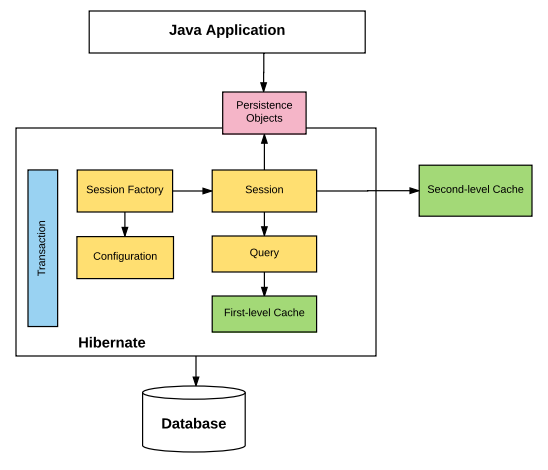
**Hibernate Tutorial**

Hibernate is an open source **Java persistence** framework project. It performs powerful object-relational mapping and query databases using HQL and SQL. Hibernate is a great tool for **ORM mappings** in Java. It can cut down a lot of complexity and thus defects as well from your application, which may otherwise find a way to exist. This is specially boon for developers with limited knowledge of SQL.

Initially started as an ORM framework, Hibernate has spun off into many projects, such as *Hibernate Search*, *Hibernate Validator*, *Hibernate OGM* (for NoSQL databases), and so on.

**Hibernate Architecture**

The following diagram summarizes the main building blocks in hibernate architecture.

Hibernate Architecture

Let’s understand what each block represents.

1. **Configuration** : Generally written in hibernate.properties or hibernate.cfg.xml files. For Java configuration, you may find class annotated with @Configuration. It is used by Session Factory to work with Java Application and the Database. It represents an entire set of mappings of an application Java Types to an SQL database.
2. **Session Factory** : Any user application requests Session Factory for a session object. Session Factory uses configuration information from above listed files, to instantiates the session object appropriately.
3. **Session** : This represents the interaction between the application and the database at any point of time. This is represented by the org.hibernate.Session class. The instance of a session can be retrieved from the SessionFactory bean.
4. **Query** : It allows applications to query the database for one or more stored objects. Hibernate provides different techniques to query database, including NamedQuery and Criteria API.
5. **First-level cache** : It represents the default cache used by Hibernate Session object while interacting with the database. It is also called as session cache and caches objects within the current session. All requests from the Session object to the database must pass through the first-level cache or session cache. One must note that the first-level cache is available with the session object until the Session object is live.
6. **Transaction** : enables you to achieve data consistency, and rollback incase something goes unexpected.
7. **Persistent objects** : These are plain old Java objects (POJOs), which get persisted as one of the rows in the related table in the database by hibernate.They can be configured in configurations files (hibernate.cfg.xml or hibernate.properties) or annotated with @Entity annotation.
8. **Second-level cache** : It is used to store objects across sessions. This needs to be explicitly enabled and one would be required to provide the cache provider for a second-level cache. One of the common second-level cache providers is *EhCache*.

**Salient features of the Hibernate framework**

* **Object/Relational Mapping**

Hibernate, as an ORM framework, allows the mapping of the Java domain object with database tables and vice versa. As a result, business logic is able to access and manipulate database entities via Java objects. It helps to speed up the overall development process by taking care of aspects such as transaction management, automatic primary key generation, managing database connections and related implementations, and so on.

* **JPA provider**

Hibernate does support the [**Java Persistence API**](https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html)**(JPA)** specification. JPA is a set of specifications for accessing, persisting, and managing data between Java objects and relational database entities.

* **Idiomatic persistence**

Any class that follows [object-oriented principles](https://howtodoinjava.com/oops/object-oriented-principles/) such as inheritance, polymorphism, and so on, can be used as a persistent class.

* **High performance and scalability**

Hibernate supports techniques such as different fetching strategies, lazy initialization, optimistic locking, and so on, to achieve high performance, and it scales well in any environment.

* **Easy to maintain**

Hibernate is easier to maintain as it requires no special database tables or fields. It generates SQL at system initialization time. It is much quicker and easier to maintain compared to JDBC.

In this page, I have categorized all available **hibernate examples** in this blog. This page will be updated every time, a new hibernate tutorial is published in this blog. Stay Tuned !!

Feel free to suggest topics you want to read more on.

**Hello world application**

[**Hibernate 3 introduction and writing hello world application**](https://howtodoinjava.com/hibernate/hibernate-3-introduction-and-writing-hello-world-application/)

In this post, I will try to detail out more information on hibernate and then will identify the basic steps to use hibernate for our first running **java hibernate example** application.

**Basic concepts**

[**How to build SessionFactory in hibernate 4**](https://howtodoinjava.com/hibernate/hibarnate-4-how-to-build-sessionfactory/)

If you have been watching previous hibernate releases then you must have noticed that they have deprecated a lot of classes in quick succession. Deprecated classes are AnnotationConfiguration, ServiceRegistryBuilder and so on.

In this hibernate tutorial, I am giving an example of building hibernate SessionFactory without using deprecated classes mentioned above. I am using the **latest hibernate version** i.e. Hibernate 4.3.6.Final, so you can make sure that you are using the latest approach for building session factory.

[**Entities Equality and Identity Concepts**](https://howtodoinjava.com/hibernate/hibernate-entities-equality-and-identity/)

Many times in our application, we face a situation where we have to compare two objects to check their equality for satisfying some business rules. In core java, we have already much knowledge about checking equality of objects, but in hibernate, we need to take care of a few extra things as well. Let’s learn what are those extra concepts.

[**Defining Association Mappings between Hibernate Entities**](https://howtodoinjava.com/hibernate/how-to-define-association-mappings-between-hibernate-entities/)

When we annotate the java classes with JPA annotations and make them persistent entities, we can face situations where two entities can be related and must be referenced from each other, in either uni-direction or in bi-direction. Let’s understand a few basic things before actually creating references between hibernate entities.

[**Entity / Persistence LifeCycle States Concepts**](https://howtodoinjava.com/hibernate/hibernate-entity-persistence-lifecycle-states/)

Given an instance of an object that is mapped to Hibernate, it can be in any one of four different states: transient, persistent, detached, or removed. We are going to learn about them today in this hibernate tutorial.

[**Using In-memory Database With Hibernate**](https://howtodoinjava.com/hibernate/hibernate-4-using-in-memory-database-with-hibernate/)

In this example, I am using HSQLDB Database for creating and accessing in-memory database through our hibernate code.

[**Hibernate JPA Cascade Types**](https://howtodoinjava.com/hibernate/hibernate-jpa-cascade-types/)

To enable **cascade and inverse** effect, we had used “CascadeType” attribute in entities. In this tutorial, we will learn about various type of available options for cascading via CascadeType.

[**Pros and Cons of Hibernate Annotations Vs Mappings**](https://howtodoinjava.com/hibernate/pros-and-cons-of-hibernate-annotations-vs-mappings/)

As you may know that prior to the inline annotations, the only way to create hibernate mappings was through XML files. Although various tools from Hibernate and third-party projects allowed part or all of these mappings to be generated from Java source code automatically. Today annotations are the newest way to define mappings but it is not automatically the best way to do so. Let’s discuss the drawbacks and benefits of hibernate (or I should say JPA) annotations before discussing when to apply them.

[**Hibernate Query Language [HQL]**](https://howtodoinjava.com/hibernate/complete-hibernate-query-language-hql-tutorial/)

HQL is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. It is a superset of the JPQL, the Java Persistence Query Language; a JPQL query is a valid HQL query, but not all HQL queries are valid JPQL queries. HQL is a language with its own syntax and grammar.

Let’s learn HQL using the following examples:

1. Basic HQL Syntax
   * Update Operation
   * Delete Operation
   * Insert Operation
   * Select Operation
2. The from Clause and Aliases
3. The select Clause and Projection
4. Using Named Parameters
5. Paging Through the Result Set
6. Obtaining a Unique Result
7. Sorting Results with the order by Clause
8. Associations
9. Aggregate Methods
10. Named Queries
11. Using Native SQL
12. Enable Logging and Commenting

[**Hibernate Criteria Queries**](https://howtodoinjava.com/hibernate/hibernate-criteria-queries-tutorial-and-examples/)

The Criteria Query API lets you build nested, structured query expressions in Java, providing a compile-time syntax checking that is not possible with a query language like HQL or SQL. The Criteria API also includes query by example (QBE) functionality. This lets you supply example objects that contain the properties you would like to retrieve instead of having to step-by-step spell out the components of the query. It also includes projection and aggregation methods, including count(). Let’s explore it’s different features in detail.

1. Basic Usage Example
2. Using Restrictions with Criteria
3. Paging Through the Result Set
4. Obtaining a Unique Result
5. Obtaining Distinct Results
6. Sorting the Query’s Results
7. Performing Associations (Joins)
8. Adding Projections
9. Query By Example (QBE)
10. Summary

[**Lazy Loading in Hibernate**](https://howtodoinjava.com/hibernate/lazy-loading-in-hibernate/)

In this this tutorial, I will be discussing a must-known feature in hibernate known as lazy loading. This is useful specially if you working in a very large application.

**CRUD Operation Examples**

[**Hello world insert data**](https://howtodoinjava.com/hibernate/hibernate-insert-query-tutorial/)

In this tutorial, I am giving example of inserting data in a single table.

[**Hibernate named query tutorial**](https://howtodoinjava.com/hibernate/hibernate-named-query-tutorial/)

Named queries in hibernate is a technique to group the HQL statements in single location, and lately refer them by some name whenever need to use them. It helps largely in code cleanup because these HQL statements are no longer scattered in whole code.

[**Loading entity from database example**](https://howtodoinjava.com/hibernate/how-to-load-get-entity-in-hibernate/)

Examples of loading an hibernate entity using either load or get method.

[**Save() and saveOrUpdate() for Saving Entities**](https://howtodoinjava.com/hibernate/save-and-saveorupdate-for-saving-hibernate-entities/)

Please not that creating an instance of a class, you mapped with a hibernate annotations, does not automatically persist the object to the database. It must be save explicitly after attaching it to a valid hibernate session. Let’s learn how to do it.

[**Merging and Refreshing Entities**](https://howtodoinjava.com/hibernate/merging-and-refreshing-hibernate-entities/)

In this tutorial, I am discussing few thoughts around refresh() and merge() method present in hibernate session class.

**Hibernate entity mappings**

[**Hibernate one to one mapping using annotations**](https://howtodoinjava.com/hibernate/hibernate-one-to-one-mapping-using-annotations/)

Let’s discuss variations of one-to-one mappings supported in hibernate:

1. Using foreign key association
2. Using a common join table
3. Using shared primary key

[**Hibernate one to many mapping using annotations**](https://howtodoinjava.com/hibernate/hibernate-one-to-many-mapping-using-annotations/)

Discuss variations of one-to-many mappings supported in hibernate:

1. Using foreign key association
2. Using a join table

[**Hibernate many to many mapping using annotations**](https://howtodoinjava.com/hibernate/hibernate-many-to-many-mapping-using-annotations/)

Discuss variations of many-to-many mappings supported in hibernate.

**Hibernate Connection Pooling and Caching**

[**C3P0 Connection Pool Configuration Tutorial**](https://howtodoinjava.com/hibernate/hibernate-c3p0-connection-pool-configuration-tutorial/)

By default, Hibernate uses JDBC connections in order to interact with a database. Creating these connections is expensive—probably the most expensive single operation Hibernate will execute in a typical-use case. Since JDBC connection management is so expensive that possibly you will advise to use a pool of connections, which can open connections ahead of time (and close them only when needed, as opposed to “when they’re no longer used”).

C3P0 is an example of an external connection pool. In this tutorial, we will learn to use it with hibernate.

[**Hibernate EhCache configuration**](https://howtodoinjava.com/hibernate/hibernate-ehcache-configuration-tutorial/)

Caching is facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce the number of queries made to the database in a single transaction. Hibernate also provide this caching functionality at first level and second level.

In this tutorial, I am giving an example using ehcache configuration as second level cache in hibernate.

[**Hibernate first level cache with example**](https://howtodoinjava.com/hibernate/understanding-hibernate-first-level-cache-with-example/)

Fist level cache in hibernate is enabled by default and you do not need to do anything to get this functionality working. Let’s learn more about it.

[**Hibernate second level cache with example**](https://howtodoinjava.com/hibernate/how-hibernate-second-level-cache-works/)

In this tutorial, I am giving concepts around hibernate second level cache and give example using code snippets.

**Hibernate best practices**

[**Hibernate @NaturalId usage and example**](https://howtodoinjava.com/hibernate/hibernate-naturalid-example-tutorial/)

Hibernate 4 has bring lots of improvements and @NaturalId is one of such nice improvements. As you know @Id annotation is used as meta data for specifying the primary key of an entity. But sometimes, entity is usually used in DAO layer code with id which not not primary key but its logical or natural id. In such cases, @NaturalId annotation will prove good replacement of named queries in hibernate.

[**Get entity reference for lazy loading**](https://howtodoinjava.com/hibernate/hibernate-4-example-to-get-entity-reference-for-lazy-loading/)

Lazy loading is a design pattern commonly used in computer programming to defer initialization of an object until the point at which it is needed. Hibernate lazy loading can be done by specifying “fetch= FetchType.LAZY” in hibernate mapping annotations.

**Integration with other frameworks**

[**Integrate Hibernate with Spring framework**](https://howtodoinjava.com/spring/spring-orm/spring-3-and-hibernate-integration-tutorial-with-example/)

This hibernate tutorial is focused on usage of Hibernate with Spring 3 framework. I will show that how a basic end to end application flow looks like as a result of this integration.

s