Purpose:
Marks a Java class as a **JPA entity**, meaning it represents a table in your database.

Example:

```
@Entity
public class User {
    // fields, getters, setters...
}
```

Details:

- Every entity class must have:
    - A **no-argument constructor** (can be `protected` or `public`).
    - A **primary key field** annotated with `@Id`.
- If you don't specify a table name using `@Table`, the class name is used as the table name by default (`User` → `user` table).
- The class must be registered in your JPA persistence context (automatically handled in Spring Boot).

`@Table`

Purpose:
Specifies the **name of the database table** that the entity is mapped to.

Example:

```
@Entity
@Table(name = "users")
public class User {
    // fields
}
```

Details:

- Optional — if you omit it, the table name defaults to the entity class name.

- You can also specify:
    - **schema** (if your DB uses multiple schemas)
    - **uniqueConstraints**
    - **indexes**

Example with additional attributes:

```
@Table(
    name = "users",
    schema = "public",
    uniqueConstraints = @UniqueConstraint(columnNames = "email")
)
```

`@Id`

Purpose:
Marks a field as the **primary key** of the entity (the unique identifier for each row).

Example:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

Details:

- Required for every entity — each entity must have exactly **one `@Id` field**.
- Usually combined with `@GeneratedValue` to auto-generate the primary key.

Common Generation Strategies:

Strategy	Description
`AUTO`	Hibernate picks a strategy automatically (default).
`IDENTITY`	Uses auto-increment column in DB (MySQL/PostgreSQL).

`SEQUENCE`	Uses a sequence object (common in Oracle).
`TABLE`	Uses a separate table to generate IDs.

`@Column`

Purpose:
Maps a Java field to a **specific database column** and allows customization of column properties.

Example:

```
@Column(name = "user_name", nullable = false, unique = true, length = 50)
private String username;
```

Common Attributes:

Attribute	Description
`name`	Specifies the column name (default = field name).
`nullable`	Allows/disallows `NULL` values.
`unique`	Adds a unique constraint.
`length`	Specifies max column length (useful for `VARCHAR`).
`precision`, `scale`	Used for numeric columns (e.g., `BigDecimal`).
`columnDefinition`	Allows custom SQL definition for the column.

Putting It All Together

Here's a simple **example entity** that uses all four annotations:

```
import jakarta.persistence.*;
```

```
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "user_name", nullable = false, unique = true,
length = 50)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(name = "email_address", unique = true)
    private String email;

    // Constructors, getters, and setters
}
```

What happens here:

- `User` → Mapped to the table `users`.
- `id` → Primary key with auto-increment.
- `username`, `password`, `email` → Each mapped to their own database columns.
- Hibernate automatically generates the table (if `spring.jpa.hibernate.ddl-auto=update` is set).

Summary Table

Annotation	Purpose	Example
`@Entity`	Marks the class as a database entity	`@Entity public class User {}`

`@Table`	Specifies the table name and settings	`@Table(name="users")`
`@Id`	Marks primary key	`@Id private Long id;`
`@Column`	Configures column properties	`@Column(name="user_name", nullable=false)`