Assignment 3: SQL Queries

These queries are written to operate on the `tasks` and `users` tables.

1. Count tasks by status (completed, pending)

This query counts the number of tasks based on their completion status.

```sql
SELECT
    CASE
        WHEN completed = TRUE THEN 'Completed'
        ELSE 'Pending'
    END AS status,
    COUNT(*) AS task_count
FROM
    tasks
GROUP BY
    status;
```

2. List users with no assigned tasks

This query identifies users who do not have any tasks assigned to them.

```sql
SELECT
    u.name,
    u.email
FROM
    users u
LEFT JOIN
    tasks t ON u.id = t.user_id
WHERE
    t.user_id IS NULL;
```

3. Find the most recently updated task

This query retrieves the single task that has the most recent updated_at timestamp.

```sql
SELECT
    id,
    title,
    updated_at
FROM
    tasks
ORDER BY
    updated_at DESC
LIMIT 1;
```

4. Join two tables (Task and User)

This query joins the tasks and users tables to display which user is assigned to which task.

```sql
SELECT
    t.title AS task_title,
    t.description,
    t.completed,
    u.name AS assigned_to
FROM
    tasks t
JOIN
    users u ON t.user_id = u.id;
```

Assignment 4: Detailed Answers (Theory)

1. Difference Between @Component, @Service, and @Repository

All three are Spring Stereotype Annotations used to define Spring-managed beans, but they serve different purposes:

@Component

Purpose: Generic annotation for any Spring-managed bean.

Usage: When a class does not fit into @Service or @Repository.

Example: Utility classes, helper components.

@Service

Purpose: Indicates a business service layer class.

Usage: Contains business logic, transactions, and service orchestration.

Example:@Service

```
public class UserService {
    public User getUserById(Long id) { ... }
}
```

@Repository

Purpose: Indicates a Data Access Object (DAO) class.

Usage: Interacts with databases (e.g., JPA, JDBC, MyBatis).

Special Feature: Automatically translates database exceptions into Spring’s DataAccessException.

Example:@Repository

```
public class UserRepository {
    public User findById(Long id) { ... }
}
```

2. Spring Boot Auto-Configuration

Spring Boot automatically configures the application based on:

- Dependencies in pom.xml (e.g., spring-boot-starter-data-jpa configures Hibernate).

- Properties in application.properties/application.yml.

3. MyBatis vs Hibernate

MyBatis

Type: SQL Mapper (not full ORM).

Control: Developers write SQL queries manually (XML/Annotations).

Use Case:

- Fine-grained control over SQL.

- Better for complex queries.

Example:<select id="getUser" resultType="User">

```
   SELECT * FROM users WHERE id = #{id}
</select>
```

Hibernate

Type: Full ORM (Object-Relational Mapping).

Control: Uses HQL (Hibernate Query Language) or JPA Criteria API.

Features:

- Automatic SQL generation.

- Caching, lazy loading, dirty checking.

Example:@Entity

```
public class User { ... }
```

4. Dependency Injection in Spring Boot

What is Dependency Injection (DI)?

- A design pattern where objects receive dependencies instead of creating them.

- Spring IoC Container manages beans and injects dependencies.

5. CORS in Spring Boot

What is CORS?

- Cross-Origin Resource Sharing (CORS) allows web apps from different domains to access APIs.
- Prevents unauthorized cross-domain requests (security feature).

+ Global Configuration

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {
   @Override
   public void addCorsMappings(CorsRegistry registry) {
      registry.addMapping("/api/**")
         .allowedOrigins("http://example.com")
         .allowedMethods("GET", "POST", "PUT");
   }
}
```

+ Controller-Level (Using @CrossOrigin)

```
@RestController
@CrossOrigin(origins = "http://example.com")
@RequestMapping("/api")
public class UserController { ... }
```

+ Security Config (If Using Spring Security)

```
http.cors().configurationSource(request -> new CorsConfiguration().applyPermitDefaultValues());
```

6. Securing a REST API

+ Authentication (Who Are You?)

Basic Auth (Not recommended for production).

JWT (JSON Web Tokens) – Stateless, scalable.

OAuth2 (e.g., Google, Facebook login).

Spring Security (Username + Password).

+ Authorization (What Can You Do?)

- Role-Based Access Control (RBAC) (ADMIN, USER).

- Method-Level Security:

```
@PreAuthorize("hasRole('ADMIN')")
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable Long id) { ... }
```

Example: Spring Security + JWT

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
      .csrf().disable()
      .authorizeRequests()
        .requestMatchers("/public/**").permitAll()
        .requestMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated()
      .and()
      .addFilter(new JwtAuthenticationFilter(authenticationManager()));
    return http.build();
  }
}
```