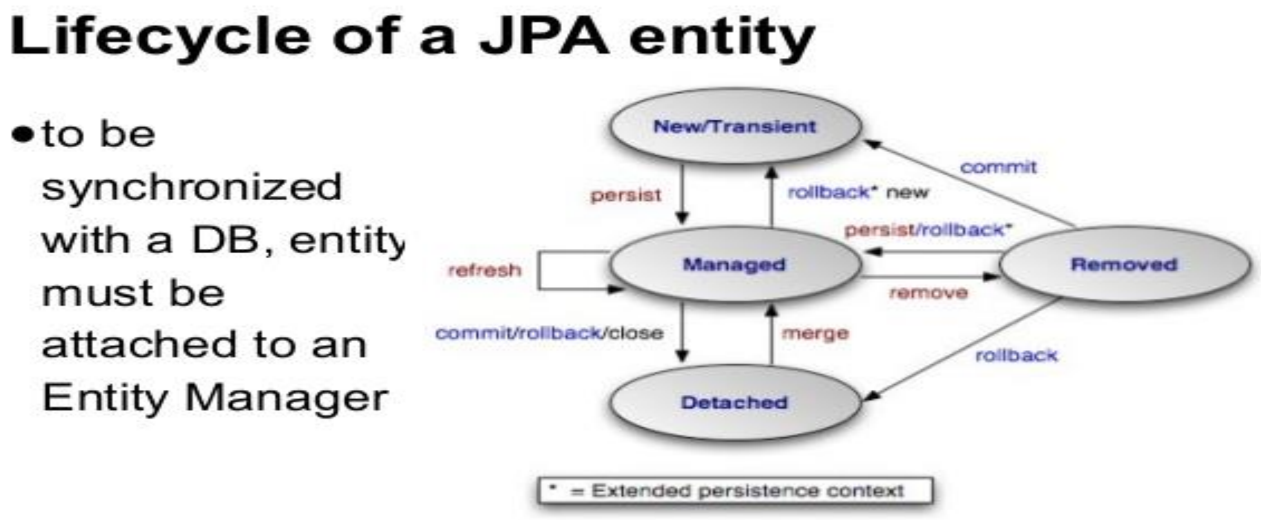
Hibernate – Entity Lifecycle

Entity Lifecycle States

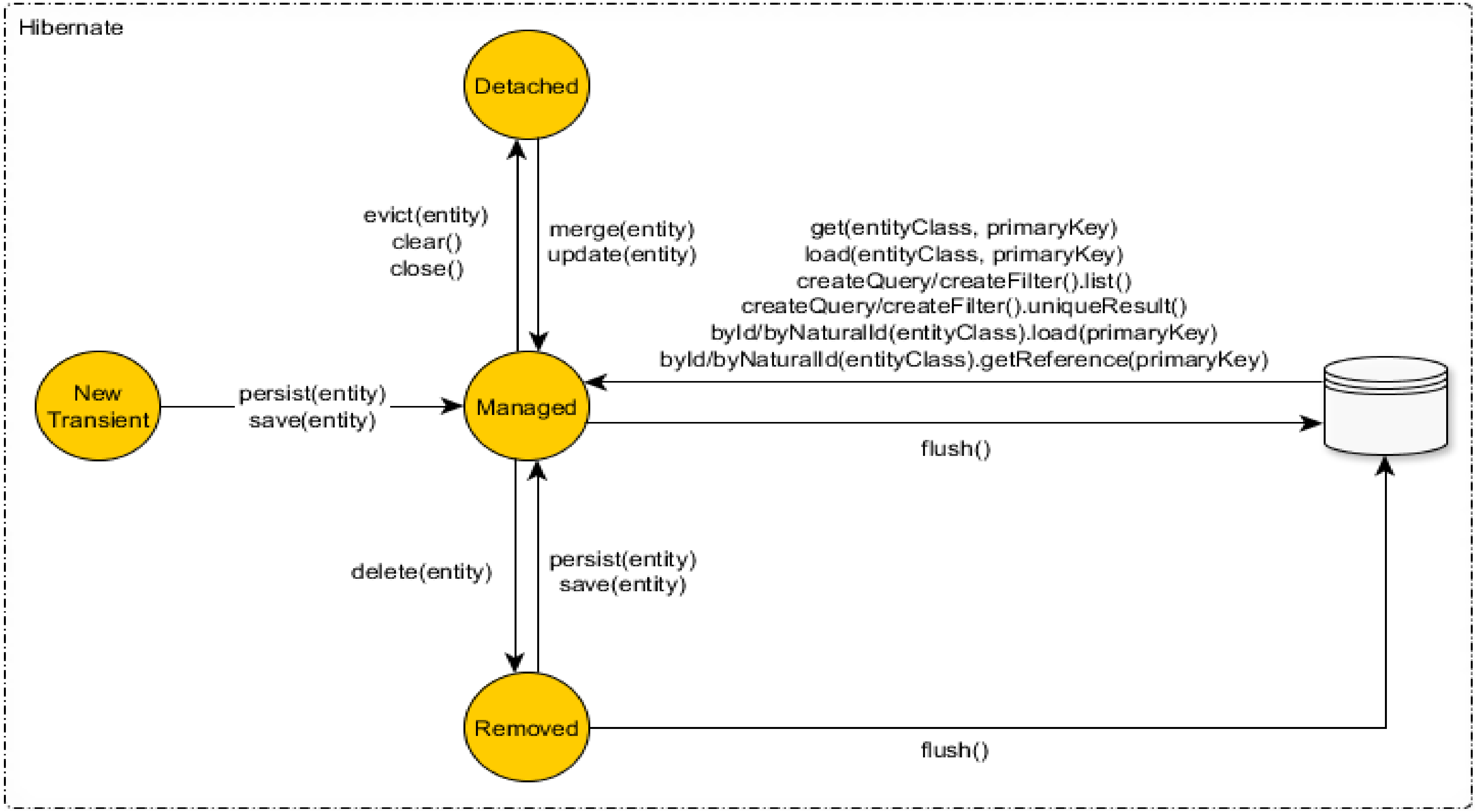
Given an instance of a class that is mapped to Hibernate, it can be in any one of **four different persistence states** (known as **hibernate entity lifecycle states**):

1. Transient
2. Persistent
3. Detached
4. Removed



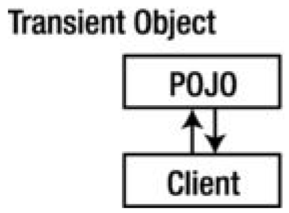
Lifecycle of a JPA entity

- to be synchronized with a DB, entity must be attached to an Entity Manager


Hibernate
Detached
evict(entity)
clear()
close()
merge(entity)
update(entity)
get(entityClass, primaryKey)
load(entityClass, primaryKey)
createQuery/createFilter().list()
createQuery/createFilter().uniqueResult()
byId/byNaturalId(entityClass).load(primaryKey)
byId/byNaturalId(entityClass).getReference(primaryKey)
New
Transient
persist(entity)
save(entity)
Managed
flush()
delete(entity)
persist(entity)
save(entity)
Removed
flush()

1.1. Transient

Transient entities exist in heap memory as normal Java objects. Hibernate does not manage transient entities or persist changes done on them.

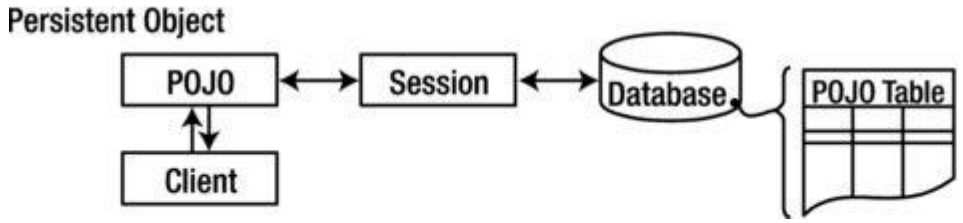


Transient objects are independent of Hibernate

To persist the changes to a transient entity, we would have to ask the hibernate session to save the transient object to the database, at which point Hibernate assigns the object an identifier and marks the object as being in persistent state.

1.2. Persistent

Persistent entities exist in the database, and Hibernate manages the persistence for persistent objects.

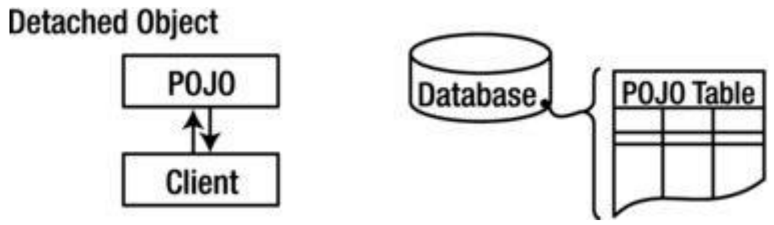


Persistent objects are maintained by Hibernate

If fields or properties change on a persistent object, hibernate will keep the database representation up to date when the application marks the changes as to be committed.

1.3. Detached

Detached entities have a representation in the database, but changes to the entity will not be reflected in the database, and vice-versa. This temporary separation of the entity and the database is shown in the image below.



Detached objects exist in the database but are not maintained by Hibernate

A detached entity can be created by closing the session that it was associated with, or by evicting it from the session with a call to the session's `evict()` method.

NOTE →One reason you might consider doing this would be to read an object out of the database, modify the properties of the object in memory, and then store the results some place other than your database. This would be an alternative to doing a deep copy of the object.

In order to persist changes made to a detached object, the application must re-attach it to a valid Hibernate session. A detached instance can be associated with a new Hibernate session when your application calls one of the `persist()`, `refresh()`, `merge()`, `update()`, or `save()` methods on the new session with a reference to the detached object.

After the method call, the detached entity would be a persistent entity managed by the new Hibernate session.

1.4. Removed

Removed entities are objects that are being managed by Hibernate (persistent entities, in other words) that have been passed to the session's `remove()` method.

When the application marks the changes held in the session as to be committed, the entries in the database that correspond to removed entities are deleted.

Points to remember

1. Newly created POJO object will be in the transient state. Transient entity doesn't represent any row of the database i.e. not associated with any session object. It's plain simple java object.
2. Persistent entity represents one row of the database and always associated with some unique hibernate session. Changes to persistent objects are tracked by hibernate and are saved into database when commit call happen.
3. Detached entities are those who were once persistent in the past, and now they are no longer persistent. To persist changes done in detached objects, you must re-attach them to hibernate session.
4. Removed entities are persistent objects that have been passed to the session's `remove()` method and soon will be deleted as soon as changes held in the session will be committed to database.