Project Versions

latest ▾

Search

Go

Table Of Contents

22. Annotations Reference

You've probably used docblock annotations in some form already, most likely to provide documentation metadata for a tool like **`PHPDocumentor`** (@author, @link, ...). Docblock annotations are a tool to embed metadata inside the documentation section which can then be processed by some tool. Doctrine 2 generalizes the concept of docblock annotations so that they can be used for any kind of metadata and so that it is easy to define new docblocl annotations. In order to allow more involved annotation values and to reduce the chances o clashes with other docblock annotations, the Doctrine 2 docblock annotations feature an alternative syntax that is heavily inspired by the Annotation syntax introduced in Java 5.

The implementation of these enhanced docblock annotations is located in the**`Doctrine\Common\Annotations`** namespace and therefore part of the Common package. Doctrine 2 docblock annotations support namespaces and nested annotations among other things. The Doctrine 2 ORM defines its own set of docblock annotations for supplying object relational mapping metadata.

> If you're not comfortable with the concept of docblock annotations, don't worry, as mentioned earlier Doctrine 2 provides XML and YAML alternatives and you could easily implement your own favourite mechanism for defining ORM metadata.

In this chapter a reference of every Doctrine 2 Annotation is given with short explanations on their context and usage.

22.1. Index

Previous topic

21. YAML Mapping

Next topic

23. PHP Mapping

This Page

Show Source

22.2. Reference

22.2.1. @Column

Marks an annotated instance variable as “persistent”. It has to be inside the instance variables PHP DocBlock comment. Any value hold inside this variable will be saved to and loaded from the database as part of the lifecycle of the instance variables entity-class.

Required attributes:

- **type**: Name of the Doctrine Type which is converted between PHP and Database representation.

Optional attributes:

- **name**: By default the property name is used for the database column name also, however the ‘name’ attribute allows you to determine the column name.
- **length**: Used by the “string” type to determine its maximum length in the database. Doctrine does not validate the length of a string values for you.
- **precision**: The precision for a decimal (exact numeric) column (applies only for decimal column), which is the maximum number of digits that are stored for the values.
- **scale**: The scale for a decimal (exact numeric) column (applies only for decimal column), which represents the number of digits to the right of the decimal point and must not be greater than **precision**.
- **unique**: Boolean value to determine if the value of the column should be unique

- **nullable**: Determines if NULL values allowed for this column.
- **options**: Array of additional options:
  - **`default`**: The default value to set for the column if no value is supplied.
  - **`unsigned`**: Boolean value to determine if the column should be capable of representing only non-negative integers (applies only for integer column and r not be supported by all vendors).
  - **`fixed`**: Boolean value to determine if the specified length of a string column sh be fixed or varying (applies only for string/binary column and might not be supported by all vendors).
  - **`comment`**: The comment of the column in the schema (might not be supported vendors).
  - **`customSchemaOptions`**: Array of additional schema options which are mostly ve specific.
- **columnDefinition**: DDL SQL snippet that starts after the column name and specifies the complete (non-portable!) column definition. This attribute allows to make use of advanced RMDBS features. However you should make careful use of this feature and the consequences. SchemaTool will not detect changes on the column correctly anymore if you use "columnDefinition".

  Additionally you should remember that the "type" attribute still handles the conversion between PHP and Database values. If you use this attribute on a column that is used for joins between tables you should also take a look at@JoinColumn.

For more detailed information on each attribute, please refer to the DBAL **`Schema`** **`Representation`** documentation.

Examples:

```
<?php
/**
 * @Column(type="string", length=32, unique=true, nullable=false)
 */
```

```
/**
 * @Column(type="string", columnDefinition="CHAR(2) NOT NULL")
 */
protected $country;

/**
 * @Column(type="decimal", precision=2, scale=1)
 */
protected $height;

/**
 * @Column(type="string", length=2, options={"fixed":true, "comment":"Initia
etters of first and last name"})
 */
protected $initials;

/**
 * @Column(type="integer", name="login_count" nullable=false, options={"unsi
d":true, "default":0})
 */
protected $loginCount;
```

22.2.2. @ColumnResult

References name of a column in the SELECT clause of a SQL query. Scalar result types can be included in the query result by specifying this annotation in the metadata.

Required attributes:

- **name**: The name of a column in the SELECT clause of a SQL query

22.2.3. @Cache

Add caching strategy to a root entity or a collection.

Optional attributes:

- **usage**: One of **`READ_ONLY`**, **`READ_WRITE`** or **`NONSTRICT_READ_WRITE`**, By default this

- **region**: An specific region name

22.2.4. @ChangeTrackingPolicy

The Change Tracking Policy annotation allows to specify how the Doctrine 2 UnitOfWork should detect changes in properties of entities during flush. By default each entity is checked according to a deferred implicit strategy, which means upon flush UnitOfWork compares all the properties of an entity to a previously stored snapshot. This works out of the box, however you might want to tweak the flush performance where using another change tracking policy is an interesting option.

The **details on all the available change tracking policies** can be found in the configuration section.

Example:

```
<?php
/**
 * @Entity
 * @ChangeTrackingPolicy("DEFERRED_IMPLICIT")
 * @ChangeTrackingPolicy("DEFERRED_EXPLICIT")
 * @ChangeTrackingPolicy("NOTIFY")
 */
class User {}
```

22.2.5. @DiscriminatorColumn

This annotation is a required annotation for the topmost/super class of an inheritance hierarchy. It specifies the details of the column which saves the name of the class, which the entity is actually instantiated as.

Required attributes:

- **name**: The column name of the discriminator. This name is also used during Array hydration as key to specify the class-name.

Optional attributes:

- **length**: By default this is 255.

22.2.6. @DiscriminatorMap

The discriminator map is a required annotation on the topmost/super class in an inheritance hierarchy. Its only argument is an array which defines which class should be saved under which name in the database. Keys are the database value and values are the classes, either as fully- or as unqualified class names depending on whether the classes are in the namespace or not.

```
<?php
/**
 * @Entity
 * @InheritanceType("JOINED")
 * @DiscriminatorColumn(name="discr", type="string")
 * @DiscriminatorMap({"person" = "Person", "employee" = "Employee"})
 */
class Person
{
    // ...
}
```

22.2.7. @Entity

Required annotation to mark a PHP class as an entity. Doctrine manages the persistence of all classes marked as entities.

Optional attributes:

- **repositoryClass**: Specifies the FQCN of a subclass of the EntityRepository. Use of repositories for entities is encouraged to keep specialized DQL and SQL operations sepa from the Model/Domain Layer.
- **readOnly**: (>= 2.1) Specifies that this entity is marked as read only and not considere change-tracking. Entities of this type can be persisted and removed though.

Example:

```php
<?php
/**
 * @Entity(repositoryClass="MyProject\UserRepository")
 */
class User
{
    //...
}
```

22.2.8. @EntityResult

References an entity in the SELECT clause of a SQL query. If this annotation is used, the SQL statement should select all of the columns that are mapped to the entity object. This should include foreign key columns to related entities. The results obtained when insufficien data is available are undefined.

Required attributes:

- **entityClass**: The class of the result.

Optional attributes:

- **fields**: Array of @FieldResult, Maps the columns specified in the SELECT list of the que the properties or fields of the entity class.
- **discriminatorColumn**: Specifies the column name of the column in the SELECT list th used to determine the type of the entity instance.

22.2.9. @FieldResult

Is used to map the columns specified in the SELECT list of the query to the properties or fields of the entity class.

Required attributes:

- **name**: Name of the persistent field or property of the class.

Optional attributes:

- **column**: Name of the column in the SELECT clause.

22.2.10. @GeneratedValue

Specifies which strategy is used for identifier generation for an instance variable which is annotated by **@Id**. This annotation is optional and only has meaning when used in conjunction with @Id.

If this annotation is not specified with @Id the NONE strategy is used as default.

Required attributes:

- **strategy**: Set the name of the identifier generation strategy. Valid values are AUTO, SEQUENCE, TABLE, IDENTITY, UUID, CUSTOM and NONE.

Example:

```
<?php
/**
 * @Id
 * @Column(type="integer")
 * @GeneratedValue(strategy="IDENTITY")
 */
protected $id = null;
```

22.2.11. @HasLifecycleCallbacks

Annotation which has to be set on the entity-class PHP DocBlock to notify Doctrine that this entity has entity lifecycle callback annotations set on at least one of its methods. Using @PostLoad, @PrePersist, @PostPersist, @PreRemove, @PostRemove, @PreUpdate or @PostUpdate without this marker annotation will make Doctrine ignore the callbacks.

Example:

```
<?php
/**
```

```
 */
class User
{
    /**
     * @PostPersist
     */
    public function sendOptinMail() {}
}
```

22.2.12. @Index

Annotation is used inside the **@Table** annotation on the entity-class level. It provides a hint to the SchemaTool to generate a database index on the specified table columns. It only has meaning in the SchemaTool schema generation context.

Required attributes:

- **name**: Name of the Index
- **columns**: Array of columns.

Optional attributes:

- **options**: Array of platform specific options:
    - `where`: SQL WHERE condition to be used for partial indexes. It will only have e on supported platforms.

Example:

```
<?php
/**
 * @Entity
 * @Table(name="ecommerce_products",indexes={@Index(name="search_idx", colum
{"name", "email"})})
 */
class ECommerceProduct
{
}
```

22.2.13. @Id

The annotated instance variable will be marked as entity identifier, the primary key in the database. This annotation is a marker only and has no required or optional attributes. For entities that have multiple identifier columns each column has to be marked with @Id.

Example:

```
<?php
/**
 * @Id
 * @Column(type="integer")
 */
protected $id = null;
```

22.2.14. @InheritanceType

In an inheritance hierarchy you have to use this annotation on the topmost/super class to define which strategy should be used for inheritance. Currently Single Table and Class Table Inheritance are supported.

This annotation has always been used in conjunction with the **@DiscriminatorMap** and**@DiscriminatorColumn** annotations.

Examples:

```
<?php
/**
 * @Entity
 * @InheritanceType("SINGLE_TABLE")
 * @DiscriminatorColumn(name="discr", type="string")
 * @DiscriminatorMap({"person" = "Person", "employee" = "Employee"})
 */
class Person
{
    // ...
```

```
/**
 * @Entity
 * @InheritanceType("JOINED")
 * @DiscriminatorColumn(name="discr", type="string")
 * @DiscriminatorMap({"person" = "Person", "employee" = "Employee"})
 */
class Person
{
    // ...
}
```

22.2.15. @JoinColumn

This annotation is used in the context of relations in @ManyToOne, @OneToOne fields an in the Context of @JoinTable nested inside a @ManyToMany. This annotation is not required. If it is not specified the attributes **name** and **referencedColumnName** are inferred from the table and primary key names.

Required attributes:

- **name**: Column name that holds the foreign key identifier for this relation. In the conte @JoinTable it specifies the column name in the join table.
- **referencedColumnName**: Name of the primary key identifier that is used for joining this relation.

Optional attributes:

- **unique**: Determines whether this relation is exclusive between the affected entities an should be enforced as such on the database constraint level. Defaults to false.
- **nullable**: Determine whether the related entity is required, or if null is an allowed stat the relation. Defaults to true.
- **onDelete**: Cascade Action (Database-level)
- **columnDefinition**: DDL SQL snippet that starts after the column name and specifies t complete (non-portable!) column definition. This attribute enables the use of advanced RMDBS features. Using this attribute on @JoinColumn is necessary if you need slightly

defaults. However by default a "columnDefinition" attribute on @Column also sets the related @JoinColumn's columnDefinition. This is necessary to make foreign keys work.

Example:

```
<?php
/**
 * @OneToOne(targetEntity="Customer")
 * @JoinColumn(name="customer_id", referencedColumnName="id")
 */
private $customer;
```

22.2.16. @JoinColumns

An array of @JoinColumn annotations for a @ManyToOne or @OneToOne relation with an entity that has multiple identifiers.

22.2.17. @JoinTable

Using @OneToMany or @ManyToMany on the owning side of the relation requires to specify the @JoinTable annotation which describes the details of the database join table. If you do not specify @JoinTable on these relations reasonable mapping defaults apply using the affected table and the column names.

Optional attributes:

- **name**: Database name of the join-table
- **joinColumns**: An array of @JoinColumn annotations describing the join-relation betwe the owning entities table and the join table.
- **inverseJoinColumns**: An array of @JoinColumn annotations describing the join-relatic between the inverse entities table and the join table.

Example:

```
 * @ManyToMany(targetEntity="Phonenumber")
 * @JoinTable(name="users_phonenumbers",
 *      joinColumns={@JoinColumn(name="user_id", referencedColumnName="id")}
 *      inverseJoinColumns={@JoinColumn(name="phonenumber_id", referencedCol
Name="id", unique=true)}
 * )
 */
public $phonenumbers;
```

22.2.18. @ManyToOne

Defines that the annotated instance variable holds a reference that describes a many-to-one
relationship between two entities.

Required attributes:

- **targetEntity**: FQCN of the referenced target entity. Can be the unqualified class name both classes are in the same namespace. **IMPORTANT:** No leading backslash!

Optional attributes:

- **cascade**: Cascade Option
- **fetch**: One of LAZY or EAGER
- inversedBy - The inversedBy attribute designates the field in the entity that is the inver side of the relationship.

Example:

```
<?php
/**
 * @ManyToOne(targetEntity="Cart", cascade={"all"}, fetch="EAGER")
 */
private $cart;
```

22.2.19. @ManyToMany

two entities. **@JoinTable** is an additional, optional annotation that has reasonable default configuration values using the table and names of the two related entities.

Required attributes:

- **targetEntity**: FQCN of the referenced target entity. Can be the unqualified class name both classes are in the same namespace. **IMPORTANT:** No leading backslash!

Optional attributes:

- **mappedBy**: This option specifies the property name on the targetEntity that is the owı side of this relation. It is a required attribute for the inverse side of a relationship.
- **inversedBy**: The inversedBy attribute designates the field in the entity that is the inve side of the relationship.
- **cascade**: Cascade Option
- **fetch**: One of LAZY, EXTRA_LAZY or EAGER
- **indexBy**: Index the collection by a field on the target entity.

For ManyToMany bidirectional relationships either side may be the owning side (the side that defines the @JoinTable and/or does not make use of the mappedBy attribute, thus using a default join table).

Example:

```
<?php
/**
 * Owning Side
 *
 * @ManyToMany(targetEntity="Group", inversedBy="features")
 * @JoinTable(name="user_groups",
 *      joinColumns={@JoinColumn(name="user_id", referencedColumnName="id")}
 *      inverseJoinColumns={@JoinColumn(name="group_id", referencedColumnNam
id")}
 *      )
 */
private $groups;

/**
```

```
 *
 * @ManyToMany(targetEntity="User", mappedBy="groups")
 */
private $features;
```

22.2.20. @MappedSuperclass

A mapped superclass is an abstract or concrete class that provides persistent entity state and mapping information for its subclasses, but which is not itself an entity. This annotation is specified on the Class docblock and has no additional attributes.

The @MappedSuperclass annotation cannot be used in conjunction with @Entity. See the Inheritance Mapping section for **more details on the restrictions of mapped superclasses**.

Optional attributes:

- **repositoryClass**: (>= 2.2) Specifies the FQCN of a subclass of the EntityRepository. T will be inherited for all subclasses of that Mapped Superclass.

Example:

```
<?php
/**
 * @MappedSuperclass
 */
class MappedSuperclassBase
{
    // ... fields and methods
}

/**
 * @Entity
 */
class EntitySubClassFoo extends MappedSuperclassBase
{
    // ... fields and methods
}
```

22.2.21. @NamedNativeQuery

Is used to specify a native SQL named query. The NamedNativeQuery annotation can be applied to an entity or mapped superclass.

Required attributes:

- **name**: The name used to refer to the query with the EntityManager methods that crea query objects.
- **query**: The SQL query string.

Optional attributes:

- **resultClass**: The class of the result.
- **resultSetMapping**: The name of a SqlResultSetMapping, as defined in metadata.

Example:

```
<?php
/**
 * @NamedNativeQueries({
 *      @NamedNativeQuery(
 *          name            = "fetchJoinedAddress",
 *          resultSetMapping= "mappingJoinedAddress",
 *          query           = "SELECT u.id, u.name, u.status, a.id AS a_id,
ountry, a.zip, a.city FROM cms_users u INNER JOIN cms_addresses a ON u.id =
ser_id WHERE u.username = ?"
 *      ),
 * })
 * @SqlResultSetMappings({
 *      @SqlResultSetMapping(
 *          name    = "mappingJoinedAddress",
 *          entities= {
 *              @EntityResult(
 *                  entityClass = "__CLASS__",
 *                  fields      = {
 *                      @FieldResult(name = "id"),
 *                      @FieldResult(name = "name"),
 *                      @FieldResult(name = "address.zip"),
 *                      @FieldResult(name = "address.city"),
 *                      @FieldResult(name = "address.country"),
 *                      @FieldResult(name = "address.id", column = "a_id"),
 *                  }
 *              )
 *          }
 *      )
 * })
 */
class User
{
    /** @Id @Column(type="integer") @GeneratedValue */
    public $id;

    /** @Column(type="string", length=50, nullable=true) */
    public $status;

    /** @Column(type="string", length=255, unique=true) */
    public $username;

    /** @Column(type="string", length=255) */
    public $name;

    /** @OneToOne(targetEntity="Address") */
    public $address;

    // ....
}
```

22.2.22. @OneToOne

The @OneToOne annotation works almost exactly as the **@ManyToOne** with one additional option which can be specified. The configuration defaults for **@JoinColumn**using the target entity table and primary key column names apply here too.

Required attributes:

- **targetEntity**: FQCN of the referenced target entity. Can be the unqualified class name both classes are in the same namespace. **IMPORTANT:** No leading backslash!

- **cascade**: Cascade Option
- **fetch**: One of LAZY or EAGER
- **orphanRemoval**: Boolean that specifies if orphans, inverse OneToOne entities that ar connected to any owning instance, should be removed by Doctrine. Defaults to false.
- **inversedBy**: The inversedBy attribute designates the field in the entity that is the inve side of the relationship.

Example:

```
<?php
/**
 * @OneToOne(targetEntity="Customer")
 * @JoinColumn(name="customer_id", referencedColumnName="id")
 */
private $customer;
```

22.2.23. @OneToMany

Required attributes:

- **targetEntity**: FQCN of the referenced target entity. Can be the unqualified class name both classes are in the same namespace. **IMPORTANT:** No leading backslash!

Optional attributes:

- **cascade**: Cascade Option
- **orphanRemoval**: Boolean that specifies if orphans, inverse OneToOne entities that ar connected to any owning instance, should be removed by Doctrine. Defaults to false.
- **mappedBy**: This option specifies the property name on the targetEntity that is the ow side of this relation. Its a required attribute for the inverse side of a relationship.
- **fetch**: One of LAZY, EXTRA_LAZY or EAGER.
- **indexBy**: Index the collection by a field on the target entity.

```php
<?php
/**
 * @OneToMany(targetEntity="Phonenumber", mappedBy="user", cascade={"persist
"remove", "merge"}, orphanRemoval=true)
 */
public $phonenumbers;
```

22.2.24. @OrderBy

Optional annotation that can be specified with a **@ManyToMany** or **@OneToMany**annotation to specify by which criteria the collection should be retrieved from the database by using an ORDER BY clause.

This annotation requires a single non-attributed value with an DQL snippet:

Example:

```php
<?php
/**
 * @ManyToMany(targetEntity="Group")
 * @OrderBy({"name" = "ASC"})
 */
private $groups;
```

The DQL Snippet in OrderBy is only allowed to consist of unqualified, unquoted field names and of an optional ASC/DESC positional statement. Multiple Fields are separated by a comma (,). The referenced field names have to exist on the `targetEntity` class of the`@ManyToMany` or `@OneToMany` annotation.

22.2.25. @PostLoad

Marks a method on the entity to be called as a @PostLoad event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.26. @PostPersist

Marks a method on the entity to be called as a @PostPersist event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.27. @PostRemove

Marks a method on the entity to be called as a @PostRemove event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.28. @PostUpdate

Marks a method on the entity to be called as a @PostUpdate event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.29. @PrePersist

Marks a method on the entity to be called as a @PrePersist event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.30. @PreRemove

Marks a method on the entity to be called as a @PreRemove event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.31. @PreUpdate

Marks a method on the entity to be called as a @PreUpdate event. Only works with @HasLifecycleCallbacks in the entity class PHP DocBlock.

22.2.32. @SequenceGenerator

For use with @GeneratedValue(strategy="SEQUENCE") this annotation allows to specify

Required attributes:

- **sequenceName**: Name of the sequence

Optional attributes:

- **allocationSize**: Increment the sequence by the allocation size when its fetched. A valu larger than 1 allows optimization for scenarios where you create more than one new en per request. Defaults to 10
- **initialValue**: Where the sequence starts, defaults to 1.

Example:

```
<?php
/**
 * @Id
 * @GeneratedValue(strategy="SEQUENCE")
 * @Column(type="integer")
 * @SequenceGenerator(sequenceName="tablename_seq", initialValue=1, allocati
ize=100)
 */
protected $id = null;
```

22.2.33. @SqlResultSetMapping

The SqlResultSetMapping annotation is used to specify the mapping of the result of a native SQL query. The SqlResultSetMapping annotation can be applied to an entity or mapped superclass.

Required attributes:

- **name**: The name given to the result set mapping, and used to refer to it in the methoc the Query API.

Optional attributes:

- **entities**: Array of @EntityResult, Specifies the result set mapping to entities.
- **columns**: Array of @ColumnResult, Specifies the result set mapping to scalar values

```
<?php
/**
 * @NamedNativeQueries({
 *      @NamedNativeQuery(
 *          name            = "fetchUserPhonenumberCount",
 *          resultSetMapping= "mappingUserPhonenumberCount",
 *          query           = "SELECT id, name, status, COUNT(phonenumber) A
umphones FROM cms_users INNER JOIN cms_phonenumbers ON id = user_id WHERE us
ame IN (?) GROUP BY id, name, status, username ORDER BY username"
 *      ),
 *      @NamedNativeQuery(
 *          name            = "fetchMultipleJoinsEntityResults",
 *          resultSetMapping= "mappingMultipleJoinsEntityResults",
 *          query           = "SELECT u.id AS u_id, u.name AS u_name, u.stat
AS u_status, a.id AS a_id, a.zip AS a_zip, a.country AS a_country, COUNT(p.p
enumber) AS numphones FROM cms_users u INNER JOIN cms_addresses a ON u.id = 
ser_id INNER JOIN cms_phonenumbers p ON u.id = p.user_id GROUP BY u.id, u.na
 u.status, u.username, a.id, a.zip, a.country ORDER BY u.username"
 *      ),
 * })
 * @SqlResultSetMappings({
 *      @SqlResultSetMapping(
 *          name    = "mappingUserPhonenumberCount",
 *          entities= {
 *              @EntityResult(
 *                  entityClass = "User",
 *                  fields      = {
 *                      @FieldResult(name = "id"),
 *                      @FieldResult(name = "name"),
 *                      @FieldResult(name = "status"),
 *                  }
 *              )
 *          },
 *          columns = {
 *              @ColumnResult("numphones")
 *          }
 *      ),
 *      @SqlResultSetMapping(
 *          name    = "mappingMultipleJoinsEntityResults",
 *          entities= {
 *              @EntityResult(
 *                  fields      = {
 *                      @FieldResult(name = "id",       column="u_id"),
 *                      @FieldResult(name = "name",     column="u_name"),
 *                      @FieldResult(name = "status",   column="u_status"),
 *                  }
 *              ),
 *              @EntityResult(
 *                  entityClass = "Address",
 *                  fields      = {
 *                      @FieldResult(name = "id",       column="a_id"),
 *                      @FieldResult(name = "zip",      column="a_zip"),
 *                      @FieldResult(name = "country",  column="a_country"),
 *                  }
 *              )
 *          },
 *          columns = {
 *              @ColumnResult("numphones")
 *          }
 *      )
 *})
 */
 class User
{
    /** @Id @Column(type="integer") @GeneratedValue */
    public $id;

    /** @Column(type="string", length=50, nullable=true) */
    public $status;

    /** @Column(type="string", length=255, unique=true) */
    public $username;

    /** @Column(type="string", length=255) */
    public $name;

    /** @OneToMany(targetEntity="Phonenumber") */
    public $phonenumbers;

    /** @OneToOne(targetEntity="Address") */
    public $address;

    // ....
```

22.2.34. @Table

Annotation describes the table an entity is persisted in. It is placed on the entity-class PHP DocBlock and is optional. If it is not specified the table name will default to the entity's unqualified classname.

Required attributes:

- **name**: Name of the table

Optional attributes:

- **indexes**: Array of @Index annotations
- **uniqueConstraints**: Array of @UniqueConstraint annotations.

Example:

```
<?php
/**
 * @Entity
 * @Table(name="user",
 *      uniqueConstraints={@UniqueConstraint(name="user_unique",columns={"us
ame"})},
 *      indexes={@Index(name="user_idx", columns={"email"})}
 * )
 */
class User { }
```

22.2.35. @UniqueConstraint

Annotation is used inside the **@Table** annotation on the entity-class level. It allows to hint the SchemaTool to generate a database unique constraint on the specified table columns. It only has meaning in the SchemaTool schema generation context.

Required attributes:

- **name**: Name of the Index

Optional attributes:

- **options**: Array of platform specific options:
    - **`where`**: SQL WHERE condition to be used for partial indexes. It will only have e on supported platforms.

Example:

```
<?php
/**
 * @Entity
 * @Table(name="ecommerce_products",uniqueConstraints={@UniqueConstraint(nam
search_idx", columns={"name", "email"})})
 */
class ECommerceProduct
{
}
```

22.2.36. @Version

Marker annotation that defines a specified column as version attribute used in an optimistic locking scenario. It only works on **@Column** annotations that have the type integer or datetime. Combining @Version with **@Id** is not supported.

Example:

```
<?php
/**
 * @Column(type="integer")
 * @Version
 */
protected $version;
```