

Hibernate JPA Annotations - Contents:

Annotation	Package Detail/Import statement
@Entity	import javax.persistence.Entity;
@Table	import javax.persistence.Table;
@Column	import javax.persistence.Column;
@Id	import javax.persistence.Id;
@GeneratedValue	import javax.persistence.GeneratedValue;
@Version	import javax.persistence.Version;
@OrderBy	import javax.persistence.OrderBy;
@Transient	import javax.persistence.Transient;
@Lob	import javax.persistence.Lob;
Hibernate Association Mapping Annotations	
@OneToOne	import javax.persistence.OneToOne;
@ManyToOne	import javax.persistence.ManyToOne;
@OneToMany	import javax.persistence.OneToMany;
@ManyToMany	import javax.persistence.ManyToMany;
@PrimaryKeyJoinColumn	import javax.persistence.PrimaryKeyJoinColumn;
@JoinColumn	import javax.persistence.JoinColumn;
@JoinTable	import javax.persistence.JoinTable;
@MapsId	import javax.persistence.Id;
Hibernate Inheritance Mapping Annotations	
@Inheritance	import javax.persistence.Inheritance;
@DiscriminatorColumn	import javax.persistence.DiscriminatorColumn;
@DiscriminatorValue	import javax.persistence.DiscriminatorValue;

@Entity

Annotate all your entity beans with @Entity.

```
1  @Entity
2  public class Company implements Serializable {
3      ...
4  }
```

@Table

Specify the database table this Entity maps to using the name attribute of @Table annotation. In the example below, the data will be stored in 'company' table in the database.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4      ...
5  }
```

@Column

Specify the column mapping using @Column annotation.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5      @Column(name = "name")
6      private String name;
7
8      ...
9  }
```

@Id

Annotate the id column using @Id.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5      @Id
6      @Column(name = "id")
7      private int id;
8  }
```

```
9 | ...
10 | }
```

@GeneratedValue

Let database generate (auto-increment) the id column.

```
1 | @Entity
2 | @Table(name = "company")
3 | public class Company implements Serializable {
4 |
5 |     @Id
6 |     @Column(name = "id")
7 |     @GeneratedValue
8 |     private int id;
9 |
10 | ...
11 | }
```

@Version

Control versioning or concurrency using @Version annotation.

```
1 | @Entity
2 | @Table(name = "company")
3 | public class Company implements Serializable {
4 |
5 |     @Version
6 |     @Column(name = "version")
7 |     private Date version;
8 |
9 | ...
10 | }
```

@OrderBy

Sort your data using @OrderBy annotation. In example below, it will sort all contacts in a company by their firstname in ascending order.

```
1 | @OrderBy("firstName asc")
2 | private Set contacts;
```

@Transient

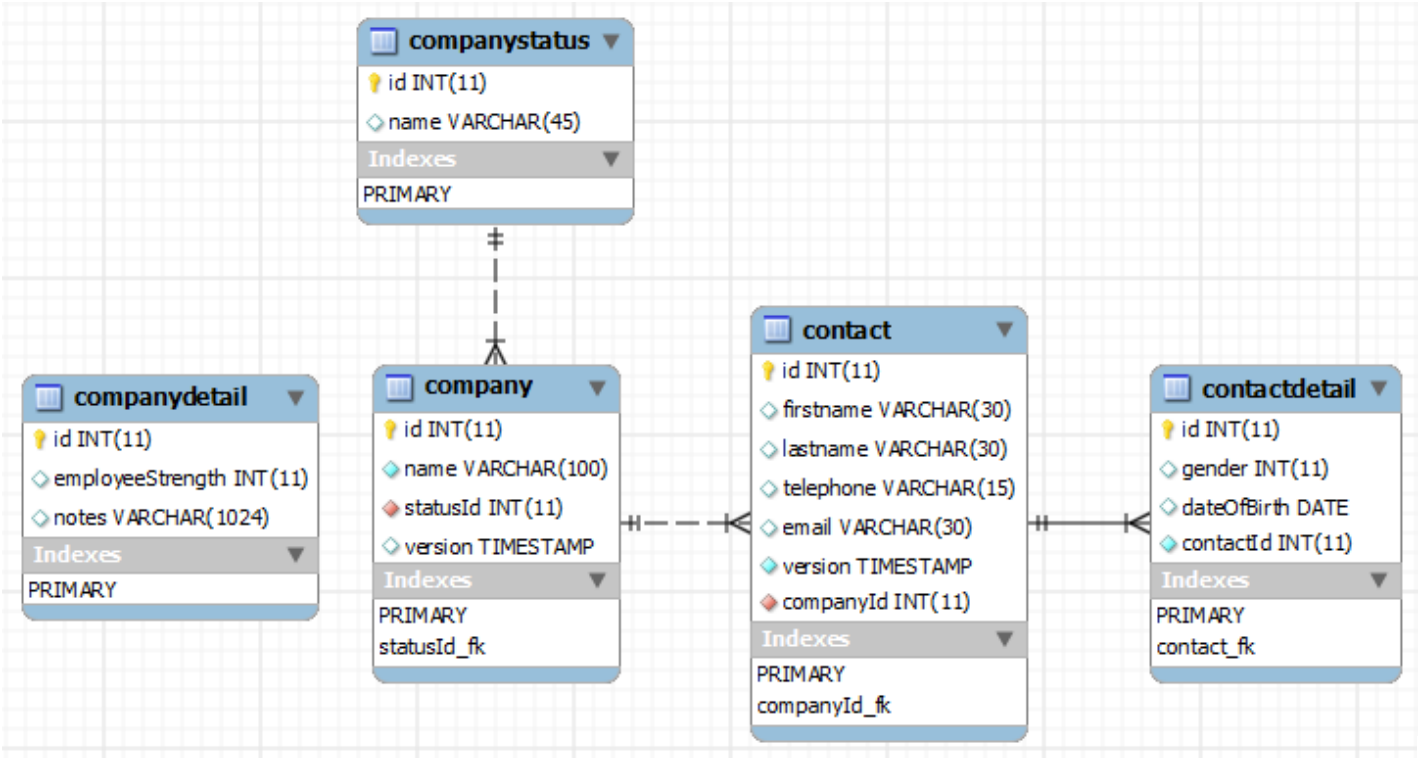
Annotate your transient properties with @Transient.

@Lob

Annotate large objects with @Lob.

Hibernate Association Mapping Annotations

Example App DB Schema



The database for this tutorial is designed to illustrate various association mapping concepts. In RDBMS implementations, entities are joined using the following ways:

- Shared Primary Key
- Foreign Key
- Association Table

In our example app,

- Tables company and companyDetail have shared values for primary key. It is a one-to-one assoication.
- Tables contact and contactDetail are linked through a foreign key. It is also a one to one association.
- Tables contact and company are linked through a foriegn key in many-to-one association with contact being the owner.
- Tables company and companyStatus are linked through a foreign key in many-to-one association with company being the owner.

@OneToOne



- Use `@PrimaryKeyJoinColumn` for associated entities sharing the same primary key.
- Use `@JoinColumn` & `@OneToOne` mappedBy attribute when foreign key is held by one of the entities.
- Use `@JoinTable` and mappedBy entities linked through an association table.
- Persist two entities with shared key using `@MapsId`

For entities `Company` and `CompanyDetail` sharing the same primary key, we can associate them using `@OneToOne` and `@PrimaryKeyJoinColumn` as shown in the example below.

Notice that the `id` property of `CompanyDetail` is NOT annotated with `@GeneratedValue`. It will be populated by `id` value of `Company`.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5      @Id
6      @Column(name = "id")
7      @GeneratedValue
8      private int id;
9
10     @OneToOne(cascade = CascadeType.MERGE)
11     @PrimaryKeyJoinColumn
12     private CompanyDetail companyDetail;
13
14     ...
15 }
16
17 @Entity
18 @Table(name = "companyDetail")
19 public class CompanyDetail implements Serializable {
20
21     @Id
22     @Column(name = "id")
23     private int id;
24
25     ...
26 }
```

For entities `Contact` and `ContactDetail` linked through a foreign key, we can use `@OneToOne` and `@JoinColumn` annotations. In example below, the `id` generated for `Contact` will be mapped to 'contact_id' column of `ContactDetail` table. Please note the usage of `@MapsId` for the same.

```
1  @Entity
2  @Table(name = "contactDetail")
3  public class ContactDetail implements Serializable {
4
5      @Id
6      @Column(name = "id")
7      @GeneratedValue
8      private int id;
9
10     @OneToOne
11     @MapsId
12     @JoinColumn(name = "contactId")
13     private Contact contact;
14
15     ...
16 }
17
18 @Entity
19 @Table(name = "contact")
20 public class Contact implements Serializable {
21
22     @Id
23     @Column(name = "ID")
24     @GeneratedValue
25     private Integer id;
26
27     @OneToOne(mappedBy = "contact", cascade = CascadeType.ALL)
28     private ContactDetail contactDetail;
29
30     ....
31 }
```

Also note that the relationship between `Company` and `CompanyDetail` is uni-directional. On the other hand, the relationship between `Contact` and `ContactDetail` is bi-directional and that can be achieved using 'mappedBy' attribute.

The rationale to have one relationship as uni-directional and other as bi-directional in this tutorial is to illustrate both concepts and their usage. You can opt for uni-directional or bi-directional relationships to suit your needs.

@ManyToOne



- Use `@JoinColumn` when foreign key is held by one of the entities.
- Use `@JoinTable` for entities linked through an association table.

The two examples below illustrate many-to-one relationships. `Contact` to `Company` and `Company` to `CompanyStatus`. Many contacts can belong to a company. Similarly many companies can share the same status (Lead, Prospect, Customer) - there will be many companies

that are currently leads.

```
1  @Entity
2  @Table(name = "contact")
3  public class Contact implements Serializable {
4
5      @ManyToOne
6      @JoinColumn(name = "companyId")
7      private Company company;
8
9      ...
10
11 }
12
13 @Entity
14 @Table(name = "company")
15 public class Company implements Serializable {
16
17     @ManyToOne
18     @JoinColumn(name = "statusId")
19     private CompanyStatus status;
20
21     ...
22
23 }
```

@OneToMany



- Use mappedBy attribute for bi-directional associations with ManyToOne being the owner.
- OneToMany being the owner or unidirectional with foreign key - try to avoid such associations but can be achieved with @JoinColumn
- @JoinTable for Unidirectional with association table

Please see the many-to-one relationship between Contact and Company above. Company to Contact will be a one-to-many relationship. The owner of this relationship is Contact and hence we will use 'mappedBy' attribute in Company to make it bi-directional relationship.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5      @OneToMany(mappedBy = "company", fetch = FetchType.EAGER)
6      @OrderBy("firstName asc")
7      private Set contacts;
8
9      ...
10
11 }
```

Again, for this tutorial, we have kept Company to CompanyStatus relationship as uni-directional.

@ManyToMany



- Use @JoinTable for entities linked through an association table.
- Use mappedBy attribute for bi-directional association.

@PrimaryKeyJoinColumn

@PrimaryKeyJoinColumn annotation is used for associated entities sharing the same primary key. See [OneToOne](#) section for details.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5      @Id
6      @Column(name = "id")
7      @GeneratedValue
8      private int id;
9
10     @OneToOne(cascade = CascadeType.MERGE)
11     @PrimaryKeyJoinColumn
12     private CompanyDetail companyDetail;
13
14     ...
15 }
```

@JoinColumn

Use @JoinColumn annotation for one-to-one or many-to-one associations when foreign key is held by one of the entities. We can use @OneToOne or @ManyToOne mappedBy attribute for bi-directional relations. Also see [OneToOne](#) and [ManyToOne](#) sections for more details.

```
1  @ManyToOne
2  @JoinColumn(name = "statusId")
3  private CompanyStatus status;
```

@JoinTable

Use @JoinTable and mappedBy for entities linked through an association table.

@MapsId

Persist two entities with shared key (when one entity holds a foreign key to the other) using @MapsId annotation. See [OneToOne](#) section for details.

```
1 | @OneToOne
2 | @MapsId
3 | @JoinColumn(name = "contactId")
4 | private Contact contact;
```

Hibernate Inheritance Mapping Annotations

To understand Inheritance Mapping annotations, you must first understand [Inheritance Mapping in Hiberate](#) in detail. Once you understand Inheritance mapping concepts, please review below for annotations to be used.

- table per class hierarchy - single table per Class Hierarchy Strategy: the <subclass> element in Hibernate

```
1 | @Entity
2 | @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
3 | @DiscriminatorColumn(name="planetype", discriminatorType=DiscriminatorType.STRING )
4 |
5 | @DiscriminatorValue("Plane")
6 | public class Plane { ... }
7 |
8 | @Entity
9 | @DiscriminatorValue("A320")
10 | public class A320 extends Plane { ... }
```

- table per class/subclass - joined subclass Strategy: the <joined-subclass> element in Hibernate

```
1 | @Entity
2 | @Inheritance(strategy=InheritanceType.JOINED)
3 | public class Boat implements Serializable { ... }
4 |
5 | @Entity
6 | @PrimaryKeyJoinColumn
7 | public class Ferry extends Boat { ... }
```

- table per concrete class - table per Class Strategy: the <union-class> element in Hibernate

```
1 | @Entity
2 | @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
3 | public class Flight implements Serializable { ... }
```



Note: This strategy does not support the IDENTITY generator strategy: the id has to be shared across several tables. Consequently, when using this strategy, you should not use AUTO nor IDENTITY.

@Inheritance

See [Hibernate Inheritance Mapping Annotations](#) section for details.

```
1 | @Entity
2 | @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
```

@DiscriminatorColumn

See [Hibernate Inheritance Mapping Annotations](#) section for details.

```
1 | @Entity
2 | @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
3 | @DiscriminatorColumn(name="planetype", discriminatorType=DiscriminatorType.STRING )
```

@DiscriminatorValue

See [Hibernate Inheritance Mapping Annotations](#) section for details.

```
1 | @Entity
2 | @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
3 | @DiscriminatorColumn(name="planetype", discriminatorType=DiscriminatorType.STRING )
4 |
5 | @DiscriminatorValue("Plane")
6 | public class Plane { ... }
7 |
8 | @Entity
9 | @DiscriminatorValue("A320")
10 | public class A320 extends Plane { ... }
```