**23-11-2022**

**Hibernate**

# What is ORM?

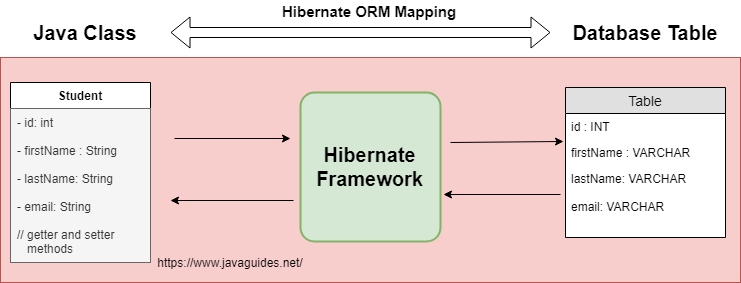
*Object-relational mapping* or *ORM* is the programming technique to map application domain model objects to the relational database tables. Hibernate is a Java-based ORM tool that provides a framework for mapping application domain objects to the relational database tables and vice versa.

# What is Hibernate Framework?

Hibernate is a java based ORM tool that provides a framework for mapping application domain objects to the relational database tables and vice versa.

Hibernate is probably the most popular JPA implementation and one of the most popular Java frameworks in general. Hibernate acts as an additional layer on top of JDBC and enables you to implement a database-independent persistence layer. It provides an object-relational mapping implementation that maps your database records to Java objects and generates the required SQL statements to replicate all operations to the database.

**Example**: Below diagram shows an *Object Relational Mapping* between **Student** Java class and **student** table in the database.

**[](https://2.bp.blogspot.com/-SKU7VOqnoDQ/XJixZ6wkuGI/AAAAAAAAFyc/IAJ3V0VEliMdtBNwfAI0_hcS4beQpxGrQCLcBGAs/s1600/hibernate-orm-mapping.png)**

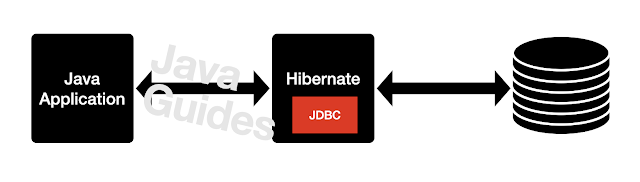
# What is the Java Persistence API (JPA)?

*Java Persistence API (JPA)* provides a specification for managing the relational data in applications.

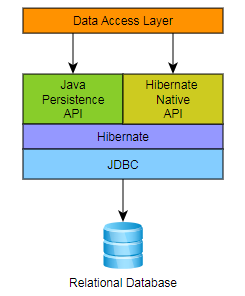
JPA specifications are defined with annotations in a *javax.persistence* package. Using JPA annotation helps us in writing implementation-independent code.

# How does Hibernate relate to JDBC?

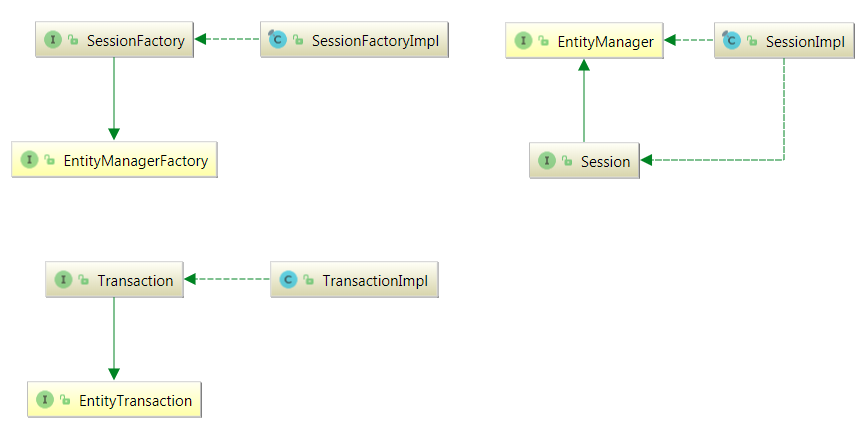
Hibernate uses [**JDBC**](http://www.javaguides.net/p/jdbc-tutorial.html) for all database communications. Hibernate uses [**JDBC**](http://www.javaguides.net/p/jdbc-tutorial.html) to interact with the database.  
  
Hibernate acts as an additional layer on top of JDBC and enables you to implement a database-independent persistence layer:

**[](https://lh3.googleusercontent.com/-Lv3e9V6kjGY/YSsXHjfVWjI/AAAAAAAAJCk/4WGSKW-TengUaV5_-7_BL4jydsM7_srzACLcBGAsYHQ/hibernate%2Barchitecture%2Bflow.png)**

# Architecture

**[](https://4.bp.blogspot.com/-oS8h3K3NVVg/W_0yF7WXm0I/AAAAAAAAE_Y/4ROZk3akTmgJt3N42oQUoMtHfvs9odAyQCLcBGAs/s1600/architecture.PNG)**

Hibernate, as an ORM solution, effectively "sits between" the Java application data access layer and the Relational Database, as can be seen in the diagram above. The Java application makes use of the Hibernate APIs to load, store, query, etc its domain data. Here we will introduce the essential Hibernate APIs.



As a JPA provider, [**Hibernate**](http://www.javaguides.net/p/hibernate-tutorial.html) implements the *Java Persistence API* specifications and the association between JPA interfaces and Hibernate specific implementations can be visualized in the following diagram:

## SessionFactory (org.hibernate.SessionFactory)

A thread-safe (and immutable) representation of the mapping of the application domain model to a database. Acts as a factory for *org.hibernate.Session* instances. The *EntityManagerFactory* is the JPA equivalent of a *SessionFactory* and basically, those two converge into the same *SessionFactory* implementation.

A *SessionFactory* is very expensive to create, so, for any given database, the application should have only one associated SessionFactory. The *SessionFactory* maintains services that Hibernate uses across all *Session*(s) such as second-level caches, connection pools, transaction system integrations, etc.

## Session (org.hibernate.Session)

A single-threaded, short-lived object conceptually modeling a "Unit of Work". In JPA nomenclature, the *Session* is represented by an *EntityManager*.

Behind the scenes, the Hibernate Session wraps a JDBC *java.sql.Connection* and acts as a factory for *org.hibernate.Transaction* instances. It maintains a generally "repeatable read" persistence context (first level cache) of the application domain model.

## Transaction (org.hibernate.Transaction)

A single-threaded, short-lived object is used by the application to demarcate individual physical transaction boundaries. *EntityTransaction* is the JPA equivalent and both acts as an abstraction API to isolate the application from the underlying transaction system in use (JDBC or JTA).

# What are the important benefits of using the Hibernate Framework?

Some of the important benefits of using [**hibernate framework**](http://www.javaguides.net/p/hibernate-tutorial.html) are:

* Hibernate eliminates all the boiler-plate code that comes with [**JDBC**](http://www.javaguides.net/p/jdbc-tutorial.html) and takes care of managing resources, so we can focus on business logic.
* [**Hibernate framework**](http://www.javaguides.net/p/hibernate-tutorial.html) provides support for XML as well as JPA annotations, which makes our code implementation independent.
* Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, pol, morphism, and association.
* Hibernate is an open-source project from Red Hat Community and is used worldwide. This makes it a better choice than others because the learning curve is small and there are tons of online documentation and help is easily available in forums.
* Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.
* Hibernate supports lazy initialization using proxy objects and performs actual database queries only when it’s required.
* Hibernate cache helps us in getting better performance.
* For a database vendor-specific feature, hibernate is suitable because we can also execute native SQL queries.
* Overall hibernate is the best choice in the current market for the ORM tool, it contains all the features that you will ever need in an ORM tool.