1. Entity Relationships:

- **One-to-Many or Many-to-One Relationships**: In this scenario, the `Employee` table should have foreign keys referring to both the `Department` and `Role` tables. The relationships between these entities are typically represented as `@ManyToOne` in the `Employee` entity, meaning each employee can belong to one department and have one role.
- **JPA's** `@ManyToOne` annotation is used to map these relationships, establishing a foreign key link between `Employee`, `Department`, and `Role` tables.

2. Database Design:

- The database should have tables for `employees`, `departments`, and `roles` with relationships:
    - The `employees` table will have a foreign key linking to the `departments` and `roles` tables. `departments` and `roles` will be separate tables with a primary key for each department and role.

3. JPA Repository:

- With Spring Data JPA, you can create repositories for each entity (Employee, Department, and Role) that extend `JpaRepository`. This allows you to easily perform CRUD operations without writing custom queries.
- For retrieving the data, you can simply use the `findById` method provided by JPA repositories to get an employee's information along with the associated department and role.

4. Fetching Data:

- In a single API call, when you fetch an `Employee` by its ID, JPA will use the defined relationships to automatically join the related `Department` and `Role` tables.
- By using `@ManyToOne` relationships, JPA will automatically perform SQL joins behind the scenes to fetch the related department and role when you retrieve an employee.

5. DTO (Data Transfer Object):

- To return data in the desired format, it's a good practice to use a DTO that consolidates employee, department, and role information. This separates the API response from the entity model, making it cleaner and more maintainable. In the response, you can include fields such as the employee’s name, email, department name, and role name.

6. Error Handling: If an employee doesn't exist, you can return a 404 Not Found response.

- In case the token or authentication fails (if implemented), return a 401 Unauthorized response.
- Ensure to check if the related department and role exist and handle errors accordingly (e.g., invalid department or role ID).