**Everything about hibernate**

Hibernate Framework

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

Hibernate uses [**JDBC**](http://www.javaguides.net/p/jdbc-tutorial.html) for all database communications. Hibernate uses [**JDBC**](http://www.javaguides.net/p/jdbc-tutorial.html) to interact with the database.

## ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

## What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

## Advantages of Hibernate Framework

Hibernate framework is open source

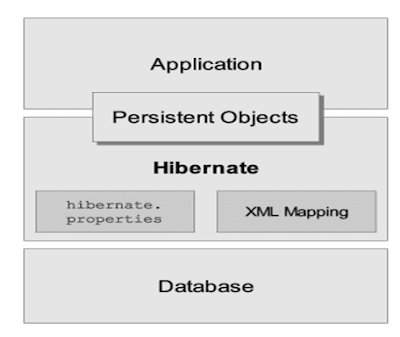
The performance of hibernate framework is fast

HQL (Hibernate Query Language) is the object-oriented version of SQL.

Automatic Table Creation

# **Hibernate Architecture**

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.



## Elements of Hibernate Architecture

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider.

Session

The session object provides an interface between the application and data stored in the database.

The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes.

[**beginTransaction**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html#beginTransaction())()  
          Begin a unit of work and return the associated Transaction object.

[**cancelQuery**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html#cancelQuery())()  
          Cancel the execution of the current query.

[**clear**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html#clear())()  
          Completely clear the session.

Transaction

The transaction object specifies the atomic unit of work.

We generally group the related work within one transaction so that if part of the work is failed then entire transaction should be failed.

**Transaction has 4 main properties (ACID)**

A transaction is associated with a Session and is usually instantiated by a call to Session.beginTransaction().

[**begin**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Transaction.html#begin())()  
          Begin a new transaction.

[**commit**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Transaction.html#commit())()  
          Flush the associated Session and end the unit of work (unless we are in [FlushMode.MANUAL](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/FlushMode.html" \l "MANUAL).

[**isActive**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Transaction.html#isActive())()  
          Is this transaction still active?

Again, this only returns information in relation to the local transaction, not the actual underlying transaction.

[**rollback**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Transaction.html#rollback())()  
          Force the underlying transaction to roll back.

[**wasCommitted**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Transaction.html#wasCommitted())()  
          Check if this transaction was successfully committed.

[**wasRolledBack**](https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Transaction.html#wasRolledBack())()  
          Was this transaction rolled back or set to rollback only?

This only accounts for actions initiated from this local transaction.

**How does Spring manage Hibernate Sessions?**

By default, Spring Boot applies transaction management at the repository level. In this case, when calling a JpaRepository method (or in general any Repository method), Spring will:

* Ask the SessionFactory to create a new session
* Open this session
* Open a transaction
* Perform the called Repository method
* Close the transaction
* Close the session

However, if you apply @Transactional to the service class or method, Spring will open the session and the transaction on entry to the service method, and the repository method will be performed within the existing transaction.

## Caching in Hibernate

cache is a memory buffer present as a link between application and database layer to store recently used data

This saves time as we application do not have to go and hit the database server to fetch the data from tables as it is readily present in cache.

**1). First level cache**

The First level cache is by default enabled by Hibernate itself. The session object maintains the first-level cache.  
An application can have many sessions. Data hold by one session object **isn't** accessible to **the whole** application — means **the info** of **a specific** session **isn't** shared with other sessions of **the appliance**. So **you'll** use the first-level cache to store local data i.e. required by the session itself.

let’s make it simple to grasp, As we all know that hibernate is an ORM tool that's wont to simplify DB operations. It “converts objects to relations (to store into DB) and vice-versa”.  
So after you query an entity or object, for the very first time it's retrieved from the database and stored into the first-level cache (associated with the hibernate session). If we query for the identical entity or object again with the identical session object, it'll be loaded from cache and no SQL query are executed. Take a glance at the below code snippet

In the above example hibernate will fire query only one time to the Database. From the second time onwards it'll return only from the session object.

## Some useful methods:

* **Session.evict():** to remove the cached/stored entity.
* **refresh():** method to refresh the cache.
* **clear():** method to remove all the entities from the cache.

# **Hibernate Example using Annotation in Eclipse**

The hibernate application can be created with annotation. There are many annotations that can be used to create hibernate application such as @Entity, @Id, @Table etc.

Hibernate Annotations are based on the JPA 2 specification and supports all the features.

All the JPA annotations are defined in the **javax.persistence** package. Hibernate EntityManager implements the interfaces and life cycle defined by the JPA specification.

The core advantage of using hibernate annotation is that you don't need to create mapping (hbm) file. Here, hibernate annotations are used to provide the meta data.

**Maven Dependencies**

If we open up *pom.xml*, we’ll see that we have *spring-boot-starter-web* and *spring-boot-starter-test* as maven dependencies. As their names suggest, these are the starting dependencies in Spring Boot.

Let’s have a quick look at the dependency that pulls in JPA:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-data-jpa</**artifactId**>

</**dependency**>Copy

This dependency includes JPA API, JPA Implementation, JDBC, and the other necessary libraries. Since the default JPA implementation is Hibernate, this dependency is actually enough to bring it in as well.