

*Alpha Release 3*

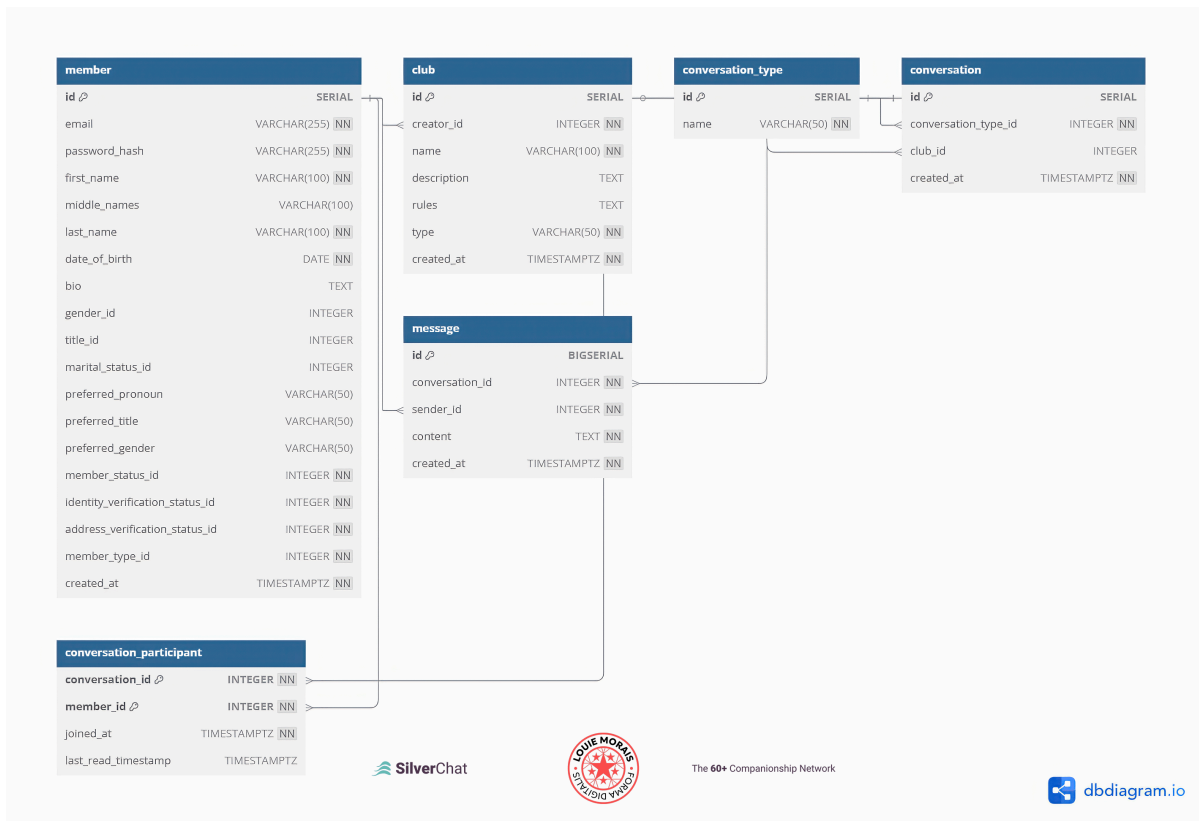
# Database Schema 3: Member Messaging

- Database Schema 3: Member Messaging
  - 1. Member Messaging 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 Messaging Tables

**SilverChat - Messaging Feature Database Schema (Target: Alpha Release 3)**

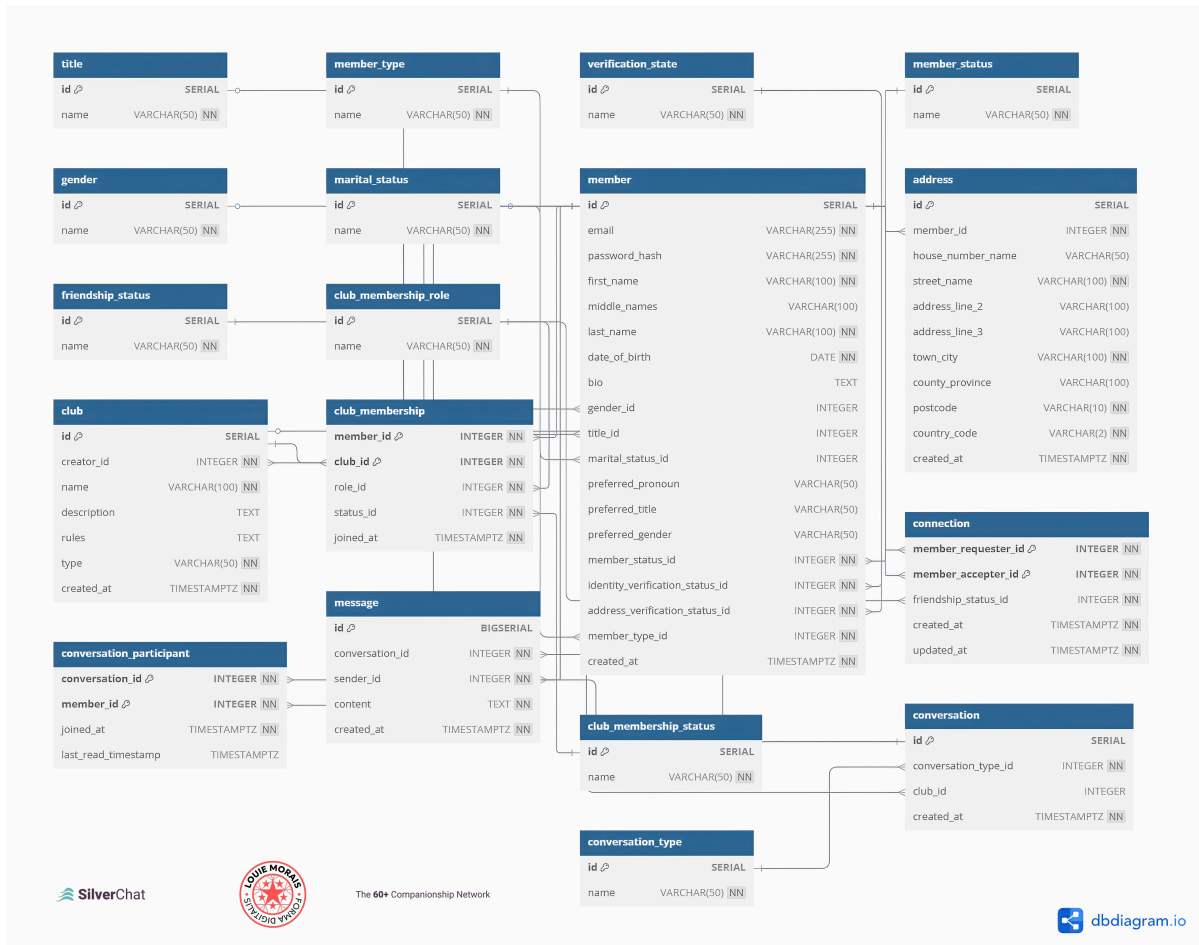
This document outlines the proposed database schema for handling member-to-member and club conversations within SilverChat, using PostgreSQL.



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

[\[BACK TO INDEX\]](#)

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



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

## 1.1. Lookup Tables

**Table:** `conversation_type`

Column Name	Data Type	Constraints	Description	Example Values
<code>id</code>	<code>SERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier for type.	1, 2
<code>name</code>	<code>VARCHAR(50)</code>	<code>UNIQUE NOT NULL</code>	Name of the conversation type (e.g., 'direct').	'direct', 'club'

## 1.2. Core Data Tables

**Table:** `conversation` (Represents a chat thread)

Column Name	Data Type	Constraints	Description
<code>id</code>	<code>SERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier for the conversation.
<code>conversation_type_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to <code>conversation_type</code> table.
<code>club_id</code>	<code>INTEGER</code>		Foreign key to <code>club</code> table (Nullable, used for club chats only).
<code>created_at</code>	<code>TIMESTAMPZ</code>	<code>NOT NULL</code> <code>DEFAULT NOW()</code>	Timestamp when the conversation was initiated.

**Table:** `conversation_participant` (Links members to conversations)

Column Name	Data Type	Constraints	Description
<code>conversation_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to <code>conversation</code> table.
<code>member_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to <code>member</code> table.

Column Name	Data Type	Constraints	Description
<code>joined_at</code>	<code>TIMESTAMPTZ</code>	<code>NOT NULL DEFAULT NOW()</code>	Timestamp when the member joined the conversation.
<code>last_read_timestamp</code>	<code>TIMESTAMPTZ</code>		Timestamp of the last message read by the member (Nullable).
		<code>PRIMARY KEY (conversation_id, member_id)</code>	Composite primary key ensures uniqueness.

**Table:** `message` (Stores individual chat messages)

Column Name	Data Type	Constraints	Description
<code>id</code>	<code>BIGSERIAL</code>	<code>PRIMARY KEY</code>	Auto-incrementing unique identifier (use <code>BIGSERIAL</code> for large volume).
<code>conversation_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to <code>conversation</code> table.
<code>sender_id</code>	<code>INTEGER</code>	<code>NOT NULL</code>	Foreign key to <code>member</code> table (who sent the message).
<code>content</code>	<code>TEXT</code>	<code>NOT NULL</code>	The actual content of the message.
<code>created_at</code>	<code>TIMESTAMPTZ</code>	<code>NOT NULL DEFAULT NOW()</code>	Timestamp when the message was sent.
		<code>INDEX (conversation_id, created_at DESC)</code>	Crucial index for fetching messages efficiently.

[\[BACK TO INDEX\]](#)



## 2. SQL Representation & Implementation Notes

These definitions assume integration into the existing PostgreSQL database used by SilverChat. Implementation should use Knex.js migrations.

---

### 2.1. SQL Syntax

*(Illustrative SQL - Actual implementation via Knex Migrations.)*

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

-- Prerequisite Lookup Tables (Assume these exist and are populated)
-- CREATE TABLE gender ...;
-- CREATE TABLE title ...;
-- CREATE TABLE marital_status ...;
-- CREATE TABLE member_status ...;
-- CREATE TABLE verification_state ...;
-- CREATE TABLE member_type ...;

-- Prerequisite: member Table
CREATE TABLE member (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  first_name VARCHAR(100) NOT NULL,
  middle_names VARCHAR(100),
  last_name VARCHAR(100) NOT NULL,
  date_of_birth DATE NOT NULL,
  bio TEXT,
  gender_id INTEGER,
  title_id INTEGER,
  marital_status_id INTEGER,
  preferred_pronoun VARCHAR(50),
  preferred_title VARCHAR(50),
  preferred_gender VARCHAR(50),
  member_status_id INTEGER NOT NULL DEFAULT 1,
  identity_verification_status_id INTEGER NOT NULL DEFAULT 1,
  address_verification_status_id INTEGER NOT NULL DEFAULT 1,
  member_type_id INTEGER NOT NULL DEFAULT 1,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
  -- Assume FKs to lookup tables are defined here in the main schema
);

-- Prerequisite: club Table
CREATE TABLE club (
  id SERIAL PRIMARY KEY,
  creator_id INTEGER NOT NULL, -- FK to member 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 (defined in main schema)
FOREIGN KEY (creator_id) REFERENCES member(id)
);

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

-- START: Messaging Feature Tables (Target: Alpha Release 3) --

-- Lookup Table for Messaging
CREATE TABLE conversation_type (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL -- e.g., 'direct', 'club'
);

-- Core Messaging Tables
CREATE TABLE conversation (
    id SERIAL PRIMARY KEY,
    conversation_type_id INTEGER NOT NULL,
    club_id INTEGER, -- Nullable
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Foreign Key Constraints
    FOREIGN KEY (conversation_type_id) REFERENCES conversation_type(id),
    FOREIGN KEY (club_id) REFERENCES club(id) -- Refers to prerequisite club table
);

CREATE TABLE conversation_participant (
    conversation_id INTEGER NOT NULL,
    member_id INTEGER NOT NULL,
    joined_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    last_read_timestamp TIMESTAMPTZ, -- Nullable

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

    -- Foreign Key Constraints
    FOREIGN KEY (conversation_id) REFERENCES conversation(id) ON DELETE CASCADE,
    FOREIGN KEY (member_id) REFERENCES member(id) ON DELETE CASCADE -- Refers to
prerequisite member table
);

CREATE TABLE message (
    id BIGSERIAL PRIMARY KEY,
    conversation_id INTEGER NOT NULL,
    sender_id INTEGER NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Foreign Key Constraints
    FOREIGN KEY (conversation_id) REFERENCES conversation(id) ON DELETE CASCADE,
    FOREIGN KEY (sender_id) REFERENCES member(id) -- Refers to prerequisite member table
);

-- Index for efficient message retrieval
CREATE INDEX idx_message_conversation_created_at ON message (conversation_id, created_at
DESC);

-- END: Messaging Feature Tables --

```

## 2.2. DBML Syntax

(DBML representation for visualization and design tools. `member` and `club` included for context only.)

```
// SilverChat - Messaging Feature Schema (Target: Alpha Release 3)

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

// Prerequisite Lookup Tables (Assume these exist)
// Table gender { ... }
// Table title { ... }
// Table marital_status { ... }
// Table member_status { ... }
// Table verification_state { ... }
// Table member_type { ... }

Table member { // PREREQUISITE - DO NOT RE-CREATE
  id SERIAL [pk]
  email VARCHAR(255) [unique, not null]
  password_hash VARCHAR(255) [not null]
  first_name VARCHAR(100) [not null]
  middle_names VARCHAR(100)
  last_name VARCHAR(100) [not null]
  date_of_birth DATE [not null]
  bio TEXT
  gender_id INTEGER // [ref: > gender.id] Assumed defined in main schema
  title_id INTEGER // [ref: > title.id] Assumed defined in main schema
  marital_status_id INTEGER // [ref: > marital_status.id] Assumed defined in main schema
  preferred_pronoun VARCHAR(50)
  preferred_title VARCHAR(50)
  preferred_gender VARCHAR(50)
  member_status_id INTEGER [not null, default: 1] // [ref: > member_status.id] Assumed
defined in main schema
  identity_verification_status_id INTEGER [not null, default: 1] // [ref: >
verification_state.id] Assumed defined in main schema
  address_verification_status_id INTEGER [not null, default: 1] // [ref: >
verification_state.id] Assumed defined in main schema
  member_type_id INTEGER [not null, default: 1] // [ref: > member_type.id] Assumed
defined in main schema
  created_at TIMESTAMPTZ [not null, default: `NOW()`]
}

Table club { // PREREQUISITE - DO NOT RE-CREATE
  id SERIAL [pk]
  creator_id INTEGER [not null, ref: > member.id] // Assumed defined in main schema
  name VARCHAR(100) [unique, not null]
  description TEXT
  rules TEXT
  type VARCHAR(50) [not null]
  created_at TIMESTAMPTZ [not null, default: `NOW()`]
}

// END: Prerequisite Tables
```

```
// =====

// START: Messaging Feature Tables (Target: Alpha Release 3)

Table conversation_type {
  id SERIAL [pk]
  name VARCHAR(50) [unique, not null, note: "'direct', 'club'"]
}

Table conversation {
  id SERIAL [pk]
  conversation_type_id INTEGER [not null, ref: > conversation_type.id]
  club_id INTEGER [ref: > club.id, note: 'Nullable, used for club chats only'] // Refers
to prerequisite club table
  created_at TIMESTAMPTZ [not null, default: `NOW()`]
}

Table conversation_participant {
  conversation_id INTEGER [not null] // FK defined via Ref below
  member_id INTEGER [not null] // FK defined via Ref below
  joined_at TIMESTAMPTZ [not null, default: `NOW()`]
  last_read_timestamp TIMESTAMPTZ [note: 'Nullable']

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

Table message {
  id BIGSERIAL [pk]
  conversation_id INTEGER [not null] // FK defined via Ref below
  sender_id INTEGER [not null] // FK defined via Ref below
  content TEXT [not null]
  created_at TIMESTAMPTZ [not null, default: `NOW()`]

  indexes {
    (conversation_id, created_at) [note: 'Index includes DESC in SQL']
  }
}

// --- Relationships (Explicit definitions for clarity/cascades) ---

// Ref conversation_club: conversation.club_id > club.id // Defined inline above

Ref participant_conversation: conversation_participant.conversation_id > conversation.id
[delete: cascade]
Ref participant_member: conversation_participant.member_id > member.id [delete: cascade]
// Refers to prerequisite member table

Ref message_conversation: message.conversation_id > conversation.id [delete: cascade]
Ref message_sender: message.sender_id > member.id // Refers to prerequisite member table

// END: Messaging Feature Tables
```



## 3. SilverChat Project Documentation

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

[\[BACK TO INDEX\]](#)

## 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](#)

---

[\[BACK TO INDEX\]](#)

### 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.](#)

---

[\[BACK TO INDEX\]](#)

## 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

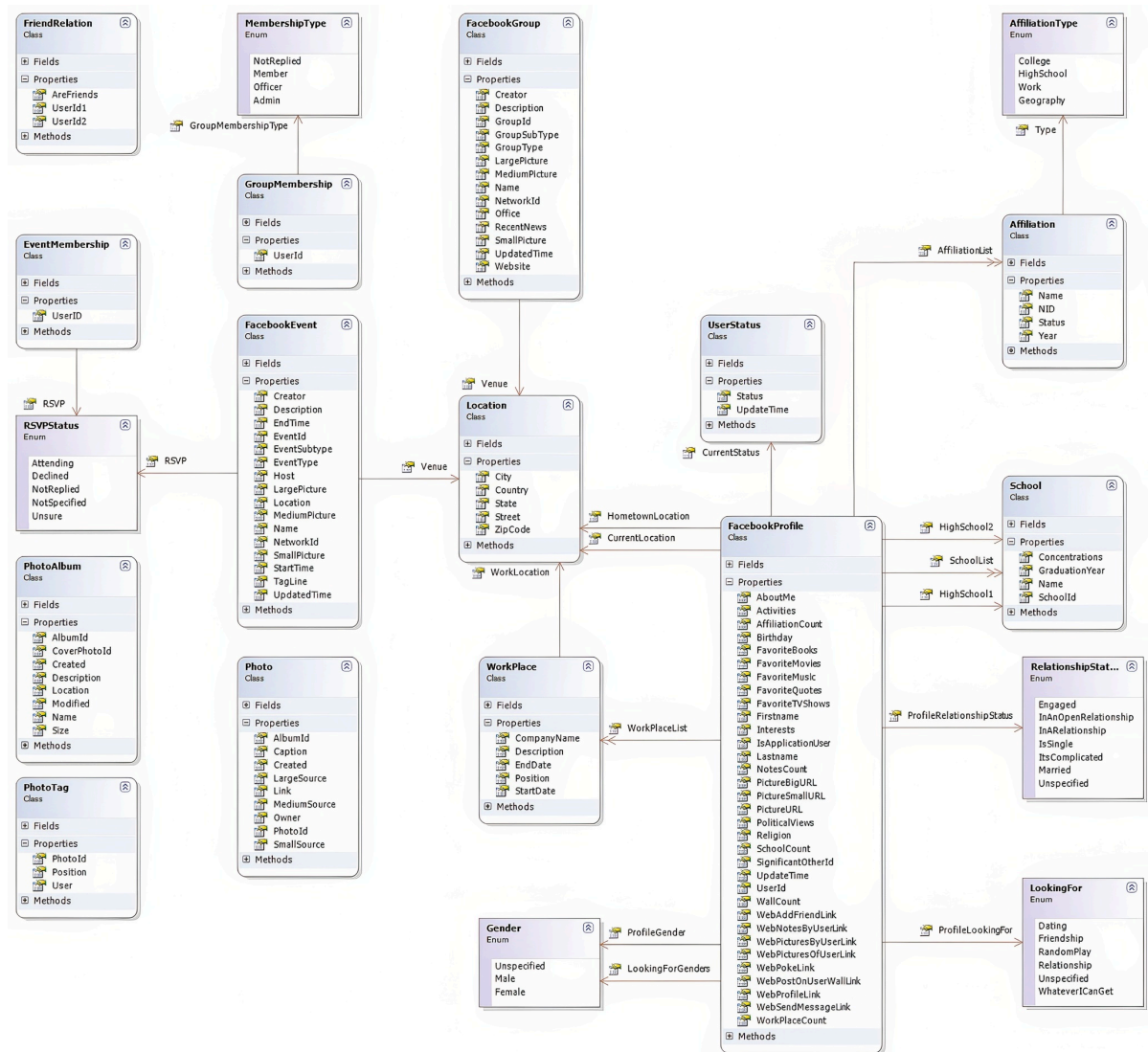
[\[BACK TO INDEX\]](#)

## 4.4. Database Design Software

- Top 10 Free Database Diagram Design Tools in 2025

[\[BACK TO INDEX\]](#)

## 4.5. Facebook Reference



[\[BACK TO INDEX\]](#)