**1. users Table**

This table stores the basic user details.

| **id** | **first\_name** | **last\_name** | **user\_name** | **email** | **password** |
| --- | --- | --- | --- | --- | --- |
| 1 | John | Doe | johndoe | john@example.com | hashed\_pw |
| 2 | Alice | Smith | alicesmith | alice@example.com | hashed\_pw |
| 3 | Bob | Brown | bobbrown | bob@example.com | hashed\_pw |

* The id is auto-generated (BIGSERIAL).
* user\_name and email are unique.

**2. authorities Table**

This table stores user roles.

| **id** | **authority** |
| --- | --- |
| 1 | ROLE\_USER |
| 2 | ROLE\_ADMIN |
| 3 | ROLE\_MANAGER |

* The authority column stores roles as strings (VARCHAR(50)) due to the EnumType.STRING mapping in the Authority entity.
* Each role is unique.

**3. user\_authority\_mapping Table (Join Table)**

This table manages the many-to-many relationship between users and roles.

| **user\_id** | **authority\_id** |
| --- | --- |
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 3 | 3 |

* This table allows each user to have multiple roles.
* Each role can be assigned to multiple users.
* user\_id and authority\_id reference users(id) and authorities(id), respectively.

Here are the SQL CREATE TABLE queries for the users, authorities, and user\_authority\_mapping tables based on your entities:

sql

CopyEdit

CREATE TABLE users (

id BIGSERIAL PRIMARY KEY,

first\_name VARCHAR(255) NOT NULL,

last\_name VARCHAR(255) NOT NULL,

user\_name VARCHAR(255) UNIQUE NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL

);

CREATE TABLE authorities (

id BIGSERIAL PRIMARY KEY,

authority VARCHAR(50) UNIQUE NOT NULL

);

CREATE TABLE user\_authority\_mapping (

user\_id BIGINT NOT NULL,

authority\_id BIGINT NOT NULL,

PRIMARY KEY (user\_id, authority\_id),

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,

FOREIGN KEY (authority\_id) REFERENCES authorities(id) ON DELETE CASCADE

);

### Explanation:

* **users table**: Stores user details with id as the primary key.
* **authorities table**: Stores roles (ROLE\_USER, ROLE\_ADMIN, ROLE\_MANAGER) as an ENUM-like structure.
* **user\_authority\_mapping table**: A join table to manage the many-to-many relationship between users and authorities.

### **Example: Adding a New User with Roles**

Assume we add a new user **Charlie Wilson** (charliew) with roles **ROLE\_USER** and **ROLE\_MANAGER**:

#### **1. Insert into** users **Table**

sql

CopyEdit

INSERT INTO users (first\_name, last\_name, user\_name, email, password)

VALUES ('Charlie', 'Wilson', 'charliew', 'charlie@example.com', 'hashed\_pw');

(Assuming Charlie gets *id = 4*.)

#### **2. Insert into** user\_authority\_mapping **Table**

sql

CopyEdit

INSERT INTO user\_authority\_mapping (user\_id, authority\_id) VALUES (4, 1); -- ROLE\_USER

INSERT INTO user\_authority\_mapping (user\_id, authority\_id) VALUES (4, 3); -- ROLE\_MANAGER

### **How This Structure Helps**

✅ **Normalized Data**: No duplicate role names in the users table.  
✅ **Scalability**: More roles can be added in the authorities table without modifying user records.  
✅ **Efficient Queries**: Easily fetch users with their roles using JOIN.

### **Fetching User Roles (Example Query)**

sql

CopyEdit

SELECT u.user\_name, a.authority

FROM users u

JOIN user\_authority\_mapping ua ON u.id = ua.user\_id

JOIN authorities a ON ua.authority\_id = a.id

WHERE u.user\_name = 'johndoe';

**Result:**

diff

CopyEdit

user\_name | authority

---------------------

johndoe | ROLE\_USER

johndoe | ROLE\_ADMIN