Table of Contents

[Database Structure 3](#_Toc462133943)

[Relationships between the tables 3](#_Toc462133944)

[Database Design 3](#_Toc462133945)

[Entity Structure 4](#_Toc462133946)

[Difference between FetchTypes 7](#_Toc462133947)

[Remove Operations 9](#_Toc462133948)

[Immutable Entity in JPA 12](#_Toc462133949)

[Projection Queries 14](#_Toc462133950)

[Cascade Operations 15](#_Toc462133951)

[CascadeType.ALL 15](#_Toc462133952)

[PERSIST 15](#_Toc462133953)

[MERGE 16](#_Toc462133954)

[Refresh 17](#_Toc462133955)

[Remove 17](#_Toc462133956)

[Many-to- Many Relationship 19](#_Toc462133957)

[Database Structure 19](#_Toc462133958)

[Entity Relationship 19](#_Toc462133959)

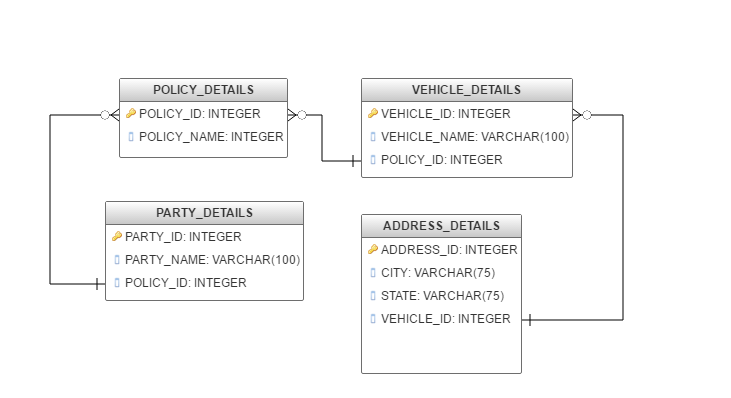
[Join Fetch 21](#_Toc462133960)

# Database Structure

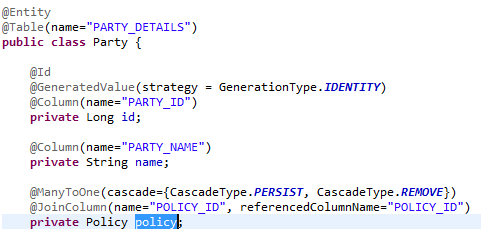
## Relationships between the tables

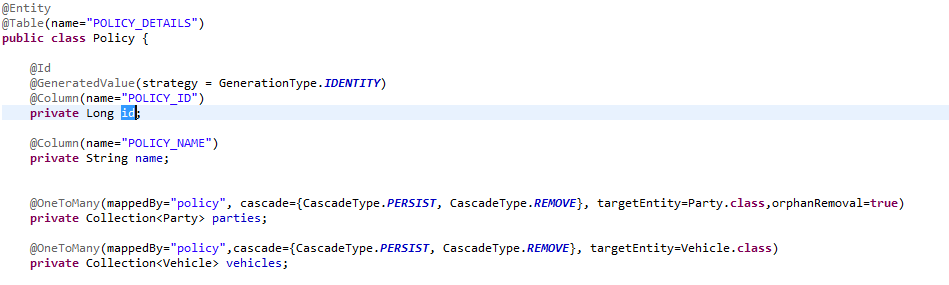
1. PARTY\_DETAILS -> POLICY\_DETAILS – Many-to-One
2. POLICY\_DETAILS -> VEHICLE\_DETAILS – One-to-Many
3. VEHICLE\_DETAILS -> ADDRESS\_DETAILS – One-to-Many

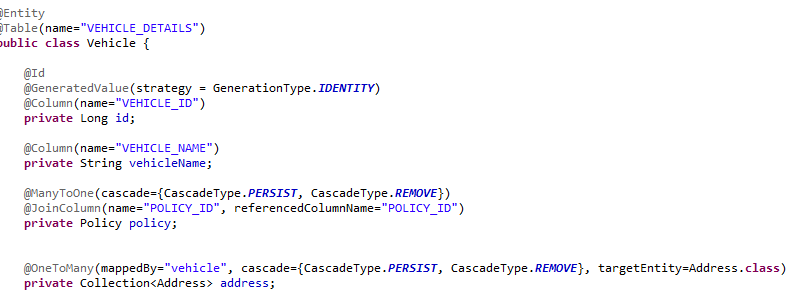
## Database Design

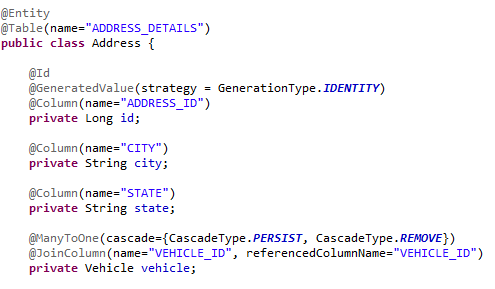


## Entity Structure

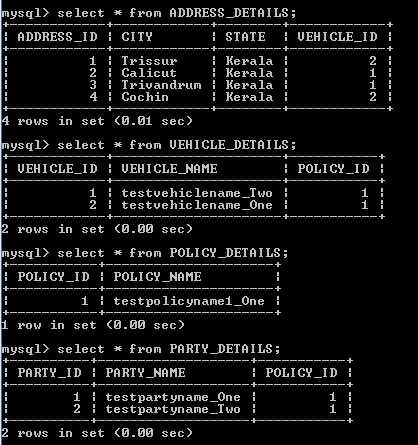






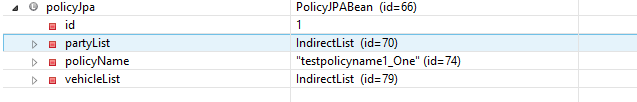


* Database with sample data



# Difference between FetchTypes

* Added FetchType.LAZY
* Only One query triggered: SELECT POLICY\_ID, POLICY\_NAME FROM POLICY\_DETAILS WHERE (POLICY\_NAME = ?)
* Only Policy details loaded in memory



* Modified to FetchType.EAGER
* All queries triggered while fetching policy details
* Queries triggered:

**JPA(Eclipselink)**

SELECT POLICY\_ID, POLICY\_NAME FROM POLICY\_DETAILS WHERE (POLICY\_NAME = ?)

SELECT PARTY\_ID, PARTY\_NAME, POLICY\_ID FROM PARTY\_DETAILS WHERE (POLICY\_ID = ?)

SELECT VEHICLE\_ID, VEHICLE\_NAME, POLICY\_ID FROM VEHICLE\_DETAILS WHERE (POLICY\_ID = ?

SELECT ADDRESS\_ID, CITY, STATE, VEHICLE\_ID FROM ADDRESS\_DETAILS WHERE (VEHICLE\_ID =?

**JPA(Hibernate)**

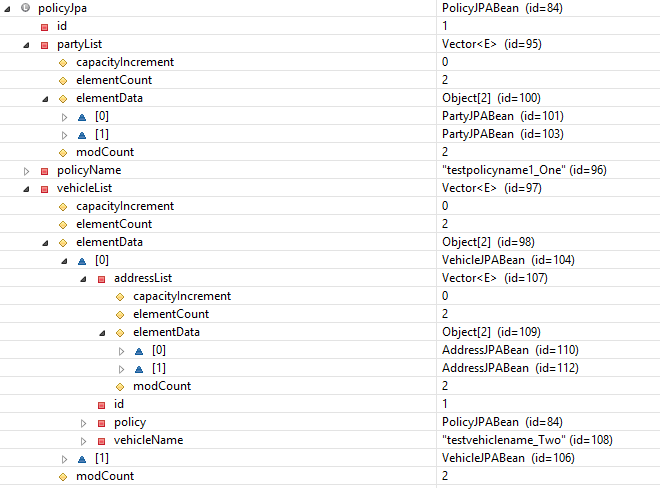
select policy0\_.POLICY\_ID as POLICY\_I1\_2\_, policy0\_.POLICY\_NAME as POLICY\_N2\_2\_ from policy\_details policy0\_ where policy0\_.POLICY\_NAME=?

select vehicles0\_.policy\_id as policy\_i3\_3\_0\_, vehicles0\_.VEHICLE\_ID as VEHICLE\_1\_3\_0\_, vehicles0\_.VEHICLE\_ID as VEHICLE\_1\_3\_1\_, vehicles0\_.policy\_id as policy\_i3\_3\_1\_, vehicles0\_.VEHICLE\_NAME as VEHICLE\_2\_3\_1\_ from vehicle\_details vehicles0\_ where vehicles0\_.policy\_id=?

select parties0\_.policy\_id as policy\_i3\_1\_0\_, parties0\_.PARTY\_ID as PARTY\_ID1\_1\_0\_, parties0\_.PARTY\_ID as PARTY\_ID1\_1\_1\_, parties0\_.PARTY\_NAME as PARTY\_NA2\_1\_1\_, parties0\_.policy\_id as policy\_i3\_1\_1\_ from party\_details parties0\_ where parties0\_.policy\_id=?

Note:

The backend behaviour will be different for different JPA providers.

All details loaded in memory

# Remove Operations

* For removing the relationship between the entities.



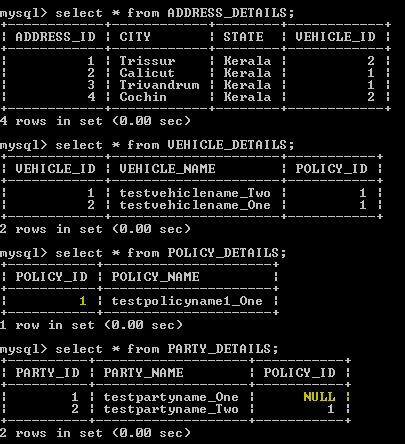
* Will remove the relationship between the parent and child entities
* Below queries are triggered

SELECT POLICY\_ID, POLICY\_NAME FROM POLICY\_DETAILS WHERE (POLICY\_NAME = ?)

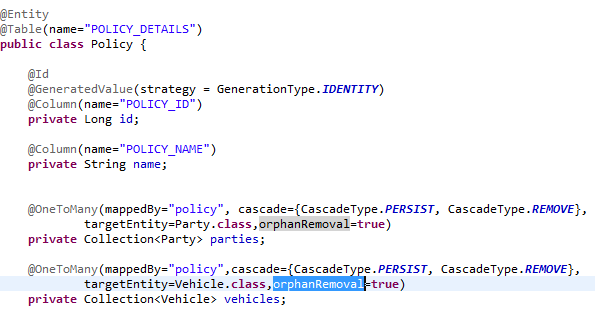
SELECT PARTY\_ID, PARTY\_NAME, POLICY\_ID FROM PARTY\_DETAILS WHERE (POLICY\_ID = ?)

UPDATE PARTY\_DETAILS SET POLICY\_ID = ? WHERE (PARTY\_ID = ?)

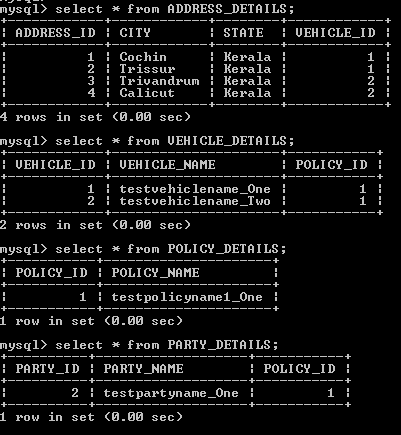
* Database with sample data



* With orphanRemoval=true, the child record will be deleted



* Database with sample data – The orphan record in PARTY\_DETAILS is deleted

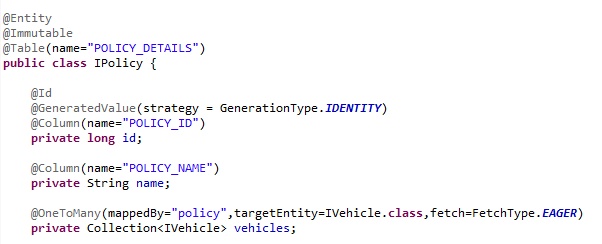


**NOTE:**

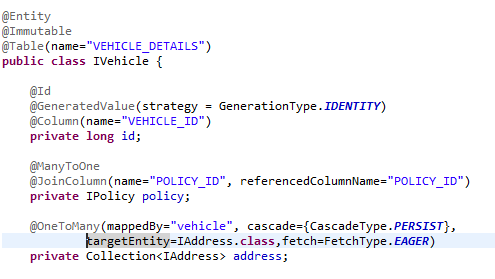
* This behaviour is different for Hibernate JPA provier when compared with eclipselink JPA provider. CascadeType=remove will remove the child entities without orphanRemal=true property

# Immutable Entity in JPA

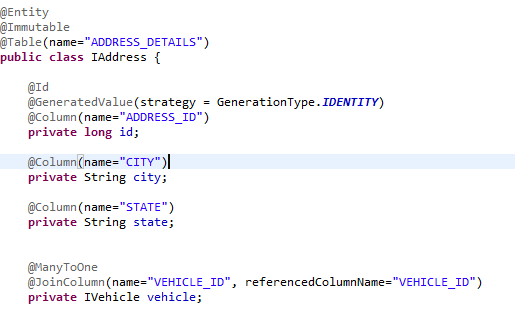
* Projection is a subset of entities properties
* It is used when we need only limited properties in an entity tree structure.
* @Immutable will not allow the entity to be modified. We can initialize an immutable entity one time but onwards it will not be modified.
* In this way entity becomes read-only.
* In This case, if we just need to load only address details for a Policy name, below would be the Projection Entity Structure



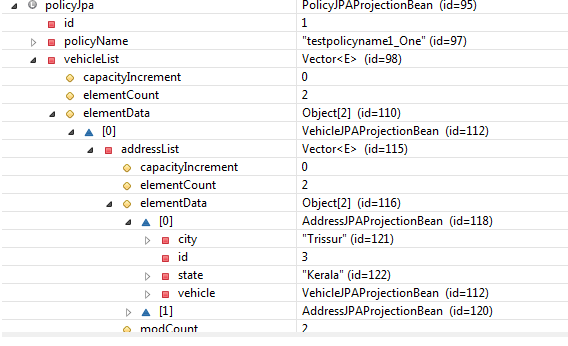
-----------------------------------------------------------------------------------------



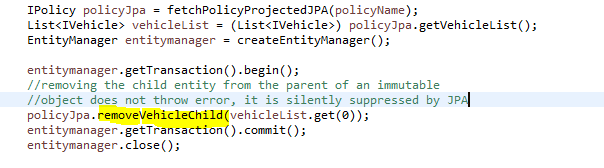
------------------------------------------------------------------------------------------



* Only the minimum number of properties defined in the new set of projected entities are loaded.

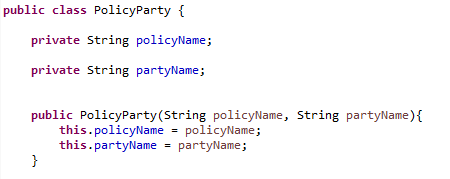


* If we try to modify, JPA will not throw error. It will silently be discarded.

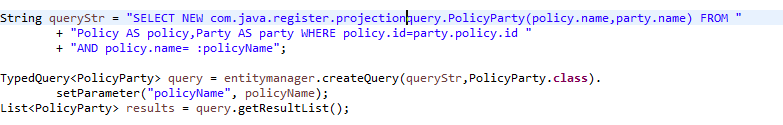


# Projection Queries

* This is done using Constructor Expressions.
* This is mainly useful for queries with multiple SELECT expressions, where custom result objects can provide an object oriented alternative to representing results as Object[] elements.
* Sample Projection DTO which holds Policy name and party name



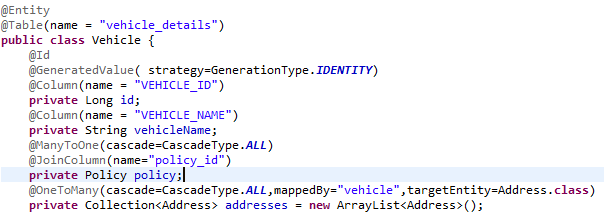
* Sample Code to retrieve the results in Object[] elements (or List<PolicyParty>)



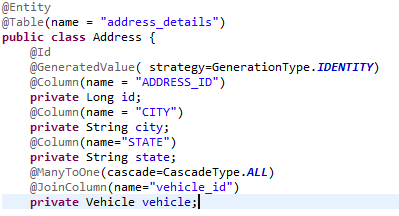
# Cascade Operations

* CascadeType attribute is used to propagate all the state changes from the parent to child entities.
* To Explain efficiently, have taken a sample One-To-Many relationship of VEHICLE and ADDRESS

Vehicle entity (parent):



Address entity (child):

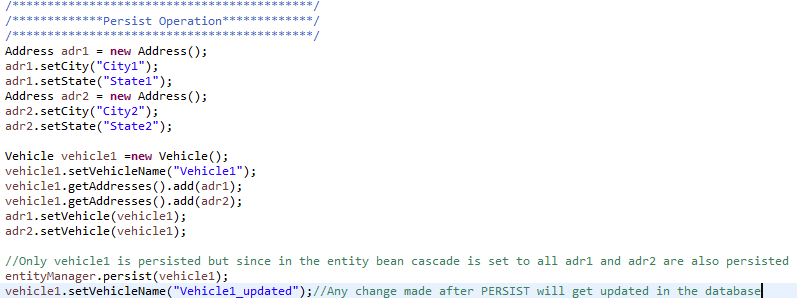


CascadeType.ALLwill cascade all the below operations to the child entities.

**Cascade Types:**

## PERSIST

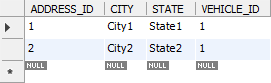
* Persist takes an entity instance, adds it to the context and makes that instance managed (i.e future updates to the entity will be tracked).
* If CascadeType.PERSIST is used then it will cascade only the persist operations



Vehicle Table:

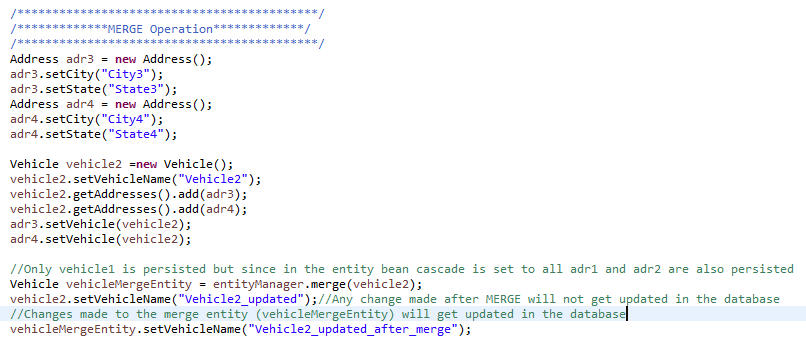


Address Table:



## MERGE

* Merge creates a new instance of your entity, copies the state from the supplied entity and any change made to the supplied entity after calling MERGE is not updated in database, contrary to PERSIST. But changes made to the new instance of your entity is updated in the database.
* If CascadeType.MERGE is used only the merge operations are cascaded.



Vehicle Table:



Address Table:

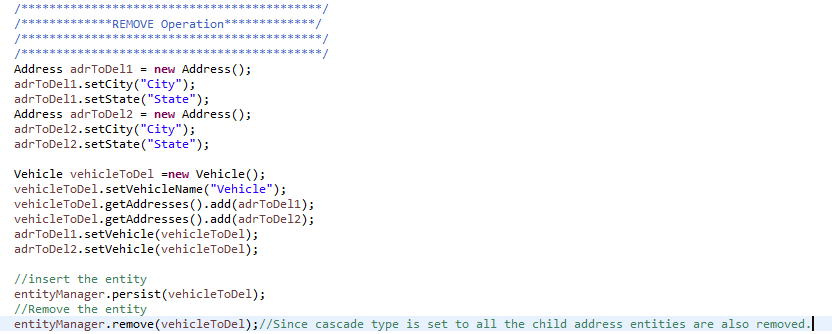


## Refresh

* Refresh operation refreshes the entity with the latest values from the database.
* CascadeType.REFRESH will refresh the child entities also when the parent is refreshed.

## Remove

* Remove is used to delete an entity from the database.
* CascadeType.REMOVE will make sure that if a parent entity is deleted then all of its child entities are also deleted.



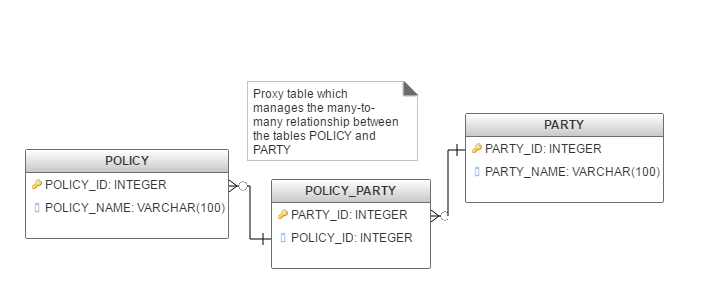
**Note:**

* The behaviour of CASCADE operations differs from multiple JPA providers
* The above example is done using hibernate JPA provider

# Many-to- Many Relationship

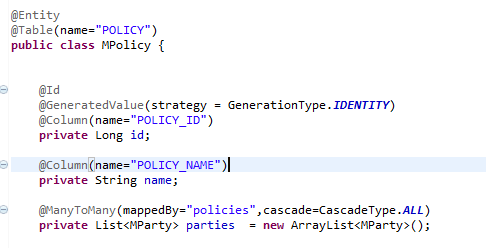
* Many-to-many relationship introduced between PARTY and POLICY.

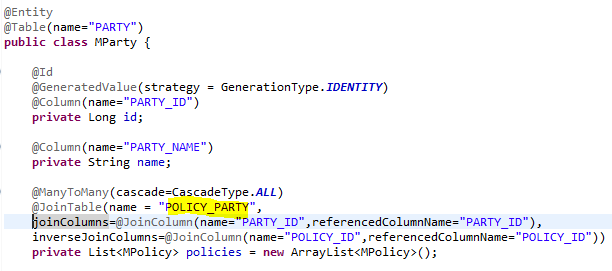
## Database Structure



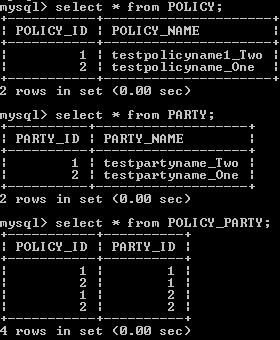
## Entity Relationship

* Below is the Entity relationship(have added prefix M to the classes since Entity names should be always unique in an application)





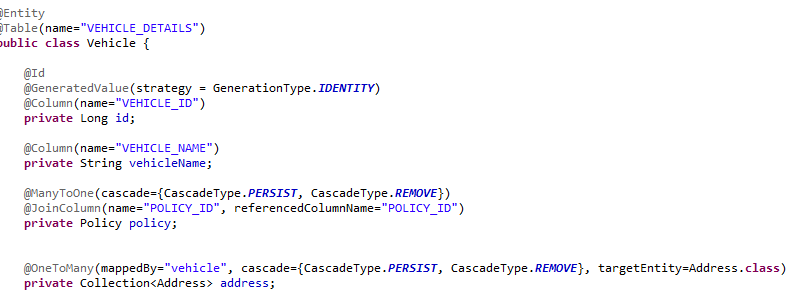
* Database With Sample Data

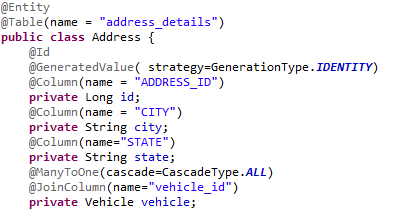


# Join Fetch

* JPA’s JOIN FETCH is useful in cases when you want to eagerly load some lazy loaded collection of your entity.
* It’s an alternative that can be used instead of Eager Fetch type.
* By default all @OneToMany and @ManyToMany relationships are **lazy loaded**.

Consider a OneToMany relationship from vehicle to address entity (one vehicle can have multiple address entry)





**WITHOUT USING JOIN FETCH :**



QUERY TRIGGERED:

select vd.VEHICLE\_ID, vd.policy\_id, vd.VEHICLE\_NAME from vehicle\_details vd

Note :

* Initially only limited amount of data are loaded from the database.
* The address details are not loaded (since its lazy loading by default). But they are loaded only when explicitly called using the address getter methods.

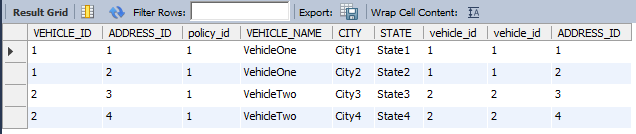
**USING JOIN FETCH :**



QUERY TRIGGERED:

select vd.VEHICLE\_ID, adr.ADDRESS\_ID, vd.policy\_id, vd.VEHICLE\_NAME, adr.CITY, adr.STATE, adr.vehicle\_id, adr.vehicle\_id, adr.ADDRESS\_ID from vehicle\_details vd inner join address\_details adr on vd.VEHICLE\_ID=adr.vehicle\_id

Note :

* The address details are loaded in the initial query itself.
* Join Fetch has duplicate result set problem (the above inner join query will output 2 vehicle record row since each vehicle has two addresses associated with it.).

**Sample Projects**

1. Using Eclipselink JPA Service provider

<https://github.com/kumardasanand28/kanand/tree/master/SampleJPAJMS>

1. Using Hibernate JPA Service provider