



# JPA Annotations Reference (v5.0)

# Table of Contents

JPA Class-Level Annotations .....	2
@Entity .....	3
@MappedSuperclass .....	4
@Embeddable .....	4
@IdClass .....	4
@Cacheable .....	5
@EntityListeners .....	5
@NamedQueries .....	5
@NamedQuery .....	6
@NamedNativeQueries .....	6
@NamedNativeQuery .....	7
@NamedStoredProcedureQueries .....	7
@NamedStoredProcedureQuery .....	8
@SqlResultSetMappings .....	9
@SqlResultSetMapping .....	9
@NamedEntityGraphs .....	10
@NamedEntityGraph .....	10
@Converter .....	11
@Inheritance .....	11
@Table .....	12
@SecondaryTable .....	12
@DiscriminatorColumn .....	13
@DiscriminatorValue .....	13
@PrimaryKeyJoinColumns .....	14
@PrimaryKeyJoinColumn .....	14
@AttributeOverride .....	15
@AttributeOverrides .....	15
@AssociationOverride .....	16
@AssociationOverrides .....	16
@SequenceGenerator .....	17
@TableGenerator .....	17
JPA Field-Level Annotations .....	19
@PrePersist .....	21
@PostPersist .....	21
@PreRemove .....	21
@PostRemove .....	22
@PreUpdate .....	22
@PostUpdate .....	23

@PostLoad .....	23
@Id .....	24
@Embedded .....	24
@EmbeddedId .....	24
@Version .....	25
@Basic .....	25
@Transient .....	26
@JoinTable .....	26
@CollectionTable .....	27
@Lob .....	27
@Temporal .....	28
@Enumerated .....	28
@OneToOne .....	29
@OneToMany .....	30
@ManyToMany .....	30
@ManyToOne .....	31
@ElementCollection .....	32
@GeneratedValue .....	32
@MapKey .....	33
@MapKeyClass .....	33
@MapKeyTemporal .....	34
@MapKeyEnumerated .....	34
@MapKeyColumn .....	35
@OrderBy .....	35
@OrderColumn .....	35
@Convert .....	36
@Column .....	36
@JoinColumn .....	37
@JoinColumns .....	38
@UniqueConstraint .....	38
@Index .....	39
@ForeignKey .....	39
DataNucleus Class-Level Extensions .....	40
@PersistenceAware .....	40
@DatastoreId .....	40
@NonDurableId .....	41
@ReadOnly .....	41
@MultiTenant .....	41
DataNucleus Field-Level Extensions .....	43
@SharedRelation .....	43
@ValueGenerator .....	44

@ReadOnly .....	44
@CreateTimestamp .....	45
@UpdateTimestamp .....	45
@Index (field/method - extension) .....	45
@JdbcType .....	46
@ColumnPosition .....	46
@Extensions .....	47
@Extension .....	47
Meta-Annotations.....	48

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
<a href="#">@Entity</a>	Class	Specifies that the class is persistent
<a href="#">@MappedSuperclass</a>	Class	Specifies that this class contains persistent information to be mapped
<a href="#">@Embeddable</a>	Class	Specifies that the class is persistent embedded in another persistent class
<a href="#">@IdClass</a>	Class	Defines the primary key class for this class
<a href="#">@Cacheable</a>	Class	Specifies that instances of this class can be cached in the L2 cache
<a href="#">@EntityListeners</a>	Class	Specifies class(es) that are listeners for events from instances of this class
<a href="#">@NamedQueries</a>	Class	Defines a series of named JPQL queries for use in the current persistence unit
<a href="#">@NamedQuery</a>	Class	Defines a named JPQL query for use in the current persistence unit
<a href="#">@NamedNativeQuery</a>	Class	Defines a named SQL query for use in the current persistence unit
<a href="#">@NamedNativeQueries</a>	Class	Defines a series of named SQL queries for use in the current persistence unit
<a href="#">@NamedStoredProcedureQuery</a>	Class	Defines a named stored procedure query for use in the current persistence unit
<a href="#">@NamedStoredProcedureQueries</a>	Class	Defines a series of named stored procedure queries for use in the current persistence unit
<a href="#">@SqlResultSetMapping</a>	Class	Defines a result mapping for an SQL query for use in the current persistence unit
<a href="#">@SqlResultSetMappings</a>	Class	Defines a series of mappings for SQL queries for use in the current persistence unit
<a href="#">@NamedEntityGraph</a>	Class	Defines a named entity graph with root of the class it is specified on
<a href="#">@NamedEntityGraphs</a>	Class	Defines named entity graphs with root of the class it is specified on
<a href="#">@Converter</a>	Class	Defines a java type converter for a field type
<a href="#">@Inheritance</a>	Class	Specifies the inheritance model for persisting this class
<a href="#">@Table</a>	Class	Defines the table where this class will be stored

Annotation	Class/Field	Description
<a href="#">@SecondaryTable</a>	Class	Defines a secondary table where some fields of this class will be stored
<a href="#">@DiscriminatorColumn</a>	Class	Defines the column where any discriminator will be stored
<a href="#">@DiscriminatorValue</a>	Class	Defines the value to be used in the discriminator for objects of this class
<a href="#">@PrimaryKeyJoinColumn</a>	Class	Defines the names of the PK columns when this class has a superclass
<a href="#">@PrimaryKeyJoinColumn</a>	Class	Defines the name of the PK column when this class has a superclass
<a href="#">@AttributeOverride</a>	Class	Defines a field in a superclass that will have its column overridden
<a href="#">@AttributeOverrides</a>	Class	Defines the field(s) of superclasses that will have their columns overridden
<a href="#">@AssociationOverride</a>	Class	Defines a N-1/1-1 field in a superclass that will have its column overridden
<a href="#">@AssociationOverrides</a>	Class	Defines the N-1/1-1 field(s) of superclasses that will have their columns overridden
<a href="#">@SequenceGenerator</a>	Class/Field/Method	Defines a generator of values using sequences in the datastore for use with persistent entities
<a href="#">@TableGenerator</a>	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(org.datanucleus.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(org.datanucleus.MyListener.class)
public class MyClass
{
    ...
}
```

See the documentation for [Lifecycle Callbacks](#)

## @NamedQueries

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

Attribute	Type	Description	Default
value	NamedQuery[]	The named queries	

```

@Entity
@NamedQueries({
    @NamedQuery(name="AllPeople",
        query="SELECT p FROM Person p"),
    @NamedQuery(name="PeopleCalledJones",
        query="SELECT p FROM Person p WHERE p.surname = 'Jones'")})
public class Person
{
    ...
}

```

Note that with DataNucleus you can also specify `@NamedQueries` on non-persistable classes See the documentation for [Named Queries](#)

## @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](#)

## @NamedNativeQueries

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

Attribute	Type	Description	Default
value	NamedNativeQuery[]	The named native queries	

```

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

```

Note that with DataNucleus you can also specify `@NamedNativeQueries` on non-persistable classes See the documentation for [Named Native Queries](#)

## @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](#)

## @NamedStoredProcedureQueries

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

Attribute	Type	Description	Default
value	NamedStoredProcedureQuery []	The named stored procedure queries	

```

@Entity
@NamedStoredProcedureQueries({
    @NamedStoredProcedureQuery(name="MyProc", procedureName="MY_PROC_SP1",
        parameters={@StoredProcedureParameter(name="PARAM1", mode=ParameterMode.IN,
            type=String.class)}),
    @NamedStoredProcedureQuery(name="MyProc2", procedureName="MY_PROC_SP2",
        parameters={@StoredProcedureParameter(name="PARAM1", mode=ParameterMode.IN,
            type=Long.class)}))
public class Person
{
    ...
}

```

Note that with DataNucleus you can also specify `@NamedStoredProcedureQueries` on non-persistable classes See the documentation for [Named Stored procedures](#)

## @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](#)

## @SqlResultSetMappings

This annotation is used to define a series of result mappings for SQL queries that can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
value	SqlResultSetMapping[]	The SQL result mappings	

```

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

```

## @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	

Attribute	Type	Description	Default
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
{
    ...
}

```

## @NamedEntityGraphs

This annotation is used to define a series of named EntityGraphs that can be used in this persistence unit. Specified on the **class**.

Attribute	Type	Description	Default
value	NamedEntityGraph[]	The named EntityGraphs	

```

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

```

## @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.	

Attribute	Type	Description	Default
attributeNodes	AttributeNode[ ]	Set of nodes in this EntityGraph	

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

## @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	<b>SINGLE_TABLE</b> , <b>JOINED</b> , <b>TABLE_PER_CLASS</b>

```

@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	PrimaryKeyJoinColumn[]	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](#)

## @PrimaryKeyJoinColumns

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

Attribute	Type	Description	Default
value	PrimaryKeyJoinColumn[]	Array of column definitions for the primary key	
foreignKey	ForeignKey	Foreign key details if wanting to override provider default	

```

@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
@PrimaryKeyJoinColumns({@PrimaryKeyJoinColumn(name="PK_FIELD_1", referredColumnName="BASE_1_ID"),
                        @PrimaryKeyJoinColumn(name="PK_FIELD_2", referredColumnName="BASE_2_ID")})
public class MyClass
{
    ...
}

```

## @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	

Attribute	Type	Description	Default
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. <b>It is not to allow joining to some non-PK column</b>	
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
{
    ...
}
```

## @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
{
    ...
}
```

## @AttributeOverrides

This annotation is used to define fields of a superclass that have their columns overridden. Specified on the **class**.

Attribute	Type	Description	Default
value	AttributeOverride[]	The overrides	

```

@Entity
@AttributeOverrides({@AttributeOverride(name="attr1", column=@Column(name="NEW_NAME_1")),
                    @AttributeOverride(name="attr2", column=@Column(name="NEW_NAME_2"))})
public class MyClass extends MySuperClass
{
    ...
}

```

## @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
{
    ...
}

```

## @AssociationOverrides

This annotation is used to define 1-1/N-1 fields of a superclass that have their columns overridden. Specified on the **class**.

Attribute	Type	Description	Default
value	AssociationOverride[]	The overrides	

```

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

```

## @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)	

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

<b>Annotation</b>	<b>Class/Field</b>	<b>Description</b>
<a href="#">@OrderBy</a>	Field/Method	Defines the field(s) used for ordering the elements in this collection
<a href="#">@OrderColumn</a>	Field/Method	Defines that ordering should be attributed by the implementation using a surrogate column.
<a href="#">@PrePersist</a>	Field/Method	Defines this method as being a callback for pre-persist events
<a href="#">@PostPersist</a>	Field/Method	Defines this method as being a callback for post-persist events
<a href="#">@PreRemove</a>	Field/Method	Defines this method as being a callback for pre-remove events
<a href="#">@PostRemove</a>	Field/Method	Defines this method as being a callback for post-remove events
<a href="#">@PreUpdate</a>	Field/Method	Defines this method as being a callback for pre-update events
<a href="#">@PostUpdate</a>	Field/Method	Defines this method as being a callback for post-update events
<a href="#">@PostLoad</a>	Field/Method	Defines this method as being a callback for post-load events
<a href="#">@JoinTable</a>	Field/Method	Defines this field as being stored using a join table
<a href="#">@CollectionTable</a>	Field/Method	Defines this field as being stored using a join table when containing non-entity objects.
<a href="#">@Lob</a>	Field/Method	Defines this field as being stored as a large object
<a href="#">@Temporal</a>	Field/Method	Defines this field as storing temporal data
<a href="#">@Enumerated</a>	Field/Method	Defines this field as storing enumerated data
<a href="#">@Convert</a>	Field/Method	Defines a converter for this field/property
<a href="#">@Column</a>	Field/Method	Defines the column where this field is stored
<a href="#">@JoinColumn</a>	Field/Method	Defines a column for joining to either a join table or foreign key relation
<a href="#">@JoinColumns</a>	Field/Method	Defines the columns for joining to either a join table or foreign key relation (1-1, 1-N, N-1)
<a href="#">@Index</a>	-	Defines the details of an index when overriding the provider default.
<a href="#">@ForeignKey</a>	-	Defines the details of a foreign key when overriding the provider default.



## @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	<b>EAGER, LAZY</b>
optional	boolean	Whether the field can store nulls.	<b>true, false</b>
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.	
<b>cascade</b>	CascadeType[]	Whether persist, update, delete, refresh operations are cascaded	
<b>orphanRemoval</b>	boolean	Whether to remove orphans when either removing this side of the relation or when nulling the relation	<b>true, false</b>

```

@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.	
<code>cascade</code>	CascadeType[]	Whether persist, update, delete, refresh operations are cascaded	
<code>orphanRemoval</code>	boolean	Whether to remove orphans when either removing this side of the relation or when nulling the relation 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.	
<a href="#">cascade</a>	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	<b>EAGER, LAZY</b>
optional	boolean	Whether the field can store nulls.	<b>true, false</b>
<a href="#">cascade</a>	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.	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.	<i>{fieldName}_ORDER</i>
nullable	boolean	Whether the column is nullable	<b>true</b> , false
insertable	boolean	Whether the column is insertable	<b>true</b> , false
updatable	boolean	Whether the column is updatable	<b>true</b> , 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	Name of the embedded field to be converted (NOT YET 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. <b>It is not to allow joining to some non-PK column</b>	
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

Attribute	Type	Description	Default
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;
    ...
}
```

## @JoinColumn

This annotation is used to define the FK columns 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
value	JoinColumn[]	Details of the columns	
foreignKey	ForeignKey	Foreign key details if wanting to override provider default	

```
@Entity
public class Person
{
    @OneToOne
    @JoinColumns({@JoinColumn(name="PET1_ID"), @JoinColumn(name="PET2_ID")})
    Animal pet; // composite PK
    ...
}
```

## @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 [Schema 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 <i>colName1, colName2</i>	
unique	boolean	Whether the index is unique	false

See the documentation for [Schema 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 <i>FOREIGN KEY ( colExpr1 {, colExpr2}... ) REFERENCES tblIdentifier {( otherColExpr1 {, otherColExpr2}... ) } { ON UPDATE updateAction } { ON DELETE deleteAction }</i>	

See the documentation for [Schema Constraints](#)

# 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
<a href="#">@PersistenceAware</a>	Class	Specifies that the class is not persistent but needs to be able to access fields of persistent classes (DataNucleus extension).
<a href="#">@DatastoreId</a>	Class	Defines a class as using datastore-identity (DataNucleus extension).
<a href="#">@NonDurableId</a>	Class	Defines a class as using nondurable identity (DataNucleus extension).
<a href="#">@ReadOnly</a>	Class	Specifies that this class is "read-only" (DataNucleus extension).
<a href="#">@MultiTenant</a>	Class	Specifies multi-tenancy details for this class (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.	<b>AUTO</b> , TABLE, SEQUENCE
generator	String	Name of the generator to use. See <a href="#">@TableGenerator</a> and <a href="#">@SequenceGenerator</a>	

Attribute	Type	Description	Default
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 nondurable 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

Attribute	Type	Description	Default
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
{
    ...
}
```

# 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
<a href="#">@SharedRelation</a>	Field/Method	Specifies that the relation for this field/property is "shared" (DataNucleus extension).
<a href="#">@ReadOnly</a>	Field/Method	Specifies that this field/property is "read-only" (DataNucleus extension).
<a href="#">@Index</a>	Field/Method	Specifies an index on this field/property (DataNucleus extension).
<a href="#">@JdbcType</a>	Field/Method	Specifies the JDBC Type to use on this field/property (DataNucleus extension).
<a href="#">@ColumnPosition</a>	Field/Method	Specifies the column position to use on this field/property (DataNucleus extension).
<a href="#">@ValueGenerator</a>	Field/Method	Specifies a non-JPA-standard value generator to use on this field/property (DataNucleus extension).
<a href="#">@Extensions</a>	Class/Field/Method	Defines a series of DataNucleus extensions (DataNucleus extension).
<a href="#">@Extension</a>	Class/Field/Method	Defines a DataNucleus extension (DataNucleus extension).
<a href="#">@CreateTimestamp</a>	Field/Method	Specifies that this field/property should store a creation timestamp when inserting (DataNucleus extension).
<a href="#">@UpdateTimestamp</a>	Field/Method	Specifies that this field/property should store an update timestamp 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	
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 as 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;

    ...
}

```

## @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;

    ...
}

```

## @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	

Attribute	Type	Description	Default
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;
    ...
}
```

## @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;
    ...
}
```

## @Extensions

*DataNucleus Extension Annotation* used to define a set of extensions specific to DataNucleus. Specified on the **class** or **field**.

Attribute	Type	Description	Default
value	Extension[]	Array of extensions - see @Extension annotation	

```
@Entity
@Extensions({@Extension(key="firstExtension", value="myValue"),
             @Extension(key="secondExtension", value="myValue")})
public class Person
{
    ...
}
```

## @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
{
    ...
}
```

# 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.