Macro

Relationship(_:deleteRule:minimumModelCount:maximumModelCount:originalName:inverse:hashModifier:)

Specifies the options that SwiftData needs to manage the annotated property as a relationship between two models.

iOS 17.0+ | iPadOS 17.0+ | Mac Catalyst 17.0+ | macOS 14.0+ | tvOS 17.0+ | visionOS 1.0+ | watchOS 10.0+ | Swift 5.9+

```
@attached(peer)
macro Relationship(
    _ options: Schema.Relationship.Option...,
    deleteRule: Schema.Relationship.DeleteRule = .nullify,
    minimumModelCount: Int? = 0,
    maximumModelCount: Int? = 0,
    originalName: String? = nil,
    inverse: AnyKeyPath? = nil,
    hashModifier: String? = nil
)
```

Parameters

`options`
A list of options to apply to the annotated property to customize its behavior. For possible values, see `Schema.Relationship.Option`.

`deleteRule`

The rule to apply when you delete the relationship's owning persistent model. For possible values, see `Schema.Relationship.DeleteRule`. The default value is `Schema.Relationship.DeleteRule.nullify`.

`minimumModelCount`
The minimum number of models the relationship can reference. The default value is `0`.

`maximumModelCount`
The maximum number of models the relationship can reference. The default value is `0`.

`originalName`
The previous name of the attribute, if it's different to the one in the current schema version. The default value is `nil`.

`inverse`
The key path of the relationship that represents the inverse of this relationship. The default value is `nil`.

`hashModifier`
A unique hash value that represents the most recent version of the annotated property. The default value is `nil`.

Mentioned in

Preserving your app's model data across launches

Overview

If one or more of a model's properties represent relationships between their containing model and another model, annotate those properties with the `@Relationship` macro. This enables SwiftData to enforce those relationships at runtime — including what happens if you delete related data – as well as write any associated metadata to the persistent storage so the relationships exist across app launches.

In the following example, a remote image may belong to a category, and a category can contain zero, one, or more images.

```
@Model
class RemoteImage {
    @Attribute(.unique) var sourceURL: URL
    @Relationship(inverse: \Category.images) var category: Category?
    var data: Data

    init(sourceURL: URL, data: Data = Data()) {
        self.sourceURL = sourceURL
        self.data = data
    }
}

@Model
class Category {
    @Attribute(.unique) var name: String
    @Relationship var images = [RemoteImage]()

    init(name: String) {
        self.name = name
    }
}
```

> **Note**
>
> If you declare a relationship attribute as optional when defining your persistent models, SwiftData only enforces `minimumModelCount` and `maximumModelCount` when that attribute isn't nil.

For more information about defining relationships between models, see Defining data relationships with enumerations and model classes.

See Also

Model definition

`macro Model()`

Converts a Swift class into a stored model that's managed by SwiftData.

`macro Attribute(Schema.Attribute.Option..., originalName: String?, hashModifier: String?)`

Specifies the custom behavior that SwiftData applies to the annotated property when managing the owning class.

`macro Unique<T>([PartialKeyPath<T>]...)`

Specifies the key-paths that SwiftData uses to enforce the uniqueness of model instances.

`macro Index<T>([PartialKeyPath<T>]...)`

Specifies the key-paths that SwiftData uses to create one or more binary indices for the associated model.

`macro Index<T>(Schema.Index<T>.Types<T>...)`

Specifies the key-paths that SwiftData uses to create one or more indicies for the associated model, where each index is either binary or R-tree.

{} Defining data relationships with enumerations and model classes

Create relationships for static and dynamic data stored in your app.

`macro Transient()`

Tells SwiftData not to persist the annotated property when managing the owning class.