27. Mapping annotations27.1. Jakarta Persistence annotations 27.1.1. @Access The @Access annotation is used to specify the access type of the associated entity class, mapped superclass, or embeddable class, or entity attribute. See the Access type section for more info. 27.1.2. @AssociationOverride The @AssociationOverride annotation is used to override an association mapping (e.g. @ManyToOne, @OneToOne, @OneToMany, @ManyToMany) inherited from a mapped superclass or an embeddable. See the Overriding Embeddable types section for more info. 27.1.3. @AssociationOverrides The @AssociationOverrides is used to group several @AssociationOverride annotations. 27.1.4. @AttributeOverride The @AttributeOverride annotation is used to override an attribute mapping inherited from a mapped superclass or an embeddable. See the Overriding Embeddable types section for more info. 27.1.5. @AttributeOverrides The @AttributeOverrides is used to group several @AttributeOverride annotations. 27.1.6. @Basic The @Basic annotation is used to map a basic attribute type to a database column. See the Basic types chapter for more info. 27.1.7. @Cacheable The @Cacheable annotation is used to specify whether an entity should be stored in the second-level cache. If the persistence.xml shared-cache-mode XML attribute is set to ENABLE\_SELECTIVE, then only the entities annotated with the @Cacheable are going to be stored in the second-level cache. If shared-cache-mode XML attribute value is DISABLE\_SELECTIVE, then the entities marked with the @Cacheable annotation are not going to be stored in the second-level cache, while all the other entities are stored in the cache. See the Caching chapter for more info. 27.1.8. @CollectionTable The @CollectionTable annotation is used to specify the database table that stores the values of a basic or an embeddable type collection. See the Collections of embeddable types section for more info. 27.1.9. @Column The @Column annotation is used to specify the mapping between a basic entity attribute and the database table column. See the @Column annotation section for more info. 27.1.10. @ColumnResult The @ColumnResult annotation is used in conjunction with the @SqlResultSetMapping or @ConstructorResult annotations to map a SQL column for a given SELECT query. See the Entity associations with named native queries section for more info. 27.1.11. @ConstructorResult The @ConstructorResult annotation is used in conjunction with the @SqlResultSetMapping annotations to map columns of a given SELECT query to a certain object constructor. See the Multiple scalar values NamedNativeQuery with ConstructorResult section for more info. 27.1.12. @Convert The @Convert annotation is used to specify the AttributeConverter implementation used to convert the currently annotated basic attribute. If the AttributeConverter uses autoApply, then all entity attributes with the same target type are going to be converted automatically. See the AttributeConverter section for more info. 27.1.13. @Converter The @Converter annotation is used to specify that the currently annotated AttributeConverter implementation can be used as a Jakarta Persistence basic attribute converter. specj If the autoApply attribute is set to true, then the Jakarta Persistence provider will automatically convert all basic attributes with the same Java type as defined by the current converter. See the AttributeConverter section for more info. 27.1.14. @Converts The @Converts annotation is used to group multiple @Convert annotations. See the AttributeConverter section for more info. 27.1.15. @DiscriminatorColumn The @DiscriminatorColumn annotation is used to specify the discriminator column name and the discriminator type for the SINGLE\_TABLE and JOINED inheritance strategies. See the Discriminator section for more info. 27.1.16. @DiscriminatorValue The @DiscriminatorValue annotation is used to specify what value of the discriminator column is used for mapping the currently annotated entity. See the Discriminator section for more info. 27.1.17. @ElementCollection The @ElementCollection annotation is used to specify a collection of a basic or embeddable types. See the Collections section for more info. 27.1.18. @Embeddable The @Embeddable annotation is used to specify embeddable types. Like basic types, embeddable types do not have any identity, being managed by their owning entity. See the Embeddables section for more info. 27.1.19. @Embedded The @Embedded annotation is used to specify that a given entity attribute represents an embeddable type. See the Embeddables section for more info. 27.1.20. @EmbeddedId The @EmbeddedId annotation is used to specify the entity identifier is an embeddable type. See the Composite identifiers with @EmbeddedId section for more info. 27.1.21. @Entity The @Entity annotation is used to specify that the currently annotated class represents an entity type. Unlike basic and embeddable types, entity types have an identity and their state is managed by the underlying Persistence Context. See the Entity section for more info. 27.1.22. @EntityListeners The @EntityListeners annotation is used to specify an array of callback listener classes that are used by the currently annotated entity. See the Jakarta Persistence callbacks section for more info. 27.1.23. @EntityResult The @EntityResult annotation is used with the @SqlResultSetMapping annotation to map the selected columns to an entity. See the Entity associations with named native queries section for more info. 27.1.24. @Enumerated The @Enumerated annotation is used to specify that an entity attribute represents an enumerated type. See the @Enumerated basic type section for more info. 27.1.25. @ExcludeDefaultListeners The @ExcludeDefaultListeners annotation is used to specify that the currently annotated entity skips the invocation of any default listener. See the Exclude default entity listeners section for more info. 27.1.26. @ExcludeSuperclassListeners The @ExcludeSuperclassListeners annotation is used to specify that the currently annotated entity skips the invocation of listeners declared by its superclass. See the Exclude default entity listeners section for more info. 27.1.27. @FieldResult The @FieldResult annotation is used with the @EntityResult annotation to map the selected columns to the fields of some specific entity. See the Entity associations with named native queries section for more info. 27.1.28. @ForeignKey The @ForeignKey annotation is used to specify the associated foreign key of a @JoinColumn mapping. The @ForeignKey annotation is only used if the automated schema generation tool is enabled, in which case, it allows you to customize the underlying foreign key definition. See the @ManyToOne with @ForeignKey section for more info. 27.1.29. @GeneratedValue The @GeneratedValue annotation specifies that the entity identifier value is automatically generated using an identity column, a database sequence, or a table generator. Hibernate supports the @GeneratedValue mapping even for UUID identifiers. See the Automatically-generated identifiers section for more info. 27.1.30. @Id The @Id annotation specifies the entity identifier. An entity must always have an identifier attribute which is used when loading the entity in a given Persistence Context. See the Identifiers section for more info. 27.1.31. @IdClass The @IdClass annotation is used if the current entity defined a composite identifier. A separate class encapsulates all the identifier attributes, which are mirrored by the current entity mapping. See the Composite identifiers with @IdClass section for more info. 27.1.32. @Index The @Index annotation is used by the automated schema generation tool to create a database index. See the Columns index chapter for more info. 27.1.33. @Inheritance The @Inheritance annotation is used to specify the inheritance strategy of a given entity class hierarchy. See the Inheritance section for more info. 27.1.34. @JoinColumn The @JoinColumn annotation is used to specify the FOREIGN KEY column used when joining an entity association or an embeddable collection. See the @ManyToOne with @JoinColumn section for more info. 27.1.35. @JoinColumns The @JoinColumns annotation is used to group multiple @JoinColumn annotations, which are used when mapping entity association or an embeddable collection using a composite identifier. 27.1.36. @JoinTable The @JoinTable annotation is used to specify the link table between two other database tables. See the @JoinTable mapping section for more info. 27.1.37. @Lob The @Lob annotation is used to specify that the currently annotated entity attribute represents a large object type. See the BLOB mapping section for more info. 27.1.38. @ManyToMany The @ManyToMany annotation is used to specify a many-to-many database relationship. See the @ManyToMany mapping section for more info. 27.1.39. @ManyToOne The @ManyToOne annotation is used to specify a many-to-one database relationship. See the @ManyToOne mapping section for more info. 27.1.40. @MapKey The @MapKey annotation is used to specify the key of a java.util.Map association for which the key type is either the primary key or an attribute of the entity which represents the value of the map. See the @MapKey mapping section for more info. 27.1.41. @MapKeyClass The @MapKeyClass annotation is used to specify the type of the map key of a java.util.Map associations. See the @MapKeyClass mapping section for more info. 27.1.42. @MapKeyColumn The @MapKeyColumn annotation is used to specify the database column which stores the key of a java.util.Map association for which the map key is a basic type. See the @MapKeyType mapping section for an example of @MapKeyColumn annotation usage. 27.1.43. @MapKeyEnumerated The @MapKeyEnumerated annotation is used to specify that the key of java.util.Map association is a Java Enum. See the @MapKeyEnumerated mapping section for more info. 27.1.44. @MapKeyJoinColumn The @MapKeyJoinColumn annotation is used to specify that the key of java.util.Map association is an entity association. The map key column is a FOREIGN KEY in a link table that also joins the Map owner’s table with the table where the Map value resides. See the @MapKeyJoinColumn mapping section for more info. 27.1.45. @MapKeyJoinColumns The @MapKeyJoinColumns annotation is used to group several @MapKeyJoinColumn mappings when the java.util.Map association key uses a composite identifier. 27.1.46. @MapKeyTemporal The @MapKeyTemporal annotation is used to specify that the key of java.util.Map association is a @TemporalType (e.g. DATE, TIME, TIMESTAMP). See the @MapKeyTemporal mapping section for more info. 27.1.47. @MappedSuperclass The @MappedSuperclass annotation is used to specify that the currently annotated type attributes are inherited by any subclass entity. See the @MappedSuperclass section for more info. 27.1.48. @MapsId The @MapsId annotation is used to specify that the entity identifier is mapped by the currently annotated @ManyToOne or @OneToOne association. See the @MapsId mapping section for more info. 27.1.49. @NamedAttributeNode The @NamedAttributeNode annotation is used to specify each individual attribute node that needs to be fetched by an Entity Graph. See the Fetch graph section for more info. 27.1.50. @NamedEntityGraph The @NamedEntityGraph annotation is used to specify an Entity Graph that can be used by an entity query to override the default fetch plan. See the Fetch graph section for more info. 27.1.51. @NamedEntityGraphs The @NamedEntityGraphs annotation is used to group multiple @NamedEntityGraph annotations. 27.1.52. @NamedNativeQueries The @NamedNativeQueries annotation is used to group multiple @NamedNativeQuery annotations. See the Custom CRUD mapping section for more info. 27.1.53. @NamedNativeQuery The @NamedNativeQuery annotation is used to specify a native SQL query that can be retrieved later by its name. See the Custom CRUD mapping section for more info. 27.1.54. @NamedQueries The @NamedQueries annotation is used to group multiple @NamedQuery annotations. 27.1.55. @NamedQuery The @NamedQuery annotation is used to specify a JPQL query that can be retrieved later by its name. See the @NamedQuery section for more info. 27.1.56. @NamedStoredProcedureQueries The @NamedStoredProcedureQueries annotation is used to group multiple @NamedStoredProcedureQuery annotations. 27.1.57. @NamedStoredProcedureQuery The @NamedStoredProcedureQuery annotation is used to specify a stored procedure query that can be retrieved later by its name. See the Using named queries to call stored procedures section for more info. 27.1.58. @NamedSubgraph The @NamedSubgraph annotation used to specify a subgraph in an Entity Graph. See the Fetch subgraph section for more info. 27.1.59. @OneToMany The @OneToMany annotation is used to specify a one-to-many database relationship. See the @OneToMany mapping section for more info. 27.1.60. @OneToOne The @OneToOne annotation is used to specify a one-to-one database relationship. See the @OneToOne mapping section for more info. 27.1.61. @OrderBy The @OrderBy annotation is used to specify the entity attributes used for sorting when fetching the currently annotated collection. See the @OrderBy mapping section for more info. 27.1.62. @OrderColumn The @OrderColumn annotation is used to specify that the current annotation collection order should be materialized in the database. See the @OrderColumn mapping section for more info. 27.1.63. @PersistenceContext The @PersistenceContext annotation is used to specify the EntityManager that needs to be injected as a dependency. See the @PersistenceContext mapping section for more info. 27.1.64. @PersistenceContexts The @PersistenceContexts annotation is used to group multiple @PersistenceContext annotations. 27.1.65. @PersistenceProperty The @PersistenceProperty annotation is used by the @PersistenceContext annotation to declare Jakarta Persistence provider properties that are passed to the underlying container when the EntityManager instance is created. See the @PersistenceProperty mapping section for more info. 27.1.66. @PersistenceUnit The @PersistenceUnit annotation is used to specify the EntityManagerFactory that needs to be injected as a dependency. See the @PersistenceUnit mapping section for more info. 27.1.67. @PersistenceUnits The @PersistenceUnits annotation is used to group multiple @PersistenceUnit annotations. 27.1.68. @PostLoad The @PostLoad annotation is used to specify a callback method that fires after an entity is loaded. See the Jakarta Persistence callbacks section for more info. 27.1.69. @PostPersist The @PostPersist annotation is used to specify a callback method that fires after an entity is persisted. See the Jakarta Persistence callbacks section for more info. 27.1.70. @PostRemove The @PostRemove annotation is used to specify a callback method that fires after an entity is removed. See the Jakarta Persistence callbacks section for more info. 27.1.71. @PostUpdate The @PostUpdate annotation is used to specify a callback method that fires after an entity is updated. See the Jakarta Persistence callbacks section for more info. 27.1.72. @PrePersist The @PrePersist annotation is used to specify a callback method that fires before an entity is persisted. See the Jakarta Persistence callbacks section for more info. 27.1.73. @PreRemove The @PreRemove annotation is used to specify a callback method that fires before an entity is removed. See the Jakarta Persistence callbacks section for more info. 27.1.74. @PreUpdate The @PreUpdate annotation is used to specify a callback method that fires before an entity is updated. See the Jakarta Persistence callbacks section for more info. 27.1.75. @PrimaryKeyJoinColumn The @PrimaryKeyJoinColumn annotation is used to specify that the primary key column of the currently annotated entity is also a foreign key to some other entity (e.g. a base class table in a JOINED inheritance strategy, the primary table in a secondary table mapping, or the parent table in a @OneToOne relationship). See the @PrimaryKeyJoinColumn mapping section for more info. 27.1.76. @PrimaryKeyJoinColumns The @PrimaryKeyJoinColumns annotation is used to group multiple @PrimaryKeyJoinColumn annotations. 27.1.77. @QueryHint The @QueryHint annotation is used to specify a Jakarta Persistence provider hint used by a @NamedQuery or a @NamedNativeQuery annotation. See the @QueryHint section for more info. 27.1.78. @SecondaryTable The @SecondaryTable annotation is used to specify a secondary table for the currently annotated entity. See the @SecondaryTable mapping section for more info. 27.1.79. @SecondaryTables The @SecondaryTables annotation is used to group multiple @SecondaryTable annotations. 27.1.80. @SequenceGenerator The @SequenceGenerator annotation is used to specify the database sequence used by the identifier generator of the currently annotated entity. See the @SequenceGenerator mapping section for more info. 27.1.81. @SqlResultSetMapping The @SqlResultSetMapping annotation is used to specify the ResultSet mapping of a native SQL query or stored procedure. See the SqlResultSetMapping mapping section for more info. 27.1.82. @SqlResultSetMappings The @SqlResultSetMappings annotation is group multiple @SqlResultSetMapping annotations. 27.1.83. @StoredProcedureParameter The @StoredProcedureParameter annotation is used to specify a parameter of a @NamedStoredProcedureQuery. See the Using named queries to call stored procedures section for more info. 27.1.84. @Table The @Table annotation is used to specify the primary table of the currently annotated entity. See the @Table mapping section for more info. 27.1.85. @TableGenerator The @TableGenerator annotation is used to specify the database table used by the identity generator of the currently annotated entity. See the @TableGenerator mapping section for more info. 27.1.86. @Temporal The @Temporal annotation is used to specify the TemporalType of the currently annotated java.util.Date or java.util.Calendar entity attribute. See the Basic temporal types chapter for more info. 27.1.87. @Transient The @Transient annotation is used to specify that a given entity attribute should not be persisted. See the @Transient mapping section for more info. 27.1.88. @UniqueConstraint The @UniqueConstraint annotation is used to specify a unique constraint to be included by the automated schema generator for the primary or secondary table associated with the currently annotated entity. See the Columns unique constraint chapter for more info. 27.1.89. @Version The @Version annotation is used to specify the version attribute used for optimistic locking. See the Optimistic locking mapping section for more info. 27.2. Hibernate annotations 27.2.1. @Any The @Any annotation is used to define the any-to-one association which can point to one of several entity types. See the @Any mapping section for more info. 27.2.2. @AnyDiscriminator The @AnyDiscriminator annotation is used to provide details about the discriminator portion of an @Any or @ManyToAny mapping. See the @Any mapping section for more info. 27.2.3. @AnyDiscriminatorValue The @AnyDiscriminatorValue annotation maps a single discriminator value to its corresponding entity See the @Any mapping section for more info. 27.2.4. @AnyDiscriminatorValues The @AnyDiscriminatorValues annotation groups multiple @AnyDiscriminatorValue annotations. See the @Any mapping section for more info. 27.2.5. @AnyKeyJavaClass The @AnyKeyJavaClass annotation specifies the Java Class to use for the foreign-key of an ANY mapping See the @Any mapping section for more info. 27.2.6. @AnyKeyJavaType The @AnyKeyJavaType annotation specifies a specific JavaType descriptor to use for the foreign-key of an ANY mapping See the @Any mapping section for more info. 27.2.7. @AnyKeyJdbcType The @AnyKeyJdbcType annotation specifies a specific JdbcType descriptor to use for the foreign-key of an ANY mapping See the @Any mapping section for more info. 27.2.8. @AnyKeyJdbcTypeCode The @AnyKeyJdbcTypeCode annotation specifies a "type code" indicating which JdbcType descriptor to use for the foreign-key of an ANY mapping See the @Any mapping section for more info. 27.2.9. @AttributeAccessor The @AttributeAccessor annotation is used to specify a custom PropertyAccessStrategy. Should only be used to name a custom PropertyAccessStrategy. For property/field access type, the Jakarta Persistence @Access annotation should be preferred. However, if this annotation is used with either value="property" or value="field", it will act just as the corresponding usage of the Jakarta Persistence @Access annotation. 27.2.10. @AttributeBinderType The @AttributeBinderType annotation is a meta-annotation used to annotate a custom annotation type used to drive customized model binding. See Customizing the domain model. 27.2.11. @BatchSize The @BatchSize annotation is used to specify the size for batch loading the entries of a lazy collection. See the Batch fetching section for more info. 27.2.12. @Cache The @Cache annotation is used to specify the CacheConcurrencyStrategy of a root entity or a collection. See the Caching chapter for more info. 27.2.13. @Cascade The @Cascade annotation is used to apply the Hibernate specific CascadeType strategies (e.g. CascadeType.LOCK, CascadeType.SAVE\_UPDATE, CascadeType.REPLICATE) on a given association. For Jakarta Persistence cascading, prefer using the jakarta.persistence.CascadeType instead. When combining both Jakarta Persistence and Hibernate CascadeType strategies, Hibernate will merge both sets of cascades. See the Cascading chapter for more info. 27.2.14. @Check The @Check annotation is used to specify an arbitrary SQL CHECK constraint which can be defined at the class level. See the Database-level checks chapter for more info. 27.2.15. @CollectionId The @CollectionId annotation is used to specify an identifier column for an idbag collection. You might want to use the Jakarta Persistence @OrderColumn instead. 27.2.16. @CollectionType The @CollectionType annotation is used to specify a custom collection type. The collection can also name a @Type, which defines the Hibernate Type of the collection elements. See the Custom collection types chapter for more info. 27.2.17. @ColumnDefault The @ColumnDefault annotation is used to specify the DEFAULT DDL value to apply when using the automated schema generator. The same behavior can be achieved using the definition attribute of the Jakarta Persistence @Column annotation. See the Default value for a database column chapter for more info. 27.2.18. @Columns The @Columns annotation is used to group multiple Jakarta Persistence @Column annotations. See the @Columns mapping section for more info. 27.2.19. @ColumnTransformer The @ColumnTransformer annotation is used to customize how a given column value is read from or written into the database. See the @ColumnTransformer mapping section for more info. 27.2.20. @ColumnTransformers The @ColumnTransformers annotation is used to group multiple @ColumnTransformer annotations. 27.2.21. @CreationTimestamp The @CreationTimestamp annotation is used to specify that the currently annotated temporal type must be initialized with the current JVM timestamp value. The supported property types are: java.util.Date java.util.Calendar java.sql.Date java.sql.Time java.sql.Timestamp java.time.Instant java.time.LocalDate java.time.LocalDateTime java.time.LocalTime java.time.MonthDay java.time.OffsetDateTime java.time.OffsetTime java.time.Year java.time.YearMonth java.time.ZonedDateTime See the @CreationTimestamp mapping section for more info. 27.2.22. @DiscriminatorFormula The @DiscriminatorFormula annotation is used to specify a Hibernate @Formula to resolve the inheritance discriminator value. See the @DiscriminatorFormula section for more info. 27.2.23. @DiscriminatorOptions The @DiscriminatorOptions annotation is used to provide the force and insert Discriminator properties. See the Discriminator section for more info. 27.2.24. @DynamicInsert The @DynamicInsert annotation is used to specify that the INSERT SQL statement should be generated whenever an entity is to be persisted. By default, Hibernate uses a cached INSERT statement that sets all table columns. When the entity is annotated with the @DynamicInsert annotation, the PreparedStatement is going to include only the non-null columns. See the @CreationTimestamp mapping section for more info on how @DynamicInsert works. 27.2.25. @DynamicUpdate The @DynamicUpdate annotation is used to specify that the UPDATE SQL statement should be generated whenever an entity is modified. By default, Hibernate uses a cached UPDATE statement that sets all table columns. When the entity is annotated with the @DynamicUpdate annotation, the PreparedStatement is going to include only the columns whose values have been changed. See the @DynamicUpdate section for more info. For reattachment of detached entities, the dynamic update is not possible without having the @SelectBeforeUpdate annotation as well. 27.2.26. @EmbeddableInstantiator The @EmbeddableInstantiator annotation is used to specify a custom instantiator for a specific embedded. See the Custom instantiation section for more info. 27.2.27. @EmbeddableInstantiatorRegistration The @EmbeddableInstantiatorRegistration annotation is used to register a custom instantiator implementation to be used for all references to a particular Embeddable. See the Custom instantiation section for more info. 27.2.28. @EmbeddableInstantiatorRegistrations The @EmbeddableInstantiatorRegistrations annotation is used to group multiple @EmbeddableInstantiatorRegistration annotations. 27.2.29. @Fetch The @Fetch annotation is used to specify the Hibernate specific FetchMode (e.g. JOIN, SELECT, SUBSELECT) used for the currently annotated association. See the @Fetch mapping section for more info. 27.2.30. @FetchProfile The @FetchProfile annotation is used to specify a custom fetching profile, similar to a Jakarta Persistence Entity Graph. See the Fetch mapping section for more info. 27.2.31. @FetchProfile.FetchOverride The @FetchProfile.FetchOverride annotation is used in conjunction with the @FetchProfile annotation, and it’s used for overriding the fetching strategy of a particular entity association. See the Fetch profile section for more info. 27.2.32. @FetchProfiles The @FetchProfiles annotation is used to group multiple @FetchProfile annotations. 27.2.33. @Filter The @Filter annotation is used to add filters to an entity or the target entity of a collection. See the Filter mapping section for more info. 27.2.34. @FilterDef The @FilterDef annotation is used to specify a @Filter definition (name, default condition and parameter types, if any). See the Filter mapping section for more info. 27.2.35. @FilterDefs The @FilterDefs annotation is used to group multiple @FilterDef annotations. 27.2.36. @FilterJoinTable The @FilterJoinTable annotation is used to add @Filter capabilities to a join table collection. See the FilterJoinTable mapping section for more info. 27.2.37. @FilterJoinTables The @FilterJoinTables annotation is used to group multiple @FilterJoinTable annotations. 27.2.38. @Filters The @Filters annotation is used to group multiple @Filter annotations. 27.2.39. @ForeignKey 27.2.40. @Formula The @Formula annotation is used to specify an SQL fragment that is executed in order to populate a given entity attribute. See the @Formula mapping section for more info. 27.2.41. @Generated The @Generated annotation is used to specify that the currently annotated entity attribute is generated by the database. See the @Generated mapping section for more info. 27.2.42. @GeneratedColumn The @GeneratedColumn annotation is used to specify that an entity attribute is generated by the database using GENERATED ALWAYS AS DDL. 27.2.43. @GeneratorType The @GeneratorType annotation is used to provide a ValueGenerator and a GenerationTime for the currently annotated generated attribute. See the @GeneratorType mapping section for more info. 27.2.44. @GenericGenerator The @GenericGenerator annotation can be used to configure any Hibernate identifier generator. See the @GenericGenerator mapping section for more info. 27.2.45. @GenericGenerators The @GenericGenerators annotation is used to group multiple @GenericGenerator annotations. 27.2.46. @Immutable The @Immutable annotation is used to specify that the annotated entity, attribute, or collection is immutable. See the @Immutable mapping section for more info. 27.2.47. @Index The @Index annotation is deprecated. Use the Jakarta Persistence @Index annotation instead. 27.2.48. @IndexColumn The @IndexColumn annotation is deprecated. Use the Jakarta Persistence @OrderColumn annotation instead. 27.2.49. @JoinColumnOrFormula The @JoinColumnOrFormula annotation is used to specify that the entity association is resolved either through a FOREIGN KEY join (e.g. @JoinColumn) or using the result of a given SQL formula (e.g. @JoinFormula). See the @JoinColumnOrFormula mapping section for more info. 27.2.50. @JoinColumnsOrFormulas The @JoinColumnsOrFormulas annotation is used to group multiple @JoinColumnOrFormula annotations. 27.2.51. @JoinFormula The @JoinFormula annotation is used as a replacement for @JoinColumn when the association does not have a dedicated FOREIGN KEY column. See the @JoinFormula mapping section for more info. 27.2.52. @LazyCollection The @LazyCollection annotation is deprecated. 27.2.53. @LazyGroup The @LazyGroup annotation is used to specify that an entity attribute should be fetched along with all the other attributes belonging to the same group. To load entity attributes lazily, bytecode enhancement is needed. By default, all non-collection attributes are loaded in one group named "DEFAULT". This annotation allows defining different groups of attributes to be initialized together when access one attribute in the group. See the @LazyGroup mapping section for more info. 27.2.54. @LazyToOne The @LazyToOne annotation is deprecated. 27.2.55. @ListIndexBase The @ListIndexBase annotation is used to specify the start value for a list index, as stored in the database. By default, List indexes are stored starting at zero. Generally used in conjunction with @OrderColumn. See the @ListIndexBase mapping section for more info. 27.2.56. @Loader The @Loader annotation is used to override the default SELECT query used for loading an entity. See the Custom CRUD mapping section for more info. 27.2.57. @ManyToAny The @ManyToAny annotation is used to specify a many-to-one association when the target type is dynamically resolved. See the @ManyToAny mapping section for more info. 27.2.58. @MapKeyType The @MapKeyType annotation is used to specify the map key type. See the @MapKeyType mapping section for more info. 27.2.59. @MetaValue The @MetaValue annotation is used by the [annotations-hibernate-anymetadef] annotation to specify the association between a given discriminator value and an entity type. See the @Any mapping section for more info. 27.2.60. @NamedNativeQueries The @NamedNativeQueries annotation is used to group multiple @NamedNativeQuery annotations. 27.2.61. @NamedNativeQuery The @NamedNativeQuery annotation extends the Jakarta Persistence @NamedNativeQuery with Hibernate specific features, like: flush mode for this particular query if the query should be cached, and which cache region should be used the selected entity CacheModeType strategy the JDBC Statement fetch size the JDBC Statement execution timeout if the query is a CallableStatement, targeting a stored procedure or a database function what SQL-level comment should be sent to the database if the query is read-only, hence it does not store the resulted entities into the currently running Persistence Context See the Hibernate @NamedNativeQuery section for more info. 27.2.62. @NamedQueries The @NamedQueries annotation is used to group multiple @NamedQuery annotations. 27.2.63. @NamedQuery The @NamedQuery annotation extends the Jakarta Persistence @NamedQuery with Hibernate specific features, like: flush mode for this particular query if the query should be cached, and which cache region should be used the selected entity CacheModeType strategy the JDBC Statement fetch size the JDBC Statement execution timeout if the query is a CallableStatement, targeting a stored procedure or a database function what SQL-level comment should be sent to the database if the query is read-only, hence it does not store the resulted entities into the currently running Persistence Context See the @NamedQuery section for more info. 27.2.64. @Nationalized The @Nationalized annotation is used to specify that the currently annotated attribute is a character type (e.g. String, Character, Clob) that is stored in a nationalized column type (NVARCHAR, NCHAR, NCLOB). See the @Nationalized mapping section for more info. 27.2.65. @NaturalId The @NaturalId annotation is used to specify that the currently annotated attribute is part of the natural id of the entity. See the Natural Ids section for more info. 27.2.66. @NaturalIdCache The @NaturalIdCache annotation is used to specify that the natural id values associated with the annotated entity should be stored in the second-level cache. See the @NaturalIdCache mapping section for more info. 27.2.67. @NotFound The @NotFound annotation is used to specify the NotFoundAction strategy for when an element is not found in a given association. The NotFoundAction defines two possibilities: EXCEPTION An exception is thrown when an element is not found (default and recommended). IGNORE Ignore the element when not found in the database. See the @NotFound mapping section for more info. 27.2.68. @OnDelete The @OnDelete annotation is used to specify the delete strategy employed by the currently annotated collection, array or joined subclasses. This annotation is used by the automated schema generation tool to generated the appropriate FOREIGN KEY DDL cascade directive. The two possible strategies are defined by the OnDeleteAction enumeration: CASCADE Use the database FOREIGN KEY cascade capabilities. NO\_ACTION Take no action. See the @OnDelete cascade chapter for more info. 27.2.69. @OptimisticLock The @OptimisticLock annotation is used to specify if the currently annotated attribute will trigger an entity version increment upon being modified. See the Excluding attributes section for more info. 27.2.70. @OptimisticLocking The @OptimisticLocking annotation is used to specify the currently annotated entity’s optimistic locking strategy. The four possible strategies are defined by the OptimisticLockType enumeration: NONE The implicit optimistic locking mechanism is disabled. VERSION The implicit optimistic locking mechanism is using a dedicated version column. ALL The implicit optimistic locking mechanism is using all attributes as part of an expanded WHERE clause restriction for the UPDATE and DELETE SQL statements. DIRTY The implicit optimistic locking mechanism is using the dirty attributes (the attributes that were modified) as part of an expanded WHERE clause restriction for the UPDATE and DELETE SQL statements. See the Versionless optimistic locking section for more info. 27.2.71. @OrderBy The @OrderBy annotation is used to specify a SQL ordering directive for sorting the currently annotated collection. It differs from the Jakarta Persistence @OrderBy annotation because the Jakarta Persistence annotation expects a JPQL order-by fragment, not an SQL directive. See the @OrderBy mapping section for more info. 27.2.72. @ParamDef The @ParamDef annotation is used in conjunction with @FilterDef so that the Hibernate Filter can be customized with runtime-provided parameter values. See the Filter mapping section for more info. 27.2.73. @Parameter The @Parameter annotation is a generic parameter (basically a key/value combination) used to parametrize other annotations, like @CollectionType, @GenericGenerator, and @Type, @TypeDef. 27.2.74. @Parent The @Parent annotation is used to specify that the currently annotated embeddable attribute references back the owning entity. See the @Parent mapping section for more info. 27.2.75. @PartitionKey The @PartitionKey annotation is used to identify a field of an entity that holds the partition key of a table. See the @PartitionKey mapping section for more info. 27.2.76. @Persister The @Persister annotation is used to specify a custom entity or collection persister. For entities, the custom persister must implement the EntityPersister interface. For collections, the custom persister must implement the CollectionPersister interface. See the @Persister mapping section for more info. 27.2.77. @Polymorphism The @Polymorphism annotation is used to define the PolymorphismType Hibernate will apply to entity hierarchies. There are two possible PolymorphismType options: EXPLICIT The currently annotated entity is retrieved only if explicitly asked. IMPLICIT The currently annotated entity is retrieved if any of its super entities are retrieved. This is the default option. See the @Polymorphism section for more info. 27.2.78. @Proxy The @Proxy annotation is used to specify a custom proxy implementation for the currently annotated entity. See the @Proxy mapping section for more info. 27.2.79. @RowId The @RowId annotation is used to specify the database column used as a ROWID pseudocolumn. For instance, Oracle defines the ROWID pseudocolumn which provides the address of every table row. According to Oracle documentation, ROWID is the fastest way to access a single row from a table. See the @RowId mapping section for more info. 27.2.80. @SelectBeforeUpdate The @SelectBeforeUpdate annotation is used to specify that the currently annotated entity state be selected from the database when determining whether to perform an update when the detached entity is reattached. See the OptimisticLockType.DIRTY mapping section for more info on how @SelectBeforeUpdate works. 27.2.81. @SortComparator The @SortComparator annotation is used to specify a Comparator for sorting the Set/Map in-memory. See the @SortComparator mapping section for more info. 27.2.82. @SortNatural The @SortNatural annotation is used to specify that the Set/Map should be sorted using natural sorting. See the @SortNatural mapping section for more info. 27.2.83. @Source The @Source annotation is used in conjunction with a @Version timestamp entity attribute indicating the SourceType of the timestamp value. The SourceType offers two options: DB Get the timestamp from the database. VM Get the timestamp from the current JVM. See the Database-generated version timestamp mapping section for more info. 27.2.84. @SQLDelete The @SQLDelete annotation is used to specify a custom SQL DELETE statement for the currently annotated entity or collection. See the Custom CRUD mapping section for more info. 27.2.85. @SQLDeleteAll The @SQLDeleteAll annotation is used to specify a custom SQL DELETE statement when removing all elements of the currently annotated collection. See the Custom CRUD mapping section for more info. 27.2.86. @SqlFragmentAlias The @SqlFragmentAlias annotation is used to specify an alias for a Hibernate @Filter. The alias (e.g. myAlias) can then be used in the @Filter condition clause using the {alias} (e.g. {myAlias}) placeholder. See the @SqlFragmentAlias mapping section for more info. 27.2.87. @SQLInsert The @SQLInsert annotation is used to specify a custom SQL INSERT statement for the currently annotated entity or collection. See the Custom CRUD mapping section for more info. 27.2.88. @SQLUpdate The @SQLUpdate annotation is used to specify a custom SQL UPDATE statement for the currently annotated entity or collection. See the Custom CRUD mapping section for more info. 27.2.89. @Subselect The @Subselect annotation is used to specify an immutable and read-only entity using a custom SQL SELECT statement. See the Mapping the entity to a SQL query section for more info. 27.2.90. @Synchronize The @Synchronize annotation is usually used in conjunction with the @Subselect annotation to specify the list of database tables used by the @Subselect SQL query. With this information in place, Hibernate will properly trigger an entity flush whenever a query targeting the @Subselect entity is to be executed while the Persistence Context has scheduled some insert/update/delete actions against the database tables used by the @Subselect SQL query. Therefore, the @Synchronize annotation prevents the derived entity from returning stale data when executing entity queries against the @Subselect entity. See the Mapping the entity to a SQL query section for more info. 27.2.91. @Table The @Table annotation is used to specify additional information to a Jakarta Persistence @Table annotation, like custom INSERT, UPDATE or DELETE statements or a specific FetchMode. See the @SecondaryTable mapping section for more info about Hibernate-specific @Table mapping. 27.2.92. @Tables The @Tables annotation is used to group multiple @Table annotations. 27.2.93. @Target The @Target annotation is used to specify an explicit target implementation when the currently annotated association is using an interface type. See the @Target mapping section for more info. 27.2.94. @Tuplizer The @Tuplizer annotation is used to specify a custom tuplizer for the currently annotated entity or embeddable. For entities, the tupelizer must implement the EntityTuplizer interface. For embeddables, the tupelizer must implement the ComponentTuplizer interface. See the @Tuplizer mapping section for more info. 27.2.95. @Tuplizers The @Tuplizers annotation is used to group multiple @Tuplizer annotations. 27.2.96. @Type The @Type annotation is used to specify the Hibernate @Type used by the currently annotated basic attribute. See the @Type mapping section for more info. 27.2.97. @TypeDef The @TypeDef annotation is used to specify a @Type definition which can later be reused for multiple basic attribute mappings. See the @TypeDef mapping section for more info. 27.2.98. @TypeDefs The @TypeDefs annotation is used to group multiple @TypeDef annotations. 27.2.99. @UpdateTimestamp The @UpdateTimestamp annotation is used to specify that the currently annotated timestamp attribute should be updated with the current JVM timestamp whenever the owning entity gets modified. The supported property types are: java.util.Date java.util.Calendar java.sql.Date java.sql.Time java.sql.Timestamp java.time.Instant java.time.LocalDate java.time.LocalDateTime java.time.LocalTime java.time.MonthDay java.time.OffsetDateTime java.time.OffsetTime java.time.Year java.time.YearMonth java.time.ZonedDateTime See the @UpdateTimestamp mapping section for more info. 27.2.100. @ValueGenerationType The @ValueGenerationType annotation is used to specify that the current annotation type should be used as a generator annotation type. See the @ValueGenerationType mapping section for more info. 27.2.101. @Where The @Where annotation is used to specify a custom SQL WHERE clause used when fetching an entity or a collection. See the @Where mapping section for more info. 27.2.102. @WhereJoinTable The @WhereJoinTable annotation is used to specify a custom SQL WHERE clause used when fetching a join collection table. See the @WhereJoinTable mapping section for more info. 27.2.103. @TenantId The @TenantId annotation identifies a field of an entity that holds a tenant id in discriminator-based multitenancy.