DataNucleus

JPA Annotations Reference (v5.1)

Table of Contents

JPA Class-Level Annotations 2

- @Entity 3
- @MappedSuperclass 3
- @Embeddable 3
- @IdClass 4
- @Cacheable 4
- @EntityListeners 4
- @NamedQuery 5
- @NamedNativeQuery 5
- @NamedStoredProcedureQuery 6
- @SqlResultSetMapping 7
- @NamedEntityGraph 7
- @Converter 8
- @Inheritance 8
- @Table 9
- @SecondaryTable 9
- @DiscriminatorColumn 10
- @DiscriminatorValue 10
- @PrimaryKeyJoinColumn 11
- @AttributeOverride 12
- @AttributeOverride 12
- @AssociationOverride 13
- @SequenceGenerator 13
- @TableGenerator 14

JPA Field-Level Annotations 15

- @PrePersist 17
- @PostPersist 17
- @PreRemove 17
- @PostRemove 18
- @PreUpdate 18
- @PostUpdate 19
- @PostLoad 19
- @Id 20
- @Embedded 20
- @EmbeddedId 20
- @Version 21
- @Basic 21
- @Transient 22

@JoinTable 22
@CollectionTable 23
@Lob 23
@Temporal 24
@Enumerated 24
@OneToOne 25
@OneToMany 26
@ManyToMany 26
@ManyToOne 27
@ElementCollection 28
@GeneratedValue 28
@MapKey 29
@MapKeyClass 29
@MapKeyTemporal 30
@MapKeyEnumerated 30
@MapKeyColumn 31
@OrderBy 31
@OrderColumn 31
@Convert 32
@Column 33
@JoinColumn 33
@UniqueConstraint 34
@Index 35
@ForeignKey 35
@MapsId 35
DataNucleus Class-Level Extensions 36
@PersistenceAware 36
@DatastoreId 36
@NonDurableId 37
@ReadOnly 37
@MultiTenant 38
@SoftDelete 38
DataNucleus Field-Level Extensions 39
@SharedRelation 39
@ValueGenerator 40
@ReadOnly 40
@CreateTimestamp 41
@CreateUser 41
@UpdateTimestamp 41
@UpdateUser 42
@Index (field/method - extension) 42

@JdbcType 42
@SqlType 43
@ColumnPosition 43
@Extension 44
Meta-Annotations 45

JPA provides the ability to use annotations to define the persistence of entities, and DataNucleus JPA supports both JPA and JDO annotations. In this section we provide a reference to the primary JPA annotations. When selecting to use annotations please bear in mind the following :-

- You must have the `datanucleus-api-jpa` jar available in your CLASSPATH.
- You must have the `javax.persistence` jar in your CLASSPATH since this provides the annotations
- Annotations should really only be used for attributes of persistence that you won't be changing at deployment. Things such as table and column names shouldn't really be specified using annotations although it is permitted. Instead it would be better to put such information in an ORM file.
- Annotations can be added in two places - for the class as a whole, or for a field in particular.
- You can annotate fields or getters with field-level information. It doesn't matter which.
- Annotations are prefixed by the @ symbol and can take attributes (in brackets after the name, comma-separated)
- JPA doesn't provide for some key JDO concepts and DataNucleus provides its own annotations for these cases.
- You have to import `javax.persistence.XXX` where XXX is the annotation name of a JPA annotation
- You have to import `org.datanucleus.api.jpa.annotations.XXX` where XXX is the annotation name of a DataNucleus value-added annotation

Annotations supported by DataNucleus are shown below. Not all have their documentation written yet.

JPA Class-Level Annotations

The following annotations are specified at class-level and are JPA standard. Using these provide portability for your application.

Annotation	Class/Field	Description
@Entity	Class	Specifies that the class is persistent
@MappedSuperclass	Class	Specifies that this class contains persistent information to be mapped
@Embeddable	Class	Specifies that the class is persistent embedded in another persistent class
@IdClass	Class	Defines the primary key class for this class
@Cacheable	Class	Specifies that instances of this class can be cached in the L2 cache
@EntityListeners	Class	Specifies class(es) that are listeners for events from instances of this class
@NamedQuery	Class	Defines a named JPQL query for use in the current persistence unit
@NamedNativeQuery	Class	Defines a named SQL query for use in the current persistence unit
@NamedStoredProcedureQuery	Class	Defines a named stored procedure query for use in the current persistence unit
@SqlResultSetMapping	Class	Defines a result mapping for an SQL query for use in the current persistence unit
@NamedEntityGraph	Class	Defines a named entity graph with root of the class it is specifed on
@Converter	Class	Defines a java type converter for a field type
@Inheritance	Class	Specifies the inheritance model for persisting this class
@Table	Class	Defines the table where this class will be stored
@SecondaryTable	Class	Defines a secondary table where some fields of this class will be stored
@DiscriminatorColumn	Class	Defines the column where any discriminator will be stored
@DiscriminatorValue	Class	Defines the value to be used in the discriminator for objects of this class
@PrimaryKeyJoinColumn	Class	Defines the name of the PK column when this class has a superclass
@AttributeOverride	Class	Defines a field in a superclass that will have its column overridden

Annotation	Class/Field	Description
@AssociationOverride	Class	Defines a N-1/1-1 field in a superclass that will have its column overridden
@SequenceGenerator	Class/Field/Method	Defines a generator of values using sequences in the datastore for use with persistent entities
@TableGenerator	Class/Field/Method	Defines a generator of sequences using a table in the datastore for use with persistent entities

@Entity

This annotation is used when you want to mark a class as persistent. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the entity (used in JPQL to refer to the class)	

```
@Entity
public class MyClass
{
    ...
}
```

See the documentation for Class Mapping

@MappedSuperclass

This annotation is used when you want to mark a class as persistent but without a table of its own and being the superclass of the class that has a table, meaning that all of its fields are persisted into the table of its subclass. Specified on the **class**.

```
@MappedSuperclass
public class MyClass
{
    ...
}
```

See the documentation for Inheritance

@Embeddable

This annotation is used when you want to mark a class as persistent and only storable embedded in another object. Specified on the **class**.

```
@Embeddable
public class MyClass
{
    ...
}
```

@IdClass

This annotation is used to define a primary-key class for the identity of this class. Specified on the **class**.

Attribute	Type	Description	Default
value	Class	Identity class	

```
@Entity
@IdClass(mydomain.samples.MyIdentity.class)
public class MyClass
{
    ...
}
```

See the documentation for Primary Keys

@Cacheable

This annotation is used when you want to mark a class so that instance of that class can be cached. Specified on the **class**.

```
@Cacheable
public class MyClass
{
    ...
}
```

See the documentation for L2 Cache

@EntityListeners

This annotation is used to define a class or classes that are listeners for events from instances of this class. Specified on the **class**.

Attribute	Type	Description	Default
value	Class[]	Entity listener class(es)	

```
@Entity
@EntityListeners(mydomain.MyListener.class)
public class MyClass
{
    ...
}
```

See the documentation for Lifecycle Callbacks

@NamedQuery

This annotation is used to define a named (JPQL) query that can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Symbolic name for the query. The query will be referred to under this name	
query	String	The JPQL query	

```
@Entity
@NamedQuery(name="AllPeople", query="SELECT p FROM Person p")
public class Person
{
    ...
}
```

Note that with DataNucleus you can also specify @NamedQuery on non-persistable classes See the documentation for Named Queries

There is a `@NamedQueries` annotation but you can achieve the same cleaner using multiple `@NamedQuery` annotations.

@NamedNativeQuery

This annotation is used to define a named (SQL) query that can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Symbolic name for the query. The query will be referred to under this name	
query	String	The SQL query	
resultClass	Class	Class into which the result rows will be placed	void.class

```
@Entity
@NamedNativeQuery(name="PeopleCalledSmith", query="SELECT * FROM PERSON WHERE SURNAME
= 'Smith'")
public class Person
{
    ...
}
```

Note that with DataNucleus you can also specify @NamedNativeQuery on non-persistable classes See the documentation for Named Native Queries

There is a `@NamedNativeQueries` annotation but you can achieve the same cleaner using multiple `@NamedNativeQuery` annotations.

@NamedStoredProcedureQuery

This annotation is used to define a named stored procedure query that can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Symbolic name for the query. The query will be referred to under this name	
procedureName	String	Name of the stored procedure in the datastore	
parameters	StoredProcedureParameter[]	Any parameter definitions for this stored procedure	
resultClasses	Class[]	Any result class(es) for this stored procedure (one per result set)	
resultSetMappings	Class[]	Any result set mapping(s) for this stored procedure (one per result set)	
hints	QueryHint[]	Any query hints for this stored procedure	

```
@Entity
@NamedStoredProcedureQuery(name="MyProc", procedureName="MY_PROC_SP1",
       parameters={@StoredProcedureParameter(name="PARAM1", mode=ParameterMode.IN,
type=String.class)})
public class Person
{
    ...
}
```

Note that with DataNucleus you can also specify `@NamedStoredProcedureQuery` on non-persistable classes See the documentation for Named StoredProcedures

There is a `@NamedStoredProcedureQueries` annotation but you can achieve the same cleaner using multiple `@NamedStoredProcedureQuery` annotations.

@SqlResultSetMapping

This annotation is used to define a mapping for the results of an SQL query and can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Symbolic name for the mapping. The mapping will be referenced under this name	
entities	EntityResult[]	Set of entities extracted from the SQL query	
columns	ColumnResult[]	Set of columns extracted directly from the SQL query	

```
@Entity
@SqlResultSetMapping(name="PEOPLE_PLUS_AGE",
    entities={@EntityResult(entityClass=Person.class)}, columns={@ColumnResult(name
="AGE")})
public class Person
{
    ...
}
```

There is a `@SqlResultSetMappings` annotation but you can achieve the same cleaner using multiple `@SqlResultSetMapping` annotations.

@NamedEntityGraph

This annotation is used to define a named EntityGraph and can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
name	String	name for the Entity Graph.	
attributeNodes	AttributeNode[]	Set of nodes in this EntityGraph	

```
@Entity
@NamedEntityGraph(name="PERSON_FULL",
    attributeNodes={@NamedAttributeNode(name="friends"), @NamedAttributeNode(name
="parents")})
public class Person
{
    ...
}
```

There is a `@NamedEntityGraphs` annotation but you can achieve the same cleaner using multiple `@NamedEntityGraph` annotations.

@Converter

This annotation is used to mark a class as being an attribute converter. *Note that DataNucleus doesn't require this specifying against a converter class except if you want to set the "autoApply".* Specified on the **class**.

Attribute	Type	Description	Default
autoApply	boolean	Whether this converter should always be used when storing this java type	false

```
@Converter
public class MyConverter
{
    ...
}
```

@Inheritance

This annotation is used to define the inheritance persistence for this class. Specified on the **class**.

Attribute	Type	Description	Default
strategy	InheritanceType	Inheritance strategy	**SINGLE_TABLE**, JOINED, TABLE_PER_CLASS

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class MyClass
{
    ...
}
```

See the documentation for Inheritance

@Table

This annotation is used to define the table where objects of a class will be stored. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the table	
catalog	String	Name of the catalog	
schema	String	Name of the schema	
uniqueConstraints	UniqueConstraint[]	Any unique constraints to apply to the table	
indexes	Index[]	Details of indexes if wanting to override provider default	

```
@Entity
@Table(name="MYTABLE", schema="PUBLIC")
public class MyClass
{
    ...
}
```

@SecondaryTable

This annotation is used to define a secondary table where some fields of this class are stored in another table. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the table	
catalog	String	Name of the catalog	
schema	String	Name of the schema	
pkJoinColumns	PrimaryKeyJoinColumns[]	Join columns for the PK of the secondary table back to the primary table	

Attribute	Type	Description	Default
uniqueConstraints	UniqueConstraint[]	Any unique constraints to apply to the table	
indexes	Index[]	Details of indexes if wanting to override provider default	
foreignKey	ForeignKey	Foreign key details if wanting to override provider default	

```
@Entity
@Table(name="MYTABLE", schema="PUBLIC")
@SecondaryTable(name="MYOTHERTABLE", schema="PUBLIC", columns={@PrimaryKeyJoinColumn
(name="MYCLASS_ID")})
public class MyClass
{
    ...
}
```

See the documentation for Secondary Tables

@DiscriminatorColumn

This annotation is used to define the discriminator column for a class. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the discriminator column	DTYPE
discriminatorType	DiscriminatorType	Type of the discriminator column	STRING, CHAR, INTEGER
length	String	Length of the discriminator column	31

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="OBJECT_TYPE", discriminatorType=DiscriminatorType.STRING)
public class MyClass
{
    ...
}
```

See the documentation for Inheritance

@DiscriminatorValue

This annotation is used to define the value to be stored in the discriminator column for a class (when used). Specified on the **class**.

Attribute	Type	Description	Default
value	String	Value for the discriminator column	

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="OBJECT_TYPE", discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("MyClass")
public class MyClass
{
    ...
}
```

See the documentation for Inheritance

@PrimaryKeyJoinColumn

This annotation is used to define the name of the primary key column when this class has a superclass. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the column	
referencedColumnName	String	Name of the associated PK column in the superclass. This is for use when you have a composite PK so acts as a way of aligning the respective columns. **It is not to allow joining to some non-PK column**	
columnDefinition	String	DDL to use for the column (everything except the column name). This must include the SQL type of the column	
foreignKey	ForeignKey	Foreign key details if wanting to override provider default	

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
@PrimaryKeyJoinColumn(name="PK_FIELD_1")
public class MyClass
{
    ...
}
```

There is a `@PrimaryKeyJoinColumns` annotation but you can achieve the same more cleanly with multiple `@PrimaryKeyJoinColumn` annotations.

@AttributeOverride

This annotation is used to define a field of a superclass that has its column overridden. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the field	
column	Column	Column information	

```
@Entity
@AttributeOverride(name="attr", column=@Column(name="NEW_NAME"))
public class MyClass extends MySuperClass
{
    ...
}
```

There is also an `@AttributeOverrides` annotation but you can achieve the same cleaner using multiple `@AttributeOverride` annotations.

@AttributeOverride

This annotation is used to define a field of an embedded class that has its column overridden. Specified on the **field/property**.

Attribute	Type	Description	Default
name	String	Name of the field	
column	Column	Column information	

```
@Entity
public class MyClass extends MySuperClass
{
    @Embedded
    @AttributeOverride(name="attr", column=@Column(name="NEW_NAME"))
    MyEmbeddedType embedded;
    ...
}
```

There is also an `@AttributeOverrides` annotation but you can achieve the same cleaner using multiple `@AttributeOverride` annotations.

@AssociationOverride

This annotation is used to define a 1-1/N-1 field of a superclass that has its column overridden. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the field	
joinColumn	JoinColumn	Column information for the FK column	

```
@Entity
@AssociationOverride(name="friend", joinColumn=@JoinColumn(name="FRIEND_ID"))
public class Employee extends Person
{
    ...
}
```

There is also an `@AssociationOverrides` annotation but you can achieve the same cleaner using multiple `@AssociationOverride` annotations.

@SequenceGenerator

This annotation is used to define a generator using sequences in the datastore. It is scoped to the persistence unit. Specified on the **class/field/method**.

Attribute	Type	Description	Default
name	String	Name for the generator (required)	
sequenceName	String	Name of the underlying sequence that will be used	
initialValue	int	Initial value for the sequence (optional)	1
allocationSize	int	Number of values to be allocated each time (optional)	50
schema	String	Name of the schema where the sequence will be stored (optional)	
catalog	String	Name of the catalog where the sequence will be stored (optional)	

```
@Entity
@SequenceGenerator(name="MySeq", sequenceName="SEQ_2")
public class MyClass
{
    ...
}
```

@TableGenerator

This annotation is used to define a generator using a table in the datastore for storing the values. It is scoped to the persistence unit. Specified on the **class/field/method**.

Attribute	Type	Description	Default
name	String	Name for the generator (required)	
table	String	Name of the table to use	SEQUENCE_TABLE
catalog	String	Catalog of the table to use (optional)	
schema	String	Schema of the table to use (optional)	
pkColumnName	String	Name of the primary key column for the table	SEQUENCE_NAME
valueColumnName	String	Name of the value column for the table	NEXT_VAL
pkColumnValue	String	Value to store in the PK column for the row used by this generator	{name of the class}
initialValue	int	Initial value for the table row (optional)	0
allocationSize	int	Number of values to be allocated each time (optional)	50
indexes	Index[]	Index(es) if wanting to override the provider default	

```
@Entity
@TableGenerator(name="MySeq", table="MYAPP_IDENTITIES", pkColumnValue="MyClass")
public class MyClass
{
    ...
}
```

JPA Field-Level Annotations

The following annotations are specified at field/method-level and are JPA standard. Using these provide portability for your application.

Annotation	Class/Field	Description
@SequenceGenerator	Class/Field/Method	Defines a generator of values using sequences in the datastore for use with persistent entities
@TableGenerator	Class/Field/Method	Defines a generator of sequences using a table in the datastore for use with persistent entities
@Embedded	Field/Method	Defines this field as being embedded
@AttributeOverride	Class	Defines a field in an embedded class that will have its column overridden
@Id	Field/Method	Defines this field as being (part of) the identity for the class
@EmbeddedId	Field/Method	Defines this field as being (part of) the identity for the class, and being embedded into this class.
@Version	Field/Method	Defines this field as storing the version for the class
@Basic	Field/Method	Defines this field as being persistent
@Transient	Field/Method	Defines this field as being transient (not persisted)
@OneToOne	Field/Method	Defines this field as being a 1-1 relation with another persistent entity
@OneToMany	Field/Method	Defines this field as being a 1-N relation with other persistent entities
@ManyToMany	Field/Method	Defines this field as being a M-N relation with other persistent entities
@ManyToOne	Field/Method	Defines this field as being a N-1 relation with another persistent entity
@ElementCollection	Field/Method	Defines this field as being a 1-N relation of Objects that are not Entities.
@GeneratedValue	Field/Method	Defines that this field has its value generated using a generator
@MapKey	Field/Method	Defines that this field is the key to a map
@MapKeyClass	Field/Method	Defines that the key type for the map in this field
@MapKeyEnumerated	Field/Method	Defines the datastore type for the map key when it is an enum
@MapKeyTemporal	Field/Method	Defines the datastore type for the map key when it is a temporal type

Annotation	Class/Field	Description
@MapKeyColumn	Field/Method	Defines the column details for the map key when stored in a join table
@OrderBy	Field/Method	Defines the field(s) used for ordering the elements in this collection
@OrderColumn	Field/Method	Defines that ordering should be attributed by the implementation using a surrogate column.
@PrePersist	Field/Method	Defines this method as being a callback for pre-persist events
@PostPersist	Field/Method	Defines this method as being a callback for post-persist events
@PreRemove	Field/Method	Defines this method as being a callback for pre-remove events
@PostRemove	Field/Method	Defines this method as being a callback for post-remove events
@PreUpdate	Field/Method	Defines this method as being a callback for pre-update events
@PostUpdate	Field/Method	Defines this method as being a callback for post-update events
@PostLoad	Field/Method	Defines this method as being a callback for post-load events
@JoinTable	Field/Method	Defines this field as being stored using a join table
@CollectionTable	Field/Method	Defines this field as being stored using a join table when containing non-entity objects.
@Lob	Field/Method	Defines this field as being stored as a large object
@Temporal	Field/Method	Defines this field as storing temporal data
@Enumerated	Field/Method	Defines this field as storing enumerated data
@Convert	Field/Method	Defines a converter for this field/property
@Column	Field/Method	Defines the column where this field is stored
@JoinColumn	Field/Method	Defines a column for joining to either a join table or foreign key relation
@Index	-	Defines the details of an index when overriding the provider default.
@ForeignKey	-	Defines the details of a foreign key when overriding the provider default.
@MapsId	Field/Method	Defines that this field maps one part of the id of the overall class. **NOT SUPPORTED**.

@PrePersist

This annotation is used to define a method that is a callback for pre-persist events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PrePersist
    void registerObject()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PostPersist

This annotation is used to define a method that is a callback for post-persist events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PostPersist
    void doSomething()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PreRemove

This annotation is used to define a method that is a callback for pre-remove events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PreRemove
    void registerObject()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PostRemove

This annotation is used to define a method that is a callback for post-remove events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PostRemove
    void doSomething()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PreUpdate

This annotation is used to define a method that is a callback for pre-update events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PreUpdate
    void registerObject()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PostUpdate

This annotation is used to define a method that is a callback for post-update events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PostUpdate
    void doSomething()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks.

@PostLoad

This annotation is used to define a method that is a callback for post-load events. Specified on the **method**. It has no attributes.

```
@Entity
public class MyClass
{
    ...

    @PostLoad
    void registerObject()
    {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@Id

This annotation is used to define a field to use for the identity of the class. Specified on the **field/method**.

```
@Entity
public class MyClass
{
    @Id
    long id;
    ...
}
```

@Embedded

This annotation is used to define a field as being embedded. Specified on the **field/method**.

```
@Entity
public class MyClass
{
    @Embedded
    Object myField;
    ...
}
```

@EmbeddedId

This annotation is used to define a field to use for the identity of the class when embedded. Specified on the **field/method**.

```
@Entity
public class MyClass
{
    @EmbeddedId
    MyPrimaryKey pk;
    ...
}
```

@Version

This annotation is used to define a field as holding the version for the class. Specified on the **field/method**.

```
@Entity
public class MyClass
{
    @Id
    long id;

    @Version
    int ver;
    ...
}
```

@Basic

This annotation is used to define a field of the class as persistent. Specified on the **field/method**.

Attribute	Type	Description	Default
fetch	FetchType	Type of fetching for this field	LAZY, **EAGER**
optional	boolean	Whether this field having a value is optional (can it have nulls)	**true**, false

```
@Entity
public class Person
{
    @Id
    long id;

    @Basic(optional=false)
    String forename;
    ...
}
```

See the documentation for Fields/Properties

@Transient

This annotation is used to define a field of the class as not persistent. Specified on the **field/method**.

```
@Entity
public class Person
{
    @Id
    long id;

    @Transient
    String personalInformation;
    ...
}
```

See the documentation for Fields/Properties

@JoinTable

This annotation is used to define that a collection/map is stored using a join table. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name of the table	
catalog	String	Name of the catalog	
schema	String	Name of the schema	
joinColumns	JoinColumn[]	Columns back to the owning object (with the collection/map)	
inverseJoinColumns	JoinColumn[]	Columns to the element object (stored in the collection/map)	
uniqueConstraints	UniqueConstraint[]	Any unique constraints to apply to the table	
indexes	Index[]	Details of indexes if wanting to override provider default	
foreignKey	ForeignKey	Foreign key details if wanting to override provider default for the join columns	
inverseForeignKey	ForeignKey	Foreign key details if wanting to override provider default for the inverse join columns	

```
@Entity
public class Person
{
    @OneToMany
    @JoinTable(name="PEOPLES_FRIENDS")
    Collection friends;
    ...
}
```

@CollectionTable

This annotation is used to define that a collection/map of non-entities is stored using a join table. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name of the table	
catalog	String	Name of the catalog	
schema	String	Name of the schema	
joinColumns	JoinColumn[]	Columns back to the owning object (with the collection/map)	
uniqueConstraints	UniqueConstraint[]	Any unique constraints to apply to the table	
indexes	Index[]	Details of indexes if wanting to override provider default	
foreignKey	ForeignKey	Details of foreign key if wanting to override provider default	

```
@Entity
public class Person
{
    @ElementCollection
    @CollectionTable(name="PEOPLES_FRIENDS")
    Collection<String> values;
    ...
}
```

@Lob

This annotation is used to define that a field will be stored using a large object in the datastore. Specified on the **field/method**.

```
@Entity
public class Person
{
    @Lob
    byte[] photo;
    ...
}
```

@Temporal

This annotation is used to define that a field is stored as a temporal type. It specifies the JDBC type to use for storage of this type, so whether it stores the date, the time, or both. Specified on the **field/method**.

Attribute	Type	Description	Default
value	TemporalType	Type for storage	DATE, TIME, TIMESTAMP

```
@Entity
public class Person
{
    @Temporal(TemporalType.TIMESTAMP)
    java.util.Date dateOfBirth;
    ...
}
```

@Enumerated

This annotation is used to define that a field is stored enumerated (not that it wasn't obvious from the type!). Specified on the **field/method**.

Attribute	Type	Description	Default
value	EnumType	Type for storage	**ORDINAL**, STRING

```
enum Gender {MALE, FEMALE};

@Entity
public class Person
{
    @Enumerated
    Gender gender;
    ...
}
```

@OneToOne

This annotation is used to define that a field represents a 1-1 relation. Specified on the **field/method**.

Attribute	Type	Description	Default
targetEntity	Class	Class at the other side of the relation	
fetch	FetchType	Whether the field should be fetched immediately	**EAGER**, LAZY
optional	boolean	Whether the field can store nulls.	**true**, false
mappedBy	String	Name of the field that owns the relation (specified on the inverse side). If the field that owns the relation is stored in an embedded object on the other side, use DOT notation to identify it.	
cascade	CascadeType[]	Whether persist, update, delete, refresh operations are cascaded	
orphanRemoval	boolean	Whether to remove orphans when either removing this side of the relation or when nulling the relation	true, **false**

```
@Entity
public class Person
{
    @OneToOne
    Person bestFriend;
    ...
}
```

See the documentation for 1-1 Relations

@OneToMany

This annotation is used to define that a field represents a 1-N relation. Specified on the **field/method**.

Attribute	Type	Description	Default
targetEntity	Class	Class at the other side of the relation	
fetch	FetchType	Whether the field should be fetched immediately	EAGER, **LAZY**
mappedBy	String	Name of the field that owns the relation (specified on the inverse side). If the field that owns the relation is stored in an embedded object on the other side, use DOT notation to identify it.	
cascade	CascadeType[]	Whether persist, update, delete, refresh operations are cascaded	
orphanRemoval	boolean	Whether to remove orphans when either removing this side of the relation or when nulling the relation and removing an element	true, **false**

```
@Entity
public class Person
{
    @OneToMany
    Collection<Person> friends;
    ...
}
```

See the documentation for 1-N Relations

@ManyToMany

This annotation is used to define that a field represents a M-N relation. Specified on the **field/method**.

Attribute	Type	Description	Default
targetEntity	Class	Class at the other side of the relation	
fetch	FetchType	Whether the field should be fetched immediately	EAGER, **LAZY**

Attribute	Type	Description	Default
mappedBy	String	Name of the field on the non-owning side that completes the relation. Specified on the owner side. If the field that owns the relation is stored in an embedded object on the other side, use DOT notation to identify it.	
cascade	CascadeType[]	Whether persist, update, delete, refresh operations are cascaded	

```
@Entity
public class Customer
{
    @ManyToMany(mappedBy="customers")
    Collection<Supplier> suppliers;
    ...
}

@Entity
public class Supplier
{
    @ManyToMany
    Collection<Customer> customers;
    ...
}
```

See the documentation for M-N Relations

@ManyToOne

This annotation is used to define that a field represents a N-1 relation. Specified on the **field/method**.

Attribute	Type	Description	Default
targetEntity	Class	Class at the other side of the relation	
fetch	FetchType	Whether the field should be fetched immediately	**EAGER**, LAZY
optional	boolean	Whether the field can store nulls.	**true**, false
cascade	CascadeType[]	Whether persist, update, delete, refresh operations are cascaded	

```
@Entity
public class House
{
    @OneToMany(mappedBy="house")
    Collection<Window> windows;
    ...
}

@Entity
public class Window
{
    @ManyToOne
    House house;
    ...
}
```

See the documentation for N-1 Relations

@ElementCollection

This annotation is used to define that a field represents a 1-N relation to non-entity objects. Specified on the **field/method**.

Attribute	Type	Description	Default
targetClass	Class	Class at the other side of the relation	
fetch	FetchType	Whether the field should be fetched immediately	EAGER, **LAZY**

```
@Entity
public class Person
{
    @ElementCollection
    Collection<String> values;
    ...
}
```

@GeneratedValue

This annotation is used to define the generation of a value for a (PK) field. Specified on the **field/method**.

Attribute	Type	Description	Default
strategy	GenerationType	Strategy to use when generating the values for this field. Has possible values of GenerationType TABLE, SEQUENCE, IDENTITY, AUTO, UUID. Note that UUID is only available if using the DN provided `javax.persistence.jar` (v2.2+)	GenerationType.AUTO
generator	String	Name of the generator to use. See @TableGenerator and @SequenceGenerator	

```
@Entity
public class Person
{
    @Id
    @GeneratedValue(strategy=GenerationType.TABLE)
    long id;
    ...
}
```

@MapKey

This annotation is used to define the field in the value class that represents the key in a Map. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name of the field in the value class to use for the key. If no value is supplied and the field is a Map then it is assumed that the key will be the primary key of the value class. DataNucleus only supports this null value treatment if the primary key of the value has a single field.	

```
@Entity
public class Person
{
    @OneToMany
    @MapKey(name="nickname")
    Map<String, Person> friends;
    ...
}
```

@MapKeyClass

This annotation is used to define the key type for a map field when generics have not been specified. Specified on the **field/method**.

Attribute	Type	Description	Default
class	String	Class to be used for the key of the map.	

```
@Entity
public class Person
{
    @OneToMany(targetEntity=Person.class)
    @MapKeyClass(String.class)
    Map friends;
    ...
}
```

Avoid use of this class and use Java generics! This is the 21st century after all

@MapKeyTemporal

This annotation is used to define the datastore type used for the key of a map when it is a temporal type. Specified on the **field/method**.

```
@Entity
public class Person
{
    @ElementCollection
    @MapKeyTemporal(TemporalType.DATE)
    Map<Date, String> dateMap;
    ...
}
```

@MapKeyEnumerated

This annotation is used to define the datastore type used for the key of a map when it is an enum. Specified on the **field/method**.

```
@Entity
public class Person
{
    @ElementCollection
    @MapKeyEnumerated(EnumType.STRING)
    Map<MyEnum, String> dateMap;
    ...
}
```

@MapKeyColumn

This annotation is used to define the column details for a key of a Map when stored in a join table. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name of the column for the key	

```
@Entity
public class Person
{
    @OneToMany
    @MapKeyColumn(name="FRIEND_NAME")
    Map<String, Person> friends;
    ...
}
```

@OrderBy

This annotation is used to define a field in the element class that is used for ordering the elements of the List when it is retrieved. Specified on the **field/method**.

Attribute	Type	Description	Default
value	String	Name of the field(s) in the element class to use for ordering the elements of the List when retrieving them from the datastore. This is used by JPA "ordered lists" as opposed to "indexed lists" (which always return the elements in the same order as they were persisted. The value will be a comma separated list of fields and optionally have ASC/DESC to signify ascending or descending	

```
@Entity
public class Person
{
    @OneToMany
    @OrderBy(value="nickname")
    List<Person> friends;
    ...
}
```

@OrderColumn

This annotation is used to define that the JPA implementation will handle the ordering of the List elements using a surrogate column ("ordered list"). Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name of the column to use.	*{fieldName}_ORDER*
nullable	boolean	Whether the column is nullable	**true**, false
insertable	boolean	Whether the column is insertable	**true**, false
updatable	boolean	Whether the column is updatable	**true**, false
base	int	Base for ordering (not currently supported)	0

```
@Entity
public class Person
{
    @OneToMany
    @OrderColumn
    List<Person> friends;
    ...
}
```

@Convert

This annotation is used to define a converter for the field/property. Specified on the **field/method**.

Attribute	Type	Description	Default
converter	Class	Converter class	
attributeName	String	"key" if specified on a Map field and converting the key. "value" if specified on a Map field and converting the value. Alternatively the name of the embedded field to be converted (**currently not supported**).	
disableConversion	boolean	Whether we should disable any use of @Converter set to auto-apply	

```
@Entity
public class Person
{
    @Basic
    @Convert(converter=MyURLConverter.class)
    URL website;
    ...
}
```

@Column

This annotation is used to define the column where a field is stored. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name for the column	
unique	boolean	Whether the field is unique	true, **false**
nullable	boolean	Whether the field is nullable	**true**, false
insertable	boolean	Whether the field is insertable	**true**, false
updatable	boolean	Whether the field is updatable	**true**, false
table	String	Name of the table	
length	int	Length for the column	255
precision	int	Decimal precision for the column	0
scale	int	Decimal scale for the column	0
columnDefinition	String	DDL to use for the column (everything except the column name). This must include the SQL type of the column	

```
@Entity
public class Person
{
    @Basic
    @Column(name="SURNAME", length=100, nullable=false)
    String surname;
    ...
}
```

@JoinColumn

This annotation is used to define the FK column for joining to another table. This is part of a 1-1, 1-N, or N-1 relation. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	Name for the column	
referencedColumnName	String	Name of the column in the other table that this is the FK for. This is for use when you have a composite PK so acts as a way of aligning the respective columns. **It is not to allow joining to some non-PK column**	
unique	boolean	Whether the field is unique	true, **false**

Attribute	Type	Description	Default
nullable	boolean	Whether the field is nullable	**true**, false
insertable	boolean	Whether the field is insertable	**true**, false
updatable	boolean	Whether the field is updatable	**true**, false
columnDefinition	String	DDL to use for the column (everything except the column name). This must include the SQL type of the column	
foreignKey	ForeignKey	Foreign key details if wanting to override provider default	

```
@Entity
public class Person
{
    @OneToOne
    @JoinColumn(name="PET_ID", nullable=true)
    Animal pet;
    ...
}
```

There is a `@JoinColumns` annotation but you can achieve the same more cleanly with multiple `@JoinColumn` annotations.

@UniqueConstraint

This annotation is used to define a unique constraint to apply to a table. It is specified as part of `@Table`, `@JoinTable` or `@SecondaryTable`.

Attribute	Type	Description	Default
columnNames	String[]	Names of the column(s)	

```
@Entity
@Table(name="PERSON", uniqueConstraints={@UniqueConstraint(columnNames={"firstName"
,"lastName"})})
public class Person
{
    @Basic
    String firstName;

    @Basic
    String lastName;
    ...
}
```

See the documentation for Unique Constraints

@Index

This annotation is used to define the details for an Index. It is specified as part of `@Table`, `@JoinTable`, `@CollectionTable` or `@SecondaryTable`.

Attribute	Type	Description	Default
name	String	Name of the index	
columnList	String	Columns to be included in this index of the form *colName1, colName2*	
unique	boolean	Whether the index is unique	false

See the documentation for Index Constraints

@ForeignKey

This annotation is used to define the details for a ForeignKey. It is specified as part of `@JoinColumn`, `@JoinTable`, `@CollectionTable` or `@SecondaryTable`.

Attribute	Type	Description	Default
name	String	Name of the foreign key	
value	ConstraintMode	Constraint mode	ConstraintMode.CONSTRAINT
foreignKeyDefinition	String	DDL for the FOREIGN KEY statement of the form *FOREIGN KEY (colExpr1 {, colExpr2}...) REFERENCES tblIdentifier {(otherColExpr1 {, otherColExpr2}...) } { ON UPDATE updateAction } { ON DELETE deleteAction }*	

See the documentation for ForeignKey Constraints

@MapsId

This annotation would be used to mark the current field as mapping on to one of the "id" fields of the current class (embedded-id). **This is not currently supported** and there are better, more efficient ways of handling it.

DataNucleus Class-Level Extensions

The following annotations are specified at class-level and are vendor extensions providing more functionality than the JPA spec defines. Using these will reduce the portability of your application.

Annotation	Class/Field	Description
@PersistenceAware	Class	Specifies that the class is not persistent but needs to be able to access fields of persistent classes (DataNucleus extension).
@DatastoreId	Class	Defines a class as using datastore-identity (DataNucleus extension).
@NonDurableId	Class	Defines a class as using nondurable identity (DataNucleus extension).
@ReadOnly	Class	Specifies that this class is "read-only" (DataNucleus extension).
@MultiTenant	Class	Specifies multi-tenancy details for this class (DataNucleus extension).
@SoftDelete	Class	Specifies that this class will be "soft-deleted" upon deletion of objects (DataNucleus extension).

@PersistenceAware

This annotation is used when you want to mark a class as knowing about persistence but not persistent itself. That is, it manipulates the fields of a persistent class directly rather than using accessors. **This is a DataNucleus extension**. Specified on the **class**.

```
@PersistenceAware
public class MyClass
{
    ...
}
```

See the documentation for Class Mapping

@DatastoreId

This DataNucleus-extension annotation is used to define that the class uses datastore-identity. Specified on the **class**.

Attribute	Type	Description	Default
generationType	GenerationType	Strategy to use when generating the values for this field. Has possible values of GenerationType TABLE, SEQUENCE, IDENTITY, AUTO, UUID. Note that UUID is only available if using the DN provided `javax.persistence.jar` (v2.2+)	GenerationType.AUTO
generator	String	Name of the generator to use. See @TableGenerator and @SequenceGenerator	
column	String	Name of the column for persisting the datastore identity value	

```
@Entity
@DatastoreId(column="MY_ID")
public class MyClass
{
    ...
}
```

@NonDurableId

This DataNucleus-extension annotation is used to define that the class uses non-durable identity. Specified on the **class**.

```
@Entity
@NonDurableId
public class MyClass
{
    ...
}
```

@ReadOnly

This DataNucleus-extension annotation is used to define a class as being read-only (equivalent as read-only="true"). Specified on the **class**.

```
@Entity
@ReadOnly
public class MyClass
{
    ...
}
```

@MultiTenant

This DataNucleus-extension annotation is used specify multi-tenancy details for a class. Specified on the **class**.

Attribute	Type	Description	Default
column	String	Name of the multi-tenancy column for this class.	TENANT_ID
columnLength	int	Length of the multi-tenancy column.	
disabled	boolean	Whether the multi-tenancy for this class is disabled.	false

```
@Entity
@MultiTenant(column="TENANT", columnLength=255)
public class MyClass
{
    ...
}
```

@SoftDelete

This DataNucleus-extension annotation is used to define a class as being soft-deleted whenever objects of this type are removed. Specified on the **class**.

Attribute	Type	Description	Default
column	String	Name of the soft-delete status column for this class.	DELETED

```
@PersistenceCapable
@SoftDelete
public class MyClass
{
    ...
}
```

DataNucleus Field-Level Extensions

The following annotations are specified at field/method-level and are vendor extensions providing more functionality than the JPA spec defines. Using these will reduce the portability of your application.

Annotation	Class/Field	Description
@SharedRelation	Field/Method	Specifies that the relation for this field/property is "shared" (DataNucleus extension).
@ReadOnly	Field/Method	Specifies that this field/property is "read-only" (DataNucleus extension).
@Index	Field/Method	Specifies an index on this field/property (DataNucleus extension).
@JdbcType	Field/Method	Specifies the JDBC Type to use on this field/property (DataNucleus extension).
@SqlType	Field/Method	Specifies the SQL Type to use on this field/property (DataNucleus extension).
@ColumnPosition	Field/Method	Specifies the column position to use on this field/property (DataNucleus extension).
@ValueGenerator	Field/Method	Specifies a non-JPA-standard value generator to use on this field/property (DataNucleus extension).
@Extension	Class/Field/Method	Defines a DataNucleus extension (DataNucleus extension).
@CreateTimestamp	Field/Method	Specifies that this field/property should store a creation timestamp when inserting (DataNucleus extension).
@CreateUser	Field/Method	Specifies that this field/property should store the current user when inserting (DataNucleus extension).
@UpdateTimestamp	Field/Method	Specifies that this field/property should store an update timestamp when updating (DataNucleus extension).
@UpdateUser	Field/Method	Specifies that this field/property should store the current user when updating (DataNucleus extension).

@SharedRelation

This DataNucleus-extension annotation is used to define a field with a (1-N/M-N) relation as being "shared" so that a distinguisher column is added. Specified on the **field/property**.

Attribute	Type	Description	Default
value	String	value to be stored in the distinguisher column for this relation field	
column	String	Name of the distinguisher column for this relation field	

Attribute	Type	Description	Default
primaryKey	boolean	Whether the distinguisher column should be part of the PK (when in a join table)	

```
@Entity
public class MyClass
{
    @OneToMany
    @JoinTable
    @SharedRelation(column="ADDRESS_TYPE", value="home")
    Collection<Address> homeAddresses;

    @OneToMany
    @JoinTable
    @SharedRelation(column="ADDRESS_TYPE", value="work")
    Collection<Address> workAddresses;
    ...
}
```

@ValueGenerator

This DataNucleus-extension annotation is used to allow use of non-JPA-standard value generators on a field/property. Specified on the **field/property**.

Attribute	Type	Description	Default
strategy	String	Name of the strategy e.g "uuid"	

```
@Entity
public class MyClass
{
    @ValueGenerator(strategy="uuid")
    String id;
    ...
}
```

@ReadOnly

This DataNucleus-extension annotation is used to define a field as being read-only (equivalent to `insertable="false", updateable="false"`). Specified on the **field/property**.

```
@Entity
public class MyClass
{
    @Basic
    @ReadOnly
    String someValue;

    ...
}
```

@CreateTimestamp

This DataNucleus-extension annotation is used to define this field as being persisted with a timestamp of the creation time of this object. Specified on the **field/property**.

```
@Entity
public class MyClass
{
    @CreateTimestamp
    Timestamp createTime;
    ...
}
```

@CreateUser

This DataNucleus-extension annotation is used to define this field as being persisted with the current user at insert of this object. Specified on the **field/property**.

```
@Entity
public class MyClass
{
    @CreateUser
    String createUser;
    ...
}
```

@UpdateTimestamp

This DataNucleus-extension annotation is used to define this field as being persisted with a timestamp of the update time of this object. Specified on the **field/property**.

```
@Entity
public class MyClass
{
    @UpdateTimestamp
    Timestamp updateTime;
    ...
}
```

@UpdateUser

This DataNucleus-extension annotation is used to define this field as being persisted with the current user at update of this object. Specified on the **field/property**.

```
@Entity
public class MyClass
{
    @UpdateUser
    String updateUser;
    ...
}
```

@Index (field/method - extension)

This DataNucleus-extension annotation is used to define an index for this field/property. Specified on the **field/property**.

Attribute	Type	Description	Default
name	String	Name of the index	
unique	boolean	Whether the index is unique	false

```
@Entity
public class MyClass
{
    @Index(name="ENABLED_IDX")
    boolean enabled;
    ...
}
```

@JdbcType

This DataNucleus-extension annotation is used to define the jdbc-type to use for this field/property. Specified on the **field/property**.

Attribute	Type	Description	Default
value	String	JDBC Type (VARCHAR, INTEGER, BLOB, etc)	

```
@Entity
public class MyClass
{
    @JdbcType("CHAR")
    boolean enabled;
    ...
}
```

@SqlType

This DataNucleus-extension annotation is used to define the sql-type to use for this field/property. Specified on the **field/property**.

Attribute	Type	Description	Default
value	String	SQL Type (VARCHAR, INTEGER, BLOB, UUID, etc)	

```
@Entity
public class MyClass
{
    @SqlType("CHAR")
    boolean enabled;
    ...
}
```

@ColumnPosition

This DataNucleus-extension annotation is used to define the column position to use for this field/property. Specified on the **field/property**.

Attribute	Type	Description	Default
value	Integer	position of the column (first is "0", increasing)	

```
@Entity
public class MyClass
{
    @ColumnPosition(0)
    boolean enabled;
    ...
}
```

@Extension

DataNucleus Extension Annotation used to define an extension specific to DataNucleus. Specified on the **class** or **field**.

Attribute	Type	Description	Default
vendorName	String	Name of the vendor	datanucleus
key	String	Key for the extension	
value	String	Value of the extension	

```
@Entity
@Extension(key="RunFast", value="true")
public class Person
{
    ...
}
```

There is an `@Extensions` annotation but you can achieve the same cleaner using multiple `@Extension` annotations.

Meta-Annotations

JPA annotations are all usable as part of *meta-annotations*. A *meta-annotation* is, in simple terms, a user-defined annotation that provides one or multiple other annotations (including annotation attributes). Let's provide a couple of examples

Firstly, say we have

```
@Entity
@Cacheable
@MultiTenant(column="TENANT")
```

and need to put this on many classes. We can introduce our own annotation

```
@Target(TYPE)
@Retention(RUNTIME)
@Entity
@Cacheable
@MultiTenant(column="TENANT")
public @interface MultiTenantCacheableEntity
{
}
```

so now we can simply annotate a JPA entity with

```
@MultiTenantCacheableEntity
public class MyClass
{
    ...
}
```

A second example is where we are specifying several attributes on an annotation, such as

```
@DiscriminatorColumn(name="DISCRIM", discriminatorType=DiscriminatorType.INTEGER)
```

so we introduce our own convenience annotation

```
@Target(TYPE)
@Retention(RUNTIME)
@DiscriminatorColumn(name="DISCRIM", discriminatorType=DiscriminatorType.INTEGER)
public @interface MyDiscriminator
{
}
```

so now we can simply annotate a JPA entity that needs this discriminator with

```
@Entity
@MyDiscriminator
public class MyClass
{
    ...
}
```

You can also make use of *meta-annotations* on fields/properties.