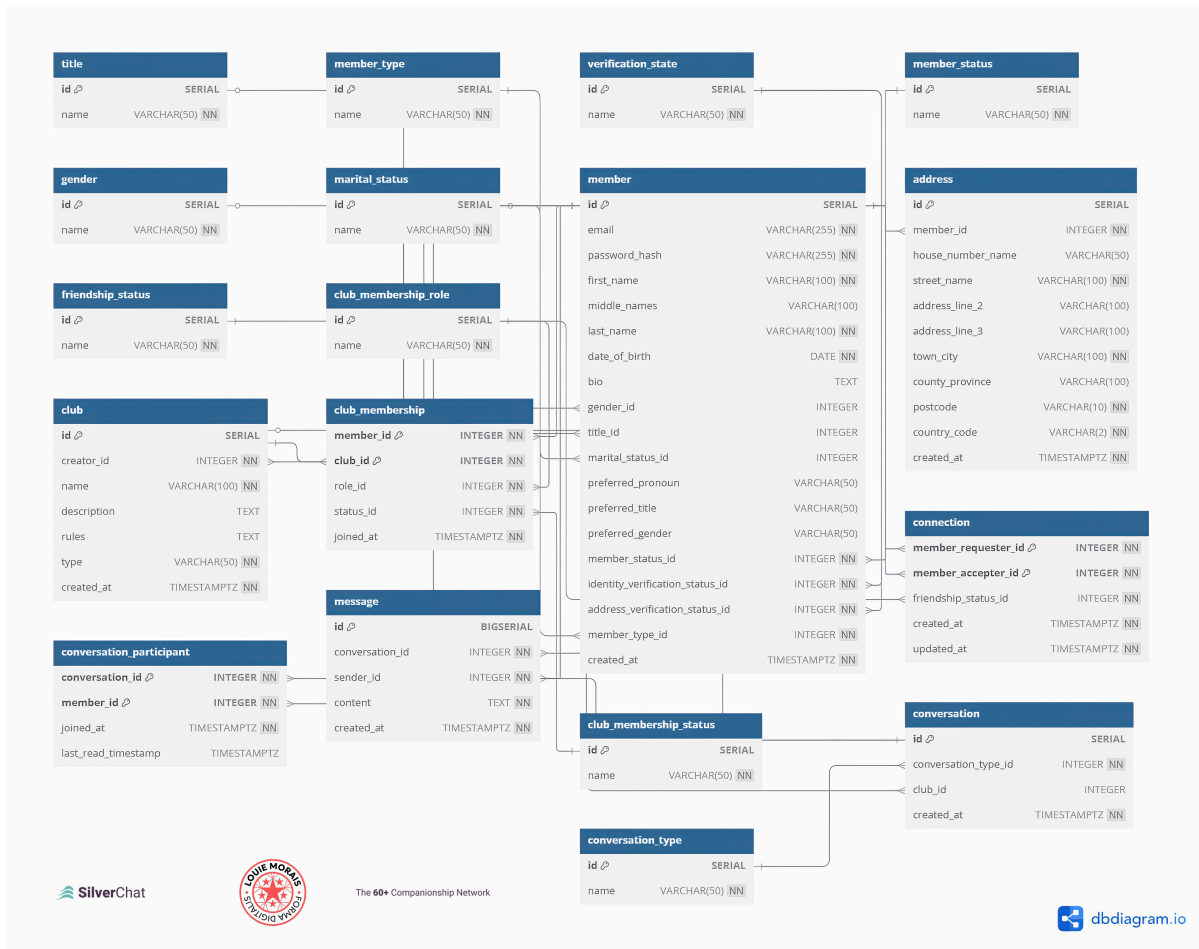
SilverChat - Networking Feature Database Schema (Target: Alpha Release 2)

This schema defines the tables required for member-to-member connections and interest clubs, building upon the core member profile schema established in Alpha 1.



[View the live diagram on dbdiagram.io](#)

Final Database Design - Alpha Releases 1, 2 & 3:



[View the live diagram on dbdiagram.io](#)

1.1. Lookup Tables

Table: `friendship_status` (Lookup for connection states)

Column Name	Data Type	Constraints	Description	Example Values
<code>id</code>	<code>SERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier for friendship status.	1, 2, 3, 4
<code>name</code>	<code>VARCHAR(50)</code>	<code>UNIQUE NOT NULL</code>	Name of the friendship status option.	'Invited', 'Connected', 'Blocked', 'Restrained' <i>(Note: 'Restrained' implies internal/safeguarding block)</i>

Table: `club_membership_role` (Lookup for roles within a club)

Column Name	Data Type	Constraints	Description	Example Values
<code>id</code>	<code>SERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier for club role.	1, 2, 3
<code>name</code>	<code>VARCHAR(50)</code>	<code>UNIQUE NOT NULL</code>	Name of the role within a club.	'Creator', 'Admin', 'Member'

Table: `club_membership_status` (Lookup for member status within a club)

Column Name	Data Type	Constraints	Description	Example Values
<code>id</code>	<code>SERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier for club status.	1, 2, 3, 4
<code>name</code>	<code>VARCHAR(50)</code>	<code>UNIQUE NOT NULL</code>	Name of the status within a club.	'Invited', 'Active', 'Suspended', 'Banned'

1.2. Core Data Tables

(These tables depend on the `member` table defined in Schema 1: Member Account & Profile)

Table: `connection` (Junction table for member-to-member connections)

Column Name	Data Type	Constraints	Description
<code>member_requester_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to the member initiating the connection request.
<code>member_accepter_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to the member receiving/accepting the request.
<code>friendship_status_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key indicating the current status of the connection.
<code>created_at</code>	<code>TIMESTAMPTZ</code>	<code>NOT NULL DEFAULT NOW()</code>	Timestamp when the connection request was initiated.
<code>updated_at</code>	<code>TIMESTAMPTZ</code>	<code>NOT NULL DEFAULT NOW()</code>	Timestamp when the connection status was last updated (e.g., accepted).
		<code>PRIMARY KEY</code> <code>(member_requester_id, member_accepter_id)</code>	Composite primary key ensures uniqueness of a connection between two members.
		<code>CHECK</code> <code>(member_requester_id <> member_accepter_id)</code>	Prevents a member from connecting to themselves.

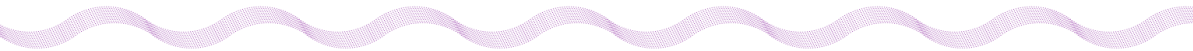
Table: `c1ub` (Defines interest clubs)

Column Name	Data Type	Constraints	Description
<code>id</code>	<code>SERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier for the club.
<code>creator_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to the member who created the club.
<code>name</code>	<code>VARCHAR(100)</code>	<code>UNIQUE NOT NULL</code>	Unique name of the club.
<code>description</code>	<code>TEXT</code>		Description of the club (Optional).
<code>rules</code>	<code>TEXT</code>		Rules of the club (Optional).
<code>type</code>	<code>VARCHAR(50)</code>	<code>NOT NULL</code>	Type of club (e.g., 'Public', 'Private', 'Hidden').

Column Name	Data Type	Constraints	Description
<code>created_at</code>	<code>TIMESTAMPTZ</code>	<code>NOT NULL DEFAULT NOW()</code>	Timestamp when the club was created.

Table: `club_membership` (Junction table linking members to clubs)

Column Name	Data Type	Constraints	Description
<code>member_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to the member joining the club.
<code>club_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to the club being joined.
<code>role_id</code>	<code>INTEGER</code>	<code>NOT NULL DEFAULT 3</code>	Foreign key indicating the member's role (Default: 'Member').
<code>status_id</code>	<code>INTEGER</code>	<code>NOT NULL DEFAULT 2</code>	Foreign key indicating the member's status in the club (Default: 'Active').
<code>joined_at</code>	<code>TIMESTAMPTZ</code>	<code>NOT NULL DEFAULT NOW()</code>	Timestamp when the member joined the club (or was invited).
		<code>PRIMARY KEY (member_id, club_id)</code>	Composite primary key ensures a member has only one entry per club.



2. SQL Representation & Implementation Notes

The following SQL `CREATE TABLE` statements represent the database structure defined above. These are provided for illustrative and cross-referencing purposes. Actual implementation will use Knex.js migrations. The `member` table definition is included below **for context and validation purposes only**; it is part of Schema 1 and should not be re-created.

2.1. SQL Syntax

```
-- Illustrative SQL for Schema 2 (Alpha Release 2) - Actual implementation via Knex
Migrations

-- ===== --
-- START: Prerequisite Tables (Defined in Schema 1 - DO NOT RE-RUN) --
-- Included for context/validation only. --

-- Prerequisite: member Table (Simplified for context)
CREATE TABLE member (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL
  -- Other columns from Schema 1 assumed to exist...
);

-- END: Prerequisite Tables --
-- ===== --

-- START: Networking Feature Tables (Target: Alpha Release 2) --

-- Lookup Tables for Networking
CREATE TABLE friendship_status (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE club_membership_role (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE club_membership_status (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50) UNIQUE NOT NULL
);

-- Core Networking Tables
CREATE TABLE connection (
  member_requester_id INTEGER NOT NULL, -- FK defined below
  member_accepter_id INTEGER NOT NULL, -- FK defined below
  friendship_status_id INTEGER NOT NULL, -- FK defined below
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

  -- Primary Key Constraint
  PRIMARY KEY (member_requester_id, member_accepter_id),

  -- Check Constraint
  CHECK (member_requester_id <> member_accepter_id),

  -- Foreign Key Constraints
  FOREIGN KEY (member_requester_id) REFERENCES member(id) ON DELETE CASCADE, -- Refers
to prerequisite member table
  FOREIGN KEY (member_accepter_id) REFERENCES member(id) ON DELETE CASCADE, -- Refers
to prerequisite member table
  FOREIGN KEY (friendship_status_id) REFERENCES friendship_status(id)
);
```

```

-- Note: Need to add trigger for 'updated_at' on connection table if auto-update is
desired

CREATE TABLE club (
    id SERIAL PRIMARY KEY,
    creator_id INTEGER NOT NULL, -- FK defined below
    name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    rules TEXT,
    type VARCHAR(50) NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Foreign Key Constraint
    FOREIGN KEY (creator_id) REFERENCES member(id) -- Refers to prerequisite member
table
);

CREATE TABLE club_membership (
    member_id INTEGER NOT NULL, -- FK defined below
    club_id INTEGER NOT NULL, -- FK defined below
    role_id INTEGER NOT NULL DEFAULT 3, -- FK defined below, Default: Member
    status_id INTEGER NOT NULL DEFAULT 2, -- FK defined below, Default: Active
    joined_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Primary Key Constraint
    PRIMARY KEY (member_id, club_id),

    -- Foreign Key Constraints
    FOREIGN KEY (member_id) REFERENCES member(id) ON DELETE CASCADE, -- Refers to
prerequisite member table
    FOREIGN KEY (club_id) REFERENCES club(id) ON DELETE CASCADE,
    FOREIGN KEY (role_id) REFERENCES club_membership_role(id),
    FOREIGN KEY (status_id) REFERENCES club_membership_status(id)
);

-- END: Networking Feature Tables
--

```

2.2. DBML Syntax

```

// SilverChat - Schema 2: Member Networking (Alpha Release 2)
// DBML Schema Definition

// =====
// START: Prerequisite Tables (Defined in Schema 1)
// Included for context/validation only.

Table member { // PREREQUISITE - DO NOT RE-CREATE
    id SERIAL [pk]
    email VARCHAR(255) [unique, not null]
    // Other columns from Schema 1 assumed to exist...
}

// END: Prerequisite Tables
// =====

```

```

// START: Networking Feature Tables (Target: Alpha Release 2)

// --- Lookup Tables ---

Table friendship_status {
    id SERIAL [pk]
    name VARCHAR(50) [unique, not null, note: 'Connection states: Invited, Connected,
Blocked, Restrained']
}

Table club_membership_role {
    id SERIAL [pk]
    name VARCHAR(50) [unique, not null, note: 'Roles within a club: Creator, Admin,
Member']
}

Table club_membership_status {
    id SERIAL [pk]
    name VARCHAR(50) [unique, not null, note: 'Member status within a club: Invited,
Active, Suspended, Banned']
}

// --- Core Data Tables ---

Table connection {
    member_requester_id INTEGER [not null] // FK defined via Ref below
    member_accepter_id INTEGER [not null] // FK defined via Ref below
    friendship_status_id INTEGER [not null, ref: > friendship_status.id]
    created_at TIMESTAMPTZ [not null, default: `NOW()`]
    updated_at TIMESTAMPTZ [not null, default: `NOW()`]

    indexes {
        (member_requester_id, member_accepter_id) [pk]
        // Note: CHECK constraint (member_requester_id <> member_accepter_id) needs to be
added manually in SQL/migration
    }
}

Table club {
    id SERIAL [pk]
    creator_id INTEGER [not null] // FK defined via Ref below
    name VARCHAR(100) [unique, not null]
    description TEXT
    rules TEXT
    type VARCHAR(50) [not null, note: 'Public, Private, Hidden']
    created_at TIMESTAMPTZ [not null, default: `NOW()`]
}

Table club_membership {
    member_id INTEGER [not null] // FK defined via Ref below
    club_id INTEGER [not null] // FK defined via Ref below
    role_id INTEGER [not null, default: 3, ref: > club_membership_role.id, note: 'Default:
Member']
    status_id INTEGER [not null, default: 2, ref: > club_membership_status.id, note:
'Default: Active']
    joined_at TIMESTAMPTZ [not null, default: `NOW()`]

    indexes {
        (member_id, club_id) [pk]
    }
}

```



```
// --- Relationships ---

// Connection FKs (referencing prerequisite member table)
Ref connection_requester: connection.member_requester_id > member.id [delete: cascade]
Ref connection_accepter: connection.member_accepter_id > member.id [delete: cascade]

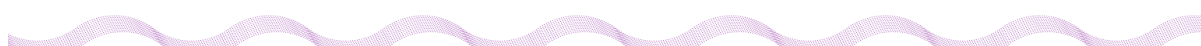
// Club FKs (referencing prerequisite member table)
Ref club_creator: club.creator_id > member.id

// Club Membership FKs (referencing prerequisite member table and new club table)
Ref membership_member: club_membership.member_id > member.id [delete: cascade]
Ref membership_club: club_membership.club_id > club.id [delete: cascade]

// END: Networking Feature Tables
```



3. SilverChat Project Documentation

- [SilverChat Project Scope \(README\) | GitHub](#)
 - [SilverChat Technical Architecture | GitHub](#)
 - [SilverChat Database Schema 1: Member Account and Profile | GitHub](#)
 - **THIS DOCUMENT:** *SilverChat Database Schema 2: Member Networking*
 - [SilverChat Database Schema 3: Member Messaging | GitHub](#)
- 

4. External Sources

4.1. Database Design for Social Networks (Schemas / Diagrams / ERMs / How Tos)

- [Facebook database schema | Reverse engineering by Anatoly Lu... | Flickr](#)
- [Database schema for Social Networking Platform - Surfside Media](#)
- [Building a Social Network: Part I | by Kenneth Reilly | ITNEXT](#)
- [Social network schema design in DynamoDB - Amazon DynamoDB](#)
- [Building a social Media Platform: How should the database schema be designed to efficiently store user data, content, and interactions? | by Brecht Corbeel | Medium](#)
- [SQLAlchemy: Designing a Social Network Database Schema - Sling Academy](#)
- [Databases, SQL Server, and Data Models Examples](#)
- [mysql - Implementing Comments and Likes in database - Stack Overflow](#)

- [How to Design Database for Social Media Platform | GeeksforGeeks](#)
 - [How to Design Database for Followers-Following Systems in Social Media Apps? | GeeksforGeeks](#)
 - [How to Design ER Diagrams for Social Media Networks | GeeksforGeeks](#)
 - [Resources: Database Design for Social Network - Code Doodle](#)
 - [Design Database For Social Network System In MySQL | Tutorials24x7](#)
-

4.2. Database GitHub Repos

- [Messenger Database Design Concept](#)
 - [The Social Network System Database Design in MySQL to manage the Users, Friends, Follower, Messages, and Groups.](#)
-

4.3. Database Infrastructure for Social Networks

- [A thorough insight into the databases used @Facebook - Scaleyourapp](#)
 - [Which database is best for creating a social networking application? - Quora](#)
 - [MySQL vs. MongoDB: The Pros and Cons When Building a Social Network](#)
-

4.4. Database Design Software

- [Top 10 Free Database Diagram Design Tools in 2025](#)
-

4.5. Facebook Reference

