ERD PROJECT : CHATBOT

Submitted to

Sir Shakir Rasheed

Submitted by

Muhammad Abdul Rehman
4767-FOC/BSSE/F23

Muneeb Ur Rehman
4751-FOC/BSSE/F23

Shah Usman Farooq
4753-FOC/BSSE/F23

Course Title: Introduction to Database Systems
Course Code: CS242

International Islamic University Islamabad
Department of Software Engineering

Introduction:

This document presents the Entity-Relationship Diagram (ERD) design for the chatbot system. It describes the data model that supports both direct (one-to-one) and group conversations, along with the entities, relationships, and constraints necessary to ensure data integrity, scalability, and efficient access to chat histories and media.

1. Purpose and scope

- Objective: Define the data model for a chatbot system that supports one-to-one and group chats, including messages, attachments, users, and group memberships.
- Scope: Focuses on chat messages, users, chat groups, attachments, and the relationships among them (who sent what, to whom, in which chat/group). Excludes backend analytics, AI models, and external integrations beyond basic messaging metadata.
- Stakeholders: End users (chat participants), product/engineering teams (data model), and support/ops teams (data integrity and backup).
- Assumptions: Each user has a unique username; a chat can be either a direct (one-to-one) chat or a group chat; attachments are linked to messages; each message has a sender and a timestamp.

2. Conceptual model (high level)

- Core entities:
    - User: A participant in one or more chats. Stores user-specific metadata (username, name, contact info, status)
    - Chat (or GroupChat): A conversation context. Can be a one-to-one chat or a group chat. Holds chat-wide metadata (type, name, creation time)
    - Message: A single piece of chat content sent by a user within a chat. Contains the text content and a timestamp
    - Attachment: A file linked to a specific message (e.g., images, documents).

- Key relationships (from the diagram):
    - User participates in Chat: A user can participate in many chats; a chat has many participants.
    - Chat contains Message: A chat contains many messages.
    - Message has Attachment: A message can have zero or more attachments.
    - Messages reference Sender (User) and Receiver (User) for direct messages, or a group context for group messages.

- Cardinalities (as inferred):
    - User 1..N ↔ N..M Chat (through a membership/participation association).
    - Chat 1..N ↔ 1..N Message (a chat has many messages; each message belongs to one chat).
    - Message 0..N ↔ 0..N Attachment (a message may have zero or more attachments).
    - Message Sender: a Message has exactly one Sender (User).
    - For direct chats, Receiver might be implied by the message or tracked as part of the chat participants.

3. Logical data model (normalized ERD)

- Entities and brief descriptions:
    - User
        - Attributes: user_id PK, username, first_name, last_name, phone, email, created_at, status
    - Chat (or GroupChat)
        - Attributes: chat_id PK, chat_type (e.g., 'direct', 'group'), name (optional for group chats), created_at
    - ChatParticipant (associative entity for many-to-many between User and Chat)
        - Attributes: chat_id FK, user_id FK, joined_at
        - Keys: Composite PK on (chat_id, user_id)
    - Message
        - Attributes: message_id PK, chat_id FK, sender_id FK (references User.user_id), content, timestamp
    - Attachment
        - Attributes: attachment_id PK, message_id FK, file_name, size, mime_type, created_at
- Relationships with cardinalities:

User — Chat (via ChatParticipant)

- Relationship: A user participates in chats; a chat has many participants.
- Cardinality: User 1..N connects to Chat via ChatParticipant; Chat 1..N connects to User via ChatParticipant.
- Interpretation: This is a many-to-many relationship realized through the associative entity ChatParticipant.
- Implications:
- You can query all chats a user participates in.

- You can query all participants in a chat.
- Adding/removing a user from a chat is a modification to ChatParticipant.

Chat — Message

- Relationship: A chat contains messages; each message belongs to exactly one chat.
- Cardinality: Chat 1..N ↔ 1..N Message (from Chat side: one chat has many messages; from Message side: each message belongs to one chat).
- Implications:
- Messages are grouped by chat, enabling efficient retrieval of chat history.
- Deleting a chat can cascade to delete its messages (depending on ON DELETE rules).

Message — Sender (User)

- Relationship: Each message has a single sender.
- Cardinality: Message 1..1 → 1..1 User (sender_id is FK to User).
- Implications:
- You can easily identify who sent each message.
- If a user is deleted, decide on cascade behavior for their messages (often you preserve messages with a synthetic/anonymous sender or cascade, depending on policy).

Message — Attachment

- Relationship: A message can have zero or more attachments; each attachment belongs to one message.
- Cardinality: Message 1..N ↔ 0..1 Attachment per link, but in practice: Message 1..1 → 0..N Attachment.
- Implications:
- Attachments are scoped to a message, so deleting a message removes its attachments (cascade).
- Attachments can be retrieved by message, enabling per-message media access.

Chat — (Implicit direct vs group context)

- Relationship: Chat_type distinguishes direct vs group chats; group chats may have a name.
- Cardinality: Direct and group chats share the same Chat entity; distinction is stored in chat_type.
- Implications:
- Queries can filter by chat_type to separate direct vs group conversations.
- Group chats leverage the same messaging mechanism with multiple participants

4. Keys and constraints

- Primary Keys (PK):
    - User: user_id
    - Chat: chat_id
    - Message: message_id
    - Attachment: attachment_id
- Foreign Keys (FK):
    - ChatParticipant: chat_id → Chat.chat_id; user_id → User.user_id
    - Message: chat_id → Chat.chat_id; sender_id → User.user_id
    - Attachment: message_id → Message.message_id
- Constraints and rules:
    - ChatParticipant: composite PK (chat_id, user_id) to enforce unique participation membership.
    - Not-null: essential fields like user_id, chat_id, sender_id, message_id, content (if required), timestamp.
    - Referential integrity: ON DELETE/UPDATE rules appropriate to your needs (e.g., cascade delete of messages when a chat is deleted; cascade delete attachments when a message is deleted).

5. Normalization and data integrity

- Normal Form: 3NF is typical here.
    - No transitive dependencies: user information separated from chat and message content.
    - Each fact stored in its appropriate table, minimizing redundancy.
- Derived data: timestamps are stored (created_at, timestamp); if you compute derived fields (e.g., message length, unread counts), consider caching or application-level computation.
- Integrity constraints:
    - Entity integrity: PKs unique and not null.
    - Referential integrity: FKs reference existing rows.

- Domain constraints: chat_type restricted to 'direct' or 'group'; content length limits; valid MIME types for attachments if enforced.

Common query patterns enabled by these relationships

- Retrieve a user's chats: join users → chat_participants → chats.
- Retrieve chat participants: join chats → chat_participants → users.
- Retrieve a chat's message history: join chats → messages → (optionally) users for sender details.
- Retrieve attachments for a message: join messages → attachments.
- Find all messages sent by a user in a chat: filter messages by sender_id and chat_id.
- Get group metadata vs direct chat flags: use chat_type and chat.name when present.

Illustrative example of a data retrieval flow

- To get the latest messages in a group chat with attachments:
  - Find chat_id of the group chat.
  - Select messages where chat_id = that_id, order by timestamp desc, limit N.
  - For each message, left join attachments to fetch any files.
- To list all participants in a direct chat:
  - Find chat_id, ensure chat_type = 'direct'.
  - Join ChatParticipant to Users to list participants (usually two).

Why this design supports both simplicity and scalability?

- Single Chat entity for both direct and group conversations reduces schema complexity and duplication.
- The ChatParticipant associative table cleanly models many-to-many membership, enabling flexible membership management without embedding lists in a single row.
- Separating Message and Attachment allows you to handle large media separately from text, and enables efficient streaming of chat histories while preserving rich media.

Conclusion:

The ERD design provides a robust, scalable foundation for a chatbot system that supports both direct and group chats. By separating concerns into Users, Chats, Messages, and Attachments (with a dedicated ChatParticipant associative table), the model achieves clear data integrity, flexible membership management, and efficient query patterns for chat histories.

Key strengths:

- Flexible chat types: direct and group chats share core structures, reducing duplication.
- Clear ownership and provenance: each message has a single sender and belongs to one chat, with attachments tied to messages.
- Strong referential integrity: well-defined FKs and cascade options help maintain consistency during deletions and updates.

Considerations:

- For very large chat histories, consider partitioning by chat_id or time-based windows to improve performance.
- Implement appropriate access controls and encryption for any PII, and audit trails if required.
- Extendability: the model supports future features such as reactions, message edits, read receipts, or reactions without major schema changes.

ERD FOR CHATBOT