DSTA 5733 Relational Database Design
FINAL PROJECT

By A.M. 9/13/2025

University of Colorado Boulder

Table of Contents

1. Step 1 – Background

DiabetesConnect Discusion Page

Introduction and Background Story:

I am a type-2 diabetic mom who also had gestational diabetes. Even in the U.S., getting quick, practical advice can be costly and slow. Small questions like why a CGM adhesive causes a rash, or why a finger-stick reads high while a sensor reads low, or how to calculate calories in a meal or in a snack…those simple questions all require scheduling an endocrinology visit.

To address this, I am creating **DiabetesConnect**: a moderated discussion board that links people living with diabetes to healthcare professionals (doctors, nurses, dietitians, …). The goal is a safe, informative space to share experiences, ask targeted questions, and receive timely guidance grounded in clinical expertise.

The initial scope focuses on connecting patients and professionals across the United States, with a long-term vision to expand globally. For this CU Boulder MS-DS Relational Database Design final project, I am delivering a right-sized "mini" version that models core functionality, users and roles, posts and threaded comments, topic tags for discovery, and simple quality signals (e.g., ratings). The platform is designed to complement, not replacing personal medical care.

Purpose:

- Empower patients with reliable, personalized advice
- Allow medical staff to share insights and monitor common concerns
- Build a community around chronic disease management

Assumptions:

- Users are either patients or medical staff or patient's guardian.
- Medical staff have verified credentials and specialties.
- Patients can post questions, share experiences, and comments.
- Medical staff can respond, moderate, and post educational content.
- Posts are tagged by topic (e.g., insulin, diet, exercise).
- There's a message board structure with threads and replies/comments.
- Users can rate posts or comments for helpfulness.

2. Step 2: Build the Entity Relationship Model (ERM)

2.1 Overview

The model below defines core entities, their purposes, attributes, and how they relate. It also captures role semantics (who can do what), discussion features (posts, comments, ratings), topic taxonomy (tags), and care relationships (guardianships).

2.2 Entities (Definitions, Purpose, Attributes)

2.2.1 Entity List:

There are 9 entities.

Entities	Description
ROLES	Defines the roles of users within the application. Possible roles include: Doctor, Nurse, Dietitian, Patient, and Patient Guardian. Additional roles can be added as the application expands, and more user types join.
USERS	Represents individuals who use the application. These include users who engage with **DiabetesConnect** to seek or provide medical advice.
MEDICAL_STAFFS	A subset of users who are certified medical professionals. They contribute by answering questions and offering medical guidance to patient users.
CONTACTS	Stores contact information for each user, such as email, phone number, or address. Addresses are considered private and optional, depending on user preference.
POSTS	Content shared by users, which may include questions from patients or informative posts from medical staff on specific topics.
COMMENTS	Responses or discussions related to posts. These often serve as answers or follow-up insights.
POST_RATINGS	Ratings given by users to evaluate the quality or usefulness of a post.
TOPIC_TAGS	Tags that categorize posts and comments by topic, such as Insulin, Diet, A1C, etc., all relevant to diabetes care.
COMMENT_RATINGS	Ratings provided by users to assess the helpfulness or relevance of comments.

2.2.2 Entity Details:

1) ROLES

Definition & Purpose:

Defines distinct categories of users and enables role-based access and tailored experiences. Roles drive permissions (e.g., who can answer medical questions vs. who can post personal updates).

Example Roles:

Doctor, Nurse, Dietitian, Patient, Patient Guardian.

Attributes:

ROLE_ID (PK): System unique identifier (e.g., 1, 2, 3, 4, 5).

ROLE_NAME: Human-readable label (unique).

Notes:

Roles are referenced by USERS.

Application logic or DB checks enforce role-specific actions.

2) USERS

Definition & Purpose:

Accounts for everyone using the application: patients, guardians, and professionals. Serves as the identity backbone for authorship, ratings, and care relations.

Core Behaviors:

Create posts/comments, give ratings.

Patients/guardians access care info; professionals provide clinical guidance.

Audited events: sign-up, last login, password update.

Attributes:

USER_ID (PK): System-wide unique ID.

ROLE_ID (FK → ROLES): Assigns role-based capabilities.

USERNAME (unique): Login handle.

PASSWORD: Stored as a secure hash (never plaintext).

JOINED_DATE: Registration timestamp.

FULL_NAME: Display name for UI and communications.

Notes:

One role per user (simple & clear). If you later need multi-role, introduce a junction table.

3) MEDICAL_STAFFS

Definition & Purpose:

Captures the subset of USERS who are licensed/qualified healthcare professionals (HCPs), such as doctors, nurses, dietitians. Extends USERS with clinical credentials and availability.

Who belongs here?

Only USERS with clinical roles (Doctor/Nurse/Dietitian). Not all USERS will have a MEDICAL_STAFFS row.

Attributes:

STAFF_ID (PK): Internal unique identifier for staff.

USER_ID (FK → USERS, unique): 1:1 link to the user identity.

SPECIALIZATION: e.g., Endocrinology, Pediatric Nursing, Clinical Nutrition.

LICENSE_NUMBER (unique): Board/State license (verify format per region).

AFFILIATED_INSTITUTION: Hospital/clinic name.

YEARS_OF_EXPERIENCE: Whole number; allow 0–60+.

AVAILABILITY_STATUS: e.g., Available | Off-duty | On-leave.

CONTACT_ID (FK → CONTACTS): Phone/email/address record.

Notes:

Enforce consistency: a MEDICAL_STAFFS row implies USER's role exists in the set of {Doctor, Nurse, Dietitian, ...}.

Consider periodic validation of LICENSE_NUMBER.

4) CONTACTS

Definition & Purpose:

Stores contact details, kept separate to simplify updates and comply with privacy controls (e.g., hide phone from non-patients).

Attributes:

CONTACT_ID (PK)

EMAIL

PHONE

ADDRESS

Notes:

Linked 1:1 from MEDICAL_STAFFS (each staff has exactly one CONTACTS record).

You can later reuse CONTACTS for other types of actors if needed.

5) POSTS

Definition & Purpose:

User-generated top-level content: questions to clinicians, educational notes, progress updates, or community posts. Foundation for discussion and tagging.

Attributes:

POST_ID (PK): System-created ID.

USER_ID (FK → USERS): Author (patient, guardian, or staff).

TITLE

CONTENT (rich text allowed)

POST_DATE (timestamp)

Notes:

Tagging is many-to-many via POST_TAG.

Moderation flags can be added later (e.g., IS_FLAGGED, FLAG_REASON).

6) COMMENTS

Definition & Purpose:

Replies to POSTS to enable threaded discussions and care Q&A. Each comment belongs to exactly one post and has an author.

Attributes:

COMMENT_ID (PK)

POST_ID (FK → POSTS)

USER_ID (FK → USERS): Author

CONTENT

COMMENT_DATE

Notes:

If you need nested replies, add PARENT_COMMENT_ID (self-FK) later.

Consider content safety and audit fields.

7) COMMENT_RATINGS

Definition & Purpose:

Feedback on comments (not posts) to prioritize helpful clinical answers and surface quality information.

Attributes:

RATING_ID (PK)

COMMENT_ID (FK → COMMENTS): Target of the rating

USER_ID (FK → USERS): Rater

SCORE: Typically 1–5

RATING_DATE

Business Rules:

A user may rate a given comment at most once → unique (USER_ID, COMMENT_ID).

Optionally prevent self-rating (author ≠ rater) via app logic or a CHECK with trigger.

8) POST_RATINGS

Definition & Purpose:

Feedback on posts to prioritize helpful clinical answers and surface quality information.

Attributes:

RATING_ID (PK)

POST_ID (FK → POST): Target of the rating

USER_ID (FK → USERS): Rater

SCORE: Typically 1–5

RATING_DATE

Business Rules:

A user may rate a given comment at most once → unique (USER_ID, POST_ID).

Optionally prevent self-rating (author ≠ rater) via app logic or a CHECK with trigger.

9) TOPIC_TAGS

Definition & Purpose:

Controlled vocabulary for categorizing posts to improve discovery, filtering, and analytics (e.g., “Insulin”, “A1C”, “Diet”, “Hypoglycemia”).

Attributes:

TOPIC_TAG_ID (PK)

TOPIC_TAG_NAME (unique)

Notes:

Maintain uniqueness and clear naming; consider a description field for clarity.

10) POST_TAG (junction)

Definition & Purpose:

Resolves the many-to-many relationship between POSTS and TOPIC_TAGS.

Attributes / Keys:

POST_ID (FK → POSTS)

TOPIC_TAG_ID (FK → TOPIC_TAGS)

(PK = POST_ID, TOPIC_TAG_ID) to prevent duplicate tags on a post.

Notes:

Add an index on TOPIC_TAG_ID for tag browsing performance.

11) USER_CONTACT (junction)

Definition & Purpose:

Resolves the many-to-many relationship between USERS and CONTACTS.

Attributes / Keys:

USER_ID (FK → USERS)

CONTACT _ID (FK → CONTACTS)

(PK = POST_ID, TOPIC_TAG_ID) to prevent duplicate tags on a post.

Notes:

Add an index on TOPIC_TAG_ID for tag browsing performance.

2.3 Relationship

The following describes the relationships among entities in the system:

1. **Users – Roles**
    - Each user is assigned exactly one role (e.g., doctor, nurse, patient, guardian).
    - Each role can be associated with many users.
    - **Cardinality:** USERS (many) → ROLES (1).

2. **Users – Contacts (via USER_CONTACT)**
    - A user can have zero or many contact methods (phone numbers, emails, etc.).
    - A contact can belong to one or many users.
    - This many-to-many relationship is resolved through the junction table USER_CONTACT.
3. **Users – Medical Staffs**
    - A user may or may not be a medical staff member.
    - If the user is a medical staff, they have exactly one record in the MEDICAL_STAFFS table.
    - Every medical staff entry must correspond to exactly one user.
    - **Cardinality:** USERS (0..1) ↔ MEDICAL_STAFFS (1).
4. **Users – Posts**
    - A user can create zero or many posts.
    - Each post must be created by exactly one user.
    - **Cardinality:** USERS (1..*) → POSTS (many).
5. **Users – Comments**
    - A user can write zero or many comments.
    - Each comment must be authored by exactly one user.
    - **Cardinality:** USERS (1..*) → COMMENTS (many).
6. **Posts – Comments**
    - A post can receive zero or many comments.
    - Each comment must belong to exactly one post.
    - **Cardinality:** POSTS (0..*) ↔ COMMENTS (1).
7. **Posts – Post Ratings**
    - A post can be rated by many users.
    - A user can rate many posts, but only once per post.

- This many-to-many relationship is resolved through the entity POST_RATINGS.

8. **Comments – Comment Ratings**
    - A comment can be rated by many users.
    - A user can rate many comments, but only once per comment.
    - This many-to-many relationship is resolved through the entity COMMENT_RATINGS.
9. **Posts – Topic Tags (via POST_TAG)**
    - A post can be associated with zero or many topic tags.
    - A topic tag can be applied to zero or many posts.
    - This many-to-many relationship is resolved through the junction table POST_TAG.

3. Step 3: Convert the ERD to the Relational Model

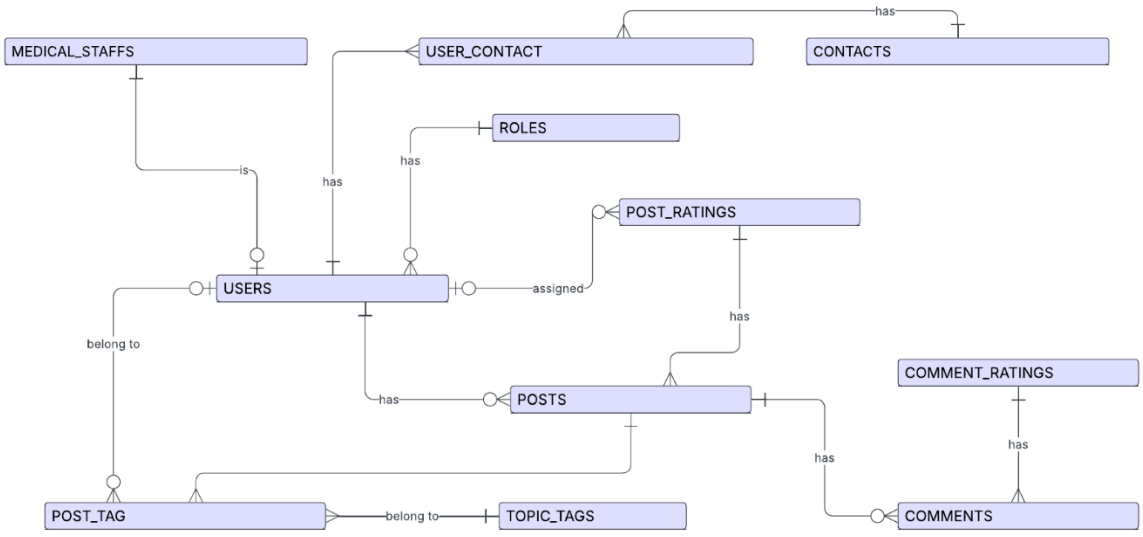

Figure 1: Short ERD of DiabetesConnect

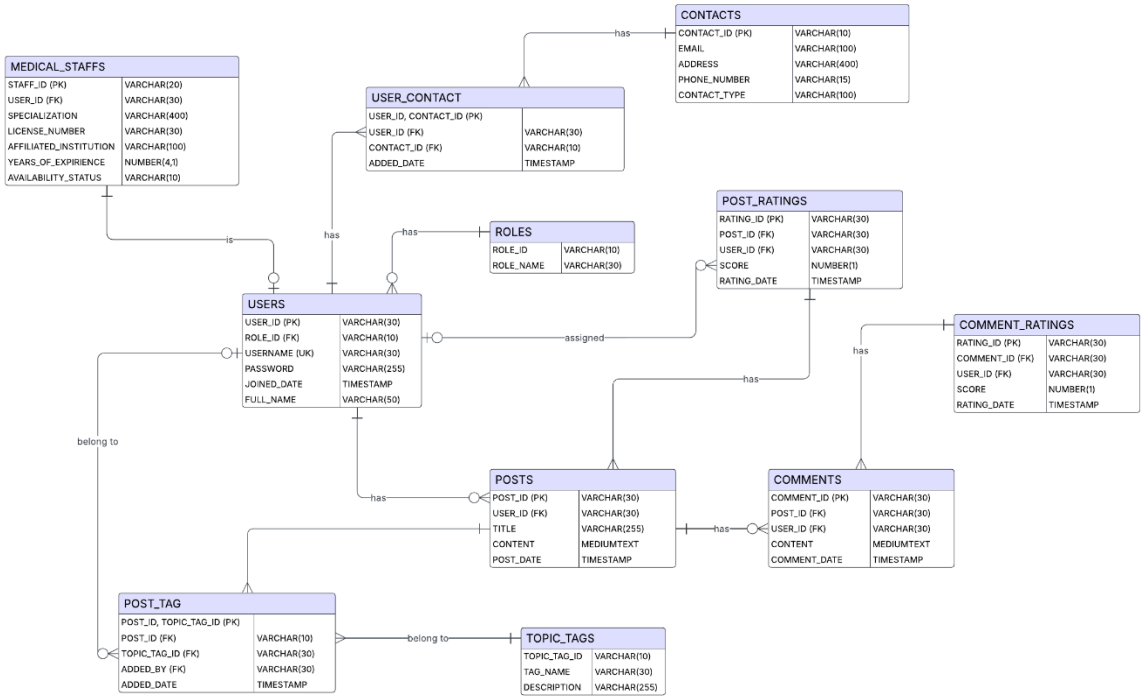

Figure 2: Detail ERD of DiabetesConnect

4. Step 4: Normalize the Relational Model to 3NF

The entities and their attributes have been normalized to ensure data integrity, eliminate redundancy, and improve query efficiency. The normalization process was applied in three stages:

2.1 First Normal Form (1NF)

- All tables have atomic attribute values (no multi-valued or repeating groups).
  Example:
  - CONTACTS separates each phone number, email, or address into individual rows, rather than storing multiple contacts in one column.
- Each table has a primary key that uniquely identifies each row.
  Example:
  - USERS (USER_ID) and POSTS (POST_ID).

2.2 Second Normal Form (2NF)

- All non-key attributes depend fully on the primary key.

- Composite relationships (many-to-many) are resolved through junction tables.
  Example:
    - USER_CONTACT ensures each contact belongs to one or more users without duplicating contact data in the USERS table.

  Example:

    - POST_TAG separates the association of posts and topic tags, avoiding repeated tag columns in the POSTS table.

2.3 Third Normal Form (3NF)

- All non-key attributes depend only on the primary key and not on other non-key attributes.
- Transitive dependencies were removed:
  Example:
    - Role information (ROLE_NAME) is stored in ROLES, not repeated in USERS.

  Example:
    - Specialization and license information for staff are stored in MEDICAL_STAFFS, not in USERS.

  Example:
    - Ratings (RATING_VALUE) are separated into POST_RATINGS and COMMENT_RATINGS instead of being embedded in POSTS or COMMENTS.

In a result, the final relational model in 3NF:

- Reduces redundancy (no repeated role names, tags, or contact information).
- Maintains data integrity through foreign keys and unique constraints.
- Provides flexibility for future extensions (for example, adding new contact types, new rating systems, or expanding roles).

5. Step 5: Final Output for Implementation

After designing the ER model, refining the relationships, and normalizing to 3NF, the final relational schema is implemented using SQL CREATE TABLE statements. The schema

includes primary keys, foreign keys, and unique constraints to enforce entity integrity and relationship cardinalities.

5.1 Core Tables

USERS

Stores information about all users in the system (patients, guardians, doctors, nurses, administrators).

Attributes: USER_ID (PK), USERNAME (unique), PASSWORD, FULL_NAME, ROLE_ID (FK).

ROLES

Defines user roles (e.g., doctor, nurse, patient, guardian).

Attributes: ROLE_ID (PK), ROLE_NAME (unique).

CONTACTS

Stores contact details (phone, email, etc.).

Attributes: CONTACT_ID (PK), CONTACT_TYPE, CONTACT_VALUE.

USER_CONTACT

Junction table between USERS and CONTACTS.

Attributes: USER_ID (FK), CONTACT_ID (FK).

Constraints: UNIQUE(USER_ID, CONTACT_ID) to prevent duplicates.

MEDICAL_STAFFS

Subset of users who are medical staff (doctors, nurses).

Attributes: STAFF_ID (PK), USER_ID (FK, unique), SPECIALIZATION, LICENSE_NUMBER, AFFILIATED_INSTITUTION, YEARS_OF_EXPERIENCE, AVAILABILITY_STATUS.

Constraints: UNIQUE(USER_ID) ensures a user can only appear once in this table.

5.2 Content Tables

POSTS

Stores user-created posts.

Attributes: POST_ID (PK), USER_ID (FK), CONTENT, CREATED_AT.

COMMENTS

Stores comments linked to posts.

Attributes: COMMENT_ID (PK), POST_ID (FK), USER_ID (FK), CONTENT, CREATED_AT.

5.3 Ratings Tables

POST_RATINGS

Allows users to rate posts.

Attributes: RATING_ID (PK), USER_ID (FK), POST_ID (FK), RATING_VALUE.

Constraints: UNIQUE(USER_ID, POST_ID) ensures a user rates a post only once.

COMMENT_RATINGS

Allows users to rate comments.

Attributes: RATING_ID (PK), USER_ID (FK), COMMENT_ID (FK), RATING_VALUE.

Constraints: UNIQUE(USER_ID, COMMENT_ID) ensures a user rates a comment only once.

5.4 Tagging Tables

TOPIC_TAGS

Stores predefined topic tags.

Attributes: TAG_ID (PK), TAG_NAME (unique).

POST_TAG

Junction table linking posts and tags.

Attributes: POST_ID (FK), TAG_ID (FK).

Constraints: UNIQUE(POST_ID, TAG_ID) prevents duplicate tagging.

5.5 Implementation Summary

Primary Keys ensure entity uniqueness (USER_ID, STAFF_ID, POST_ID, etc.).

Foreign Keys maintain referential integrity across entities (e.g., POSTS.USER_ID → USERS.USER_ID).

Unique Constraints enforce business rules (e.g., one rating per user per post/comment, one med staff record per user).

Cascade Options (ON DELETE CASCADE) ensure dependent records are removed when parent records are deleted, preserving integrity.

5.6 Implementation in MySQL

```
-- ============================================================
-- Schema DDL (MySQL 8.0+)
-- Notes: N/A
-- ============================================================

SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;

-- ------------------------------------------------------------
-- 1) LOOKUP TABLES
-- ------------------------------------------------------------
CREATE TABLE IF NOT EXISTS ROLES (
  ROLE_ID    VARCHAR(10)  NOT NULL,
  ROLE_NAME  VARCHAR(30)  NOT NULL,
  PRIMARY KEY (ROLE_ID),
  CONSTRAINT uq_roles_role_name UNIQUE (ROLE_NAME)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS USERS (
  USER_ID     VARCHAR(30)  NOT NULL,
  ROLE_ID     VARCHAR(10)  NOT NULL,
  USERNAME    VARCHAR(30)  NOT NULL,
  PASSWORD    VARCHAR(255) NOT NULL,       -- store HASHED passwords
  JOINED_DATE TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
  PRIMARY KEY (USER_ID),
  CONSTRAINT uq_users_username UNIQUE (USERNAME),
  CONSTRAINT fk_users_role
    FOREIGN KEY (ROLE_ID) REFERENCES ROLES(ROLE_ID)
    ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS CONTACTS (
  CONTACT_ID    VARCHAR(10)   NOT NULL,
  EMAIL         VARCHAR(100),
  ADDRESS       VARCHAR(400),
  PHONE_NUMBER  VARCHAR(15),
  CONTACT_TYPE  VARCHAR(100),
  PRIMARY KEY (CONTACT_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS TOPIC_TAGS (
  TOPIC_TAG_ID  VARCHAR(10)  NOT NULL,
  TAG_NAME      VARCHAR(30)  NOT NULL,
  DESCRIPTION   VARCHAR(255),
  PRIMARY KEY (TOPIC_TAG_ID),
  CONSTRAINT uq_topic_tags_tag_name UNIQUE (TAG_NAME)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ------------------------------------------------------------
-- 2) USER <-> CONTACT (bridge, many-to-many)
-- ------------------------------------------------------------
CREATE TABLE IF NOT EXISTS USER_CONTACT (
  USER_ID     VARCHAR(30) NOT NULL,
  CONTACT_ID  VARCHAR(10) NOT NULL,
  ADDED_DATE  TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (USER_ID, CONTACT_ID),
  CONSTRAINT fk_user_contact_user
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_user_contact_contact
    FOREIGN KEY (CONTACT_ID) REFERENCES CONTACTS(CONTACT_ID)
    ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ------------------------------------------------------------
-- 3) MEDICAL STAFFS (subset of USERS; 1-to-0..1)
-- ------------------------------------------------------------
CREATE TABLE IF NOT EXISTS MEDICAL_STAFFS (
  STAFF_ID               VARCHAR(20) NOT NULL,
  USER_ID                VARCHAR(30) NOT NULL,
  SPECIALIZATION         VARCHAR(400),
  LICENSE_NUMBER         VARCHAR(30),
  AFFILIATED_INSTITUTION VARCHAR(100),
  YEARS_OF_EXPERIENCE    DECIMAL(4,1),
  AVAILABILITY_STATUS    VARCHAR(10),
  PRIMARY KEY (STAFF_ID),
  CONSTRAINT uq_medstaff_user UNIQUE (USER_ID),
  CONSTRAINT fk_medstaff_user
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
-- ------------------------------------------------------
-- 4) POSTS & COMMENTS
-- ------------------------------------------------------
CREATE TABLE IF NOT EXISTS POSTS (
  POST_ID    VARCHAR(30)  NOT NULL,
  USER_ID    VARCHAR(30)  NOT NULL,
  TITLE      VARCHAR(255),
  CONTENT    MEDIUMTEXT,
  POST_DATE  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (POST_ID),
  KEY ix_posts_user (USER_ID),
  CONSTRAINT fk_posts_user
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS COMMENTS (
  COMMENT_ID   VARCHAR(30) NOT NULL,
  POST_ID      VARCHAR(30) NOT NULL,
  USER_ID      VARCHAR(30) NOT NULL,
  CONTENT      MEDIUMTEXT,
  COMMENT_DATE TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (COMMENT_ID),
  KEY ix_comments_post (POST_ID),
  KEY ix_comments_user (USER_ID),
  CONSTRAINT fk_comments_post
    FOREIGN KEY (POST_ID) REFERENCES POSTS(POST_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_comments_user
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ------------------------------------------------------
-- 5) POST <-> TAGS (junction)
--    Fix: POST_ID length now matches POSTS.POST_ID (VARCHAR(30))
-- ------------------------------------------------------
CREATE TABLE IF NOT EXISTS POST_TAG (
  POST_ID       VARCHAR(30) NOT NULL,   -- match POSTS.POST_ID
  TOPIC_TAG_ID  VARCHAR(10) NOT NULL,
  ADDED_BY      VARCHAR(30),
  ADDED_DATE    TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (POST_ID, TOPIC_TAG_ID),
  KEY ix_post_tag_tag (TOPIC_TAG_ID),
  KEY ix_post_tag_added_by (ADDED_BY),
  CONSTRAINT fk_post_tag_post
    FOREIGN KEY (POST_ID) REFERENCES POSTS(POST_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_post_tag_tag
    FOREIGN KEY (TOPIC_TAG_ID) REFERENCES TOPIC_TAGS(TOPIC_TAG_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_post_tag_added_by
    FOREIGN KEY (ADDED_BY) REFERENCES USERS(USER_ID)
    ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
-- ------------------------------------------------------------
-- 6) RATINGS for POSTS & COMMENTS (1 per user per target)
-- ------------------------------------------------------------
CREATE TABLE IF NOT EXISTS POST_RATINGS (
  RATING_ID   VARCHAR(30) NOT NULL,
  POST_ID     VARCHAR(30) NOT NULL,
  USER_ID     VARCHAR(30) NOT NULL,
  SCORE       TINYINT     NOT NULL,
  RATING_DATE TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (RATING_ID),
  UNIQUE KEY uq_post_ratings_user_post (USER_ID, POST_ID),
  KEY ix_post_ratings_post (POST_ID),
  KEY ix_post_ratings_user (USER_ID),
  CONSTRAINT fk_post_ratings_post
    FOREIGN KEY (POST_ID) REFERENCES POSTS(POST_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_post_ratings_user
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT ck_post_ratings_score CHECK (SCORE BETWEEN 1 AND 5)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS COMMENT_RATINGS (
  RATING_ID   VARCHAR(30) NOT NULL,
  COMMENT_ID  VARCHAR(30) NOT NULL,
  USER_ID     VARCHAR(30) NOT NULL,
  SCORE       TINYINT     NOT NULL,
  RATING_DATE TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (RATING_ID),
  UNIQUE KEY uq_comment_ratings_user_comment (USER_ID, COMMENT_ID),
  KEY ix_comment_ratings_comment (COMMENT_ID),
  KEY ix_comment_ratings_user (USER_ID),
  CONSTRAINT fk_comment_ratings_comment
    FOREIGN KEY (COMMENT_ID) REFERENCES COMMENTS(COMMENT_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_comment_ratings_user
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT ck_comment_ratings_score CHECK (SCORE BETWEEN 1 AND 5)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

SET FOREIGN_KEY_CHECKS = 1;
```

5.7 Implementation in PostgreSQL

```sql
-- ================================================
-- PostgreSQL DDL (compatible with Postgres 14+)
-- Notes:N/A
-- ================================================

-- 1) LOOKUP TABLES
CREATE TABLE IF NOT EXISTS roles (
  role_id    VARCHAR(10)  NOT NULL,
  role_name  VARCHAR(30)  NOT NULL,
```

```
  CONSTRAINT pk_roles PRIMARY KEY (role_id),
  CONSTRAINT uq_roles_role_name UNIQUE (role_name)
);

CREATE TABLE IF NOT EXISTS users (
  user_id     VARCHAR(30)  NOT NULL,
  role_id     VARCHAR(10)  NOT NULL,
  username    VARCHAR(30)  NOT NULL,
  password    VARCHAR(255) NOT NULL,       -- store HASHED passwords
  joined_date TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_users PRIMARY KEY (user_id),
  CONSTRAINT uq_users_username UNIQUE (username),
  CONSTRAINT fk_users_role
    FOREIGN KEY (role_id) REFERENCES roles(role_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
);

CREATE TABLE IF NOT EXISTS contacts (
  contact_id    VARCHAR(10)   NOT NULL,
  email         VARCHAR(100),
  address       VARCHAR(400),
  phone_number  VARCHAR(15),
  contact_type  VARCHAR(100),
  CONSTRAINT pk_contacts PRIMARY KEY (contact_id)
);

CREATE TABLE IF NOT EXISTS topic_tags (
  topic_tag_id  VARCHAR(10)  NOT NULL,
  tag_name      VARCHAR(30)  NOT NULL,
  description   VARCHAR(255),
  CONSTRAINT pk_topic_tags PRIMARY KEY (topic_tag_id),
  CONSTRAINT uq_topic_tags_tag_name UNIQUE (tag_name)
);

-- 2) USER <-> CONTACT (bridge, many-to-many)
CREATE TABLE IF NOT EXISTS user_contact (
  user_id     VARCHAR(30) NOT NULL,
  contact_id  VARCHAR(10) NOT NULL,
  added_date  TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_user_contact PRIMARY KEY (user_id, contact_id),
  CONSTRAINT fk_user_contact_user
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_user_contact_contact
    FOREIGN KEY (contact_id) REFERENCES contacts(contact_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);

-- 3) MEDICAL STAFFS (subset of USERS; 1-to-0..1)
CREATE TABLE IF NOT EXISTS medical_staffs (
  staff_id               VARCHAR(20) NOT NULL,
  user_id                VARCHAR(30) NOT NULL,
  specialization         VARCHAR(400),
  license_number         VARCHAR(30),
  affiliated_institution VARCHAR(100),
```

```
  years_of_experience    NUMERIC(4,1),    -- consider NUMERIC(3,1) or
INTEGER if preferred
  availability_status    VARCHAR(10),
  CONSTRAINT pk_medical_staffs PRIMARY KEY (staff_id),
  CONSTRAINT uq_medstaff_user UNIQUE (user_id),
  CONSTRAINT fk_medstaff_user
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);

-- 4) POSTS & COMMENTS
CREATE TABLE IF NOT EXISTS posts (
  post_id    VARCHAR(30)  NOT NULL,
  user_id    VARCHAR(30)  NOT NULL,
  title      VARCHAR(255),
  content    TEXT,
  post_date  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_posts PRIMARY KEY (post_id),
  CONSTRAINT fk_posts_user
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE INDEX IF NOT EXISTS ix_posts_user ON posts(user_id);

CREATE TABLE IF NOT EXISTS comments (
  comment_id   VARCHAR(30) NOT NULL,
  post_id      VARCHAR(30) NOT NULL,
  user_id      VARCHAR(30) NOT NULL,
  content      TEXT,
  comment_date TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_comments PRIMARY KEY (comment_id),
  CONSTRAINT fk_comments_post
    FOREIGN KEY (post_id) REFERENCES posts(post_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_comments_user
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE INDEX IF NOT EXISTS ix_comments_post ON comments(post_id);
CREATE INDEX IF NOT EXISTS ix_comments_user ON comments(user_id);

-- 5) POST <-> TAGS (junction)
CREATE TABLE IF NOT EXISTS post_tag (
  post_id       VARCHAR(30) NOT NULL,   -- matches posts.post_id
  topic_tag_id  VARCHAR(10) NOT NULL,
  added_by      VARCHAR(30),
  added_date    TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_post_tag PRIMARY KEY (post_id, topic_tag_id),
  CONSTRAINT fk_post_tag_post
    FOREIGN KEY (post_id) REFERENCES posts(post_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_post_tag_tag
    FOREIGN KEY (topic_tag_id) REFERENCES topic_tags(topic_tag_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_post_tag_added_by
```

```
    FOREIGN KEY (added_by) REFERENCES users(user_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);

CREATE INDEX IF NOT EXISTS ix_post_tag_tag ON post_tag(topic_tag_id);
CREATE INDEX IF NOT EXISTS ix_post_tag_added_by ON post_tag(added_by);

-- 6) RATINGS for POSTS & COMMENTS (1 per user per target)
CREATE TABLE IF NOT EXISTS post_ratings (
  rating_id   VARCHAR(30) NOT NULL,
  post_id     VARCHAR(30) NOT NULL,
  user_id     VARCHAR(30) NOT NULL,
  score       SMALLINT    NOT NULL,
  rating_date TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_post_ratings PRIMARY KEY (rating_id),
  CONSTRAINT uq_post_ratings_user_post UNIQUE (user_id, post_id),
  CONSTRAINT fk_post_ratings_post
    FOREIGN KEY (post_id) REFERENCES posts(post_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_post_ratings_user
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT ck_post_ratings_score CHECK (score BETWEEN 1 AND 5)
);

CREATE INDEX IF NOT EXISTS ix_post_ratings_post ON post_ratings(post_id);
CREATE INDEX IF NOT EXISTS ix_post_ratings_user ON post_ratings(user_id);

CREATE TABLE IF NOT EXISTS comment_ratings (
  rating_id   VARCHAR(30) NOT NULL,
  comment_id  VARCHAR(30) NOT NULL,
  user_id     VARCHAR(30) NOT NULL,
  score       SMALLINT    NOT NULL,
  rating_date TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_comment_ratings PRIMARY KEY (rating_id),
  CONSTRAINT uq_comment_ratings_user_comment UNIQUE (user_id, comment_id),
  CONSTRAINT fk_comment_ratings_comment
    FOREIGN KEY (comment_id) REFERENCES comments(comment_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_comment_ratings_user
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT ck_comment_ratings_score CHECK (score BETWEEN 1 AND 5)
);

CREATE INDEX IF NOT EXISTS ix_comment_ratings_comment ON
comment_ratings(comment_id);
CREATE INDEX IF NOT EXISTS ix_comment_ratings_user ON
comment_ratings(user_id);
```

5.8 ER from PostgreSQL Database

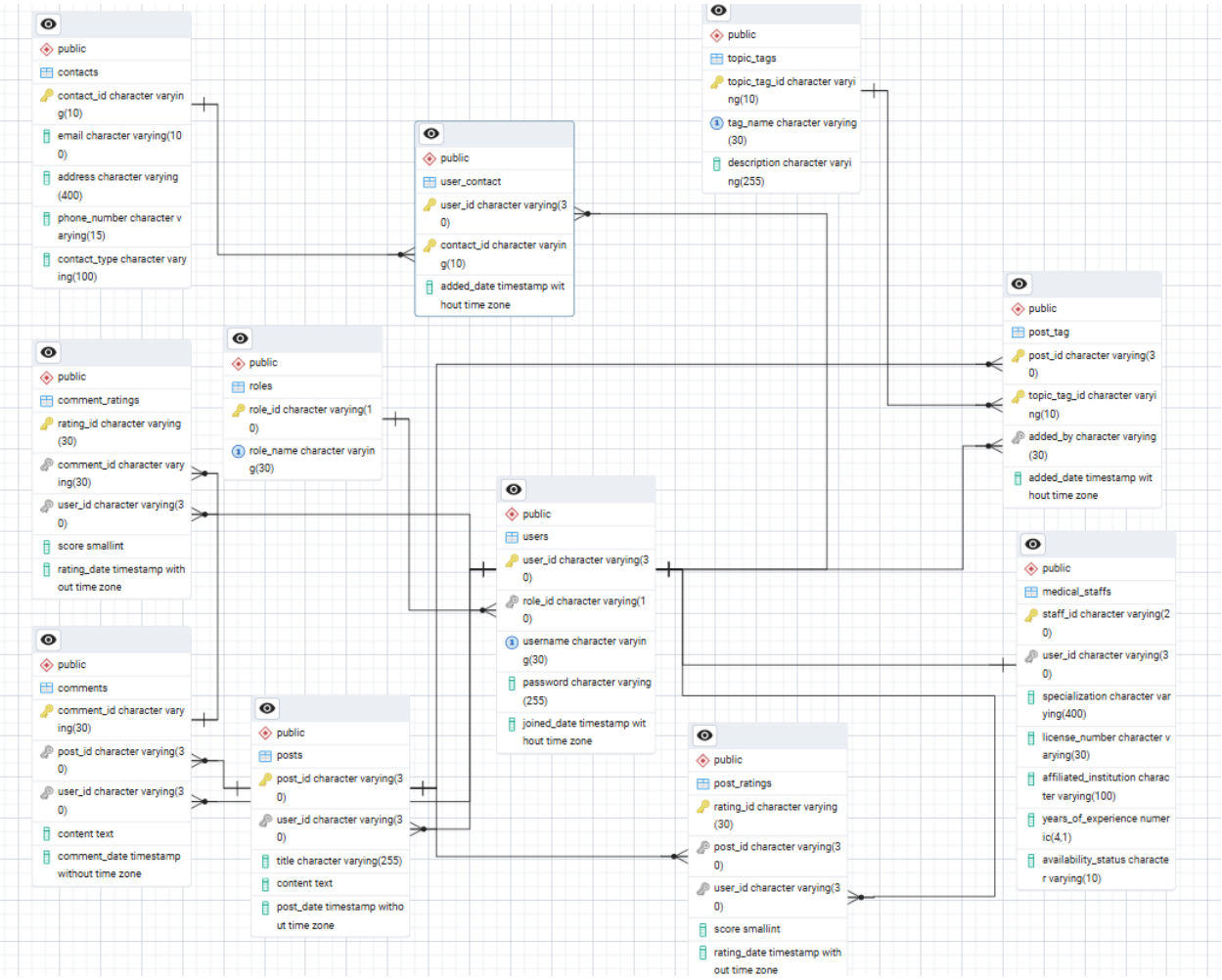

Figure 3: Full ERD for DiabetesConnect from PostgreSQL