Tables & Normalization Analysis

1. `users` (3NF)

sql
Copy

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Normalization**:
  - **1NF**: Atomic fields (no repeating groups).
  - **3NF**: No transitive dependencies (e.g., `username` → `email` isn't valid here).
  - **BCNF**: Every determinant is a candidate key (`user_id` is the PK).

2. `interests` (3NF)

sql
Copy

```
CREATE TABLE interests (
    interest_id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL -- e.g., "CSE", "Web Dev"
);
```

- **Normalization**:
  - Eliminates multi-valued dependency (users can have multiple interests).

3. `user_interests` (BCNF)

sql
Copy

```
CREATE TABLE user_interests (
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    interest_id INT REFERENCES interests(interest_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, interest_id)
);
```

- **Normalization**:
  - Solves multi-valued dependency (1 user → many interests).

---

4. `posts` (3NF)

sql
Copy

```sql
CREATE TABLE posts (
    post_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    type VARCHAR(20) CHECK (type IN ('QUERY', 'ACHIEVEMENT')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Normalization**:
    - **1NF**: No composite attributes.
    - **3NF**: `user_id` → `username` is handled via FK.

---

5. `tags` (3NF)

sql
Copy

```sql
CREATE TABLE tags (
    tag_id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL -- e.g., "Java", "Algorithms"
);
```

6. `post_tags` (BCNF)

sql
Copy

```sql
CREATE TABLE post_tags (
    post_id INT REFERENCES posts(post_id) ON DELETE CASCADE,
    tag_id INT REFERENCES tags(tag_id) ON DELETE CASCADE,
    PRIMARY KEY (post_id, tag_id)
);
```

- **Normalization**:
    - Eliminates repeating groups (1 post → many tags).

---

7. `contests` (3NF)

sql
Copy

```sql
CREATE TABLE contests (
    contest_id SERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL
);
```

8. `problems` (3NF)

sql
Copy

```
CREATE TABLE problems (
    problem_id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT NOT NULL,
    test_cases JSONB NOT NULL -- Input/output pairs
);
```

9. `contest_problems` (BCNF)

sql

Copy

```
CREATE TABLE contest_problems (
    contest_id INT REFERENCES contests(contest_id) ON DELETE CASCADE,
    problem_id INT REFERENCES problems(problem_id) ON DELETE CASCADE,
    PRIMARY KEY (contest_id, problem_id)
);
```

- **Normalization**:
    - Solves many-to-many relationship without redundancy.

---

10. `submissions` (3NF)

sql

Copy

```
CREATE TABLE submissions (
    submission_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    problem_id INT REFERENCES problems(problem_id) ON DELETE CASCADE,
    code TEXT NOT NULL,
    language VARCHAR(50) NOT NULL,
    score INT CHECK (score BETWEEN 0 AND 100),
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Normalization**:
    - **2NF**: All attributes depend on the composite key (`user_id` + `problem_id` + `submitted_at`? No – `submission_id` is PK).

---

11. `badges` (3NF)

sql

Copy

```
CREATE TABLE badges (
    badge_id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    criteria TEXT NOT NULL -- e.g., "Solve 50 problems"
);
```

12. `user_badges` (BCNF)

sql
Copy

```
CREATE TABLE user_badges (
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    badge_id INT REFERENCES badges(badge_id) ON DELETE CASCADE,
    earned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, badge_id)
);
```

---

13. `messages` (3NF)

sql
Copy

```
CREATE TABLE messages (
    message_id SERIAL PRIMARY KEY,
    sender_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    receiver_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_read BOOLEAN DEFAULT FALSE
);
```

Functional Dependencies & Normalization Proof

Example: `submissions` Table

- **FDs**:
  - `submission_id` → `user_id`, `problem_id`, `code`, `score`
  - `user_id` → (no direct dependency on other fields)
  - `problem_id` → (no direct dependency on other fields)
- **Normalization**:
  - **3NF**: No transitive dependencies (all non-key attributes depend only on the PK `submission_id`).

Example: `user_badges` Table

- **FDs**:
  - `(user_id, badge_id)` → `earned_at`
- **BCNF**: The determinant (`user_id`, `badge_id`) is the candidate key.

Denormalization Considerations

- **Leaderboards**:

    - Store precomputed rankings in a `rankings` table (denormalized for performance).
- **Activity Feed**:
    - Use materialized views for personalized feeds (e.g., combining `posts` + `post_tags` + `user_interests`).

Key Normalization Decisions

1. **Splitting Tags/Interests**:
    - Avoided storing arrays (`TEXT[]`) in favor of junction tables (`post_tags`, `user_interests`).
2. **Contest-Problem Relationship**:
    - Separated `problems` from `contests` to reuse problems across contests.
3. **Achievement System**:
    - Decoupled badges from users via `user_badges` to allow scalable criteria.