### JPA Annotations Overview

JPA provides several annotations that define how Java objects are mapped to relational database tables. These annotations make it easier to manage data and specify the relationships, identifiers, and table columns within an entity. Below are some of the key JPA annotations, along with explanations and examples.

---

### 1. \*\*`@Entity`\*\*

- \*\*Purpose\*\*: Marks a class as a persistent entity, meaning that its objects will be stored in a database table.

- \*\*Placement\*\*: Applied at the class level.

#### Example:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Getters and setters

}

```

- In this example, the `User` class is marked as an entity, which means it is a table in the database.

---

### 2. \*\*`@Table`\*\*

- \*\*Purpose\*\*: Specifies the table name in the database that the entity is mapped to. If not provided, the entity class name is used as the table name by default.

- \*\*Placement\*\*: Applied at the class level, used in conjunction with `@Entity`.

#### Example:

```java

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Getters and setters

}

```

- This maps the `User` entity to a table named `users` in the database.

---

### 3. \*\*`@Id`\*\*

- \*\*Purpose\*\*: Denotes the primary key of the entity.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Getters and setters

}

```

- The `id` field is marked as the primary key for the `User` entity.

---

### 4. \*\*`@GeneratedValue`\*\*

- \*\*Purpose\*\*: Specifies how the primary key value is automatically generated. It is used in conjunction with `@Id`.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Getters and setters

}

```

- \*\*`strategy = GenerationType.IDENTITY`\*\*: Uses the database's identity column feature to auto-increment the primary key value.

#### Generation Strategies:

- \*\*`AUTO`\*\*: Default, lets JPA pick the appropriate strategy based on the database.

- \*\*`IDENTITY`\*\*: Relies on an auto-increment column in the database.

- \*\*`SEQUENCE`\*\*: Uses a database sequence to generate the primary key.

- \*\*`TABLE`\*\*: Uses a special table to store and generate primary key values.

---

### 5. \*\*`@Column`\*\*

- \*\*Purpose\*\*: Specifies the mapping between a field and a column in the database. If not used, JPA assumes the field name matches the column name.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "user\_name", nullable = false, length = 100)

private String name;

// Getters and setters

}

```

- \*\*`name = "user\_name"`\*\*: Maps the `name` field to the `user\_name` column.

- \*\*`nullable = false`\*\*: Specifies that the column cannot have `null` values.

- \*\*`length = 100`\*\*: Specifies the maximum length for the column.

---

### 6. \*\*`@Temporal`\*\*

- \*\*Purpose\*\*: Specifies the format of date or time fields. It can be used with `java.util.Date` or `java.util.Calendar` fields.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Temporal(TemporalType.DATE)

private Date birthDate;

// Getters and setters

}

```

- \*\*`TemporalType.DATE`\*\*: Maps the `Date` field to only store the date (without time).

- \*\*`TemporalType.TIME`\*\*: Stores only the time.

- \*\*`TemporalType.TIMESTAMP`\*\*: Stores both date and time.

---

### 7. \*\*`@Enumerated`\*\*

- \*\*Purpose\*\*: Maps an enumeration (enum) field to a database column. By default, JPA stores the ordinal value of the enum.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

public enum Status {

ACTIVE, INACTIVE;

}

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Enumerated(EnumType.STRING)

private Status status;

// Getters and setters

}

```

- \*\*`EnumType.STRING`\*\*: Stores the enum name (`ACTIVE`, `INACTIVE`) as a `VARCHAR`.

- \*\*`EnumType.ORDINAL`\*\*: Stores the ordinal value of the enum (0 for `ACTIVE`, 1 for `INACTIVE`).

---

### 8. \*\*`@Lob`\*\*

- \*\*Purpose\*\*: Maps large object (LOB) fields to the corresponding database column type. It can store large text or binary data.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class Document {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Lob

private String content;

// Getters and setters

}

```

- The `content` field will be stored as a large object (e.g., `TEXT` or `BLOB`) in the database.

---

### 9. \*\*`@Transient`\*\*

- \*\*Purpose\*\*: Specifies that a field is not persistent and should not be mapped to the database.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@Transient

private String tempPassword;

// Getters and setters

}

```

- The `tempPassword` field is transient and will not be saved in the database.

---

### 10. \*\*`@ManyToOne`, `@OneToMany`, `@OneToOne`, `@ManyToMany`\*\*

- \*\*Purpose\*\*: Specifies the relationships between entities (associations).

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example of `@ManyToOne`:

```java

@Entity

public class Post {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

@ManyToOne

@JoinColumn(name = "user\_id")

private User user;

// Getters and setters

}

```

- \*\*`@ManyToOne`\*\*: Indicates a many-to-one relationship (many `Post` entities belong to one `User`).

- \*\*`@JoinColumn(name = "user\_id")`\*\*: Specifies the foreign key column that refers to the `User`.

#### Other Relationship Annotations:

- \*\*`@OneToMany`\*\*: One-to-many relationship, typically mapped using a `List` or `Set`.

- \*\*`@OneToOne`\*\*: One-to-one relationship between two entities.

- \*\*`@ManyToMany`\*\*: Many-to-many relationship between two entities, often requiring a join table.

---

### 11. \*\*`@JoinColumn`\*\*

- \*\*Purpose\*\*: Specifies the foreign key column in a relationship.

- \*\*Placement\*\*: Applied at the field or getter method level.

#### Example:

```java

@Entity

public class Post {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

@ManyToOne

@JoinColumn(name = "user\_id")

private User user;

// Getters and setters

}

```

- \*\*`name = "user\_id"`\*\*: Specifies the column in the `Post` table that stores the foreign key for the `User` entity.

---

### Summary

JPA annotations play a vital role in defining how Java objects map to relational database structures. They help developers focus on object-oriented design without needing to write SQL queries directly. Understanding these annotations allows developers to create robust and maintainable applications that interact seamlessly with relational databases.