NOTES ON MANY TO MANY RELATION IN DB:

Sure, here are the notes on creating database tables and entity classes for a many-to-many relationship:

1. If you have a many-to-many relationship between two tables in the database, you need to create a pivot table to represent this relationship. The pivot table will contain the primary keys of both tables, forming a composite key that uniquely identifies each relationship entry.

Example for a many-to-many relationship between "Student" and "Course" tables:

```sql
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE Course (
    course_id INT PRIMARY KEY,
    title VARCHAR(100)
);

CREATE TABLE Student_Course (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

2. When creating entity classes in your Java application, you will need to represent the many-to-many relationship by having a list property of the associated entity type in each class.

Example for entity classes representing the above database tables:

```java
@Entity
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long studentId;

    private String name;

    @ManyToMany
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
```

```
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private List<Course> courses;

    // Constructors, getters, setters, etc.
}

@Entity
@Table(name = "courses")
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long courseId;

    private String title;

    @ManyToMany(mappedBy = "courses")
    private List<Student> students;

    // Constructors, getters, setters, etc.
}
```

In JPA (Java Persistence API), the pivot table is represented through annotations in the entity classes using the `@ManyToMany`, `@JoinTable`, and `@JoinColumn` annotations. The relationship between entities is handled directly through these annotations, and JPA takes care of the underlying database operations, including handling the pivot table.

So, in summary, to handle a many-to-many relationship between two tables, create a pivot table in the database, set up a list property in each entity class to represent the relationship, and use JPA annotations to define the relationship between the entities. No separate class is needed for the pivot table.

Example of a social media application where users can have a many-to-many relationship with other users. In this scenario, we'll represent the "friendship" relationship between users.

1. Database Tables:

```sql
CREATE TABLE User (
    user_id INT PRIMARY KEY,
    username VARCHAR(50)
);

CREATE TABLE Friendship (
    user1_id INT,
    user2_id INT,
    PRIMARY KEY (user1_id, user2_id),
    FOREIGN KEY (user1_id) REFERENCES User(user_id),
```

```
    FOREIGN KEY (user2_id) REFERENCES User(user_id)
);
```

2. Entity Classes using Hibernate Annotations:

User Entity:
```java
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    private String username;

    @ManyToMany
    @JoinTable(
        name = "friendship",
        joinColumns = @JoinColumn(name = "user1_id"),
        inverseJoinColumns = @JoinColumn(name = "user2_id")
    )
    private List<User> friends;

    public User() {
        // Default constructor required by Hibernate
    }

    public User(String username) {
        this.username = username;
        this.friends = new ArrayList<>();
    }

    // Getters, setters, etc.
}
```

3. Using Hibernate to Save and Retrieve Data:

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
```

```
public class Main {
    public static void main(String[] args) {
        // Create Hibernate SessionFactory
        SessionFactory sessionFactory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(User.class)
                .buildSessionFactory();

        // Create some users
        User user1 = new User("John");
        User user2 = new User("Alice");
        User user3 = new User("Bob");

        // Establish friendships (Many-to-Many)
        user1.getFriends().add(user2);
        user1.getFriends().add(user3);

        user2.getFriends().add(user1);
        user2.getFriends().add(user3);

        user3.getFriends().add(user1);
        user3.getFriends().add(user2);

        // Save the data into the database
        try (Session session = sessionFactory.getCurrentSession()) {
            Transaction transaction = session.beginTransaction();

            session.save(user1);
            session.save(user2);
            session.save(user3);

            transaction.commit();
        }

        // Retrieve data from the database
        try (Session session = sessionFactory.getCurrentSession()) {
            Transaction transaction = session.beginTransaction();

            User retrievedUser = session.get(User.class, 1L); // Assuming ID 1 exists
            System.out.println("Retrieved User: " + retrievedUser.getUsername());
            System.out.println("Friends: " + retrievedUser.getFriends());

            transaction.commit();
        }

        // Close the session factory when done
        sessionFactory.close();
    }
}
```

This example demonstrates how to create the entities for users and their friendships using Hibernate annotations. We establish a many-to-many relationship between users through the "Friendship" table,

which represents the friendships between users. The application allows users to have multiple friends, and these friendships are bidirectional (i.e., if User A is friends with User B, then User B is also friends with User A). Hibernate takes care of managing the relationships and persisting data in the database.