**Wiki - 305 - JDBC, ORM, and Hibernate**

# What is Hibernate?

Hibernate is an open source object relational mapping ([ORM](https://theserverside.techtarget.com/definition/object-relational-mapping)) tool that provides a framework to map object-oriented domain models to relational databases for web applications.

Object relational mapping is based on the containerization of objects and the abstraction that provides that capacity. Abstraction makes it possible to address, access and manipulate objects without having to consider how they are related to their data sources.

The Hibernate ORM framework guides [mapping Java classes](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/JPA-and-Hibernate-enum-mapping-with-annotations-or-the-hbmxml-file) to database tables and Java data types to SQL data types and provides [querying and retrieval](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/The-JPA-and-Hibernate-CRUD-operations-example).

# Object/Relational Mapping

Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC). Unfamiliar with the notion of ORM? [Read here](https://hibernate.org/orm/what-is-an-orm/).

# JPA Provider

In addition to its own "native" API, Hibernate is also an implementation of the Java Persistence API (JPA) specification. As such, it can be easily used in any environment supporting JPA including Java SE applications, Java EE application servers, Enterprise OSGi containers, etc.

# Session.merge() method:

The merge() method is used to merge an object with a persistent object on the basis of the same identifier. The object, as an argument is not changed, and the method returns persistent object:

1. The state of the object passed as an argument is copied to the object in the Hibernate session with the same identifier, and returns the persistent object.

2. If the object is not already present in the session with the same identifier as passed in the argument, the session will first load the object for the identifier of the object passed as an argument, and then merge it and return the persistent object of the session.

3. If the persistent object for the identifier of the object passed as an argument is not already in session and database, then a copy of object passed as an argument is persisted and is returned.

*Visit the link below for more information:* <https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html>

# Hibernate - Query Language

# 

Hibernate Query Language (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. HQL queries are translated by Hibernate into conventional SQL queries

# Named Queries

Named queries are created via class-level annotations on entities; normally, the queries apply to the entity in whose source file they occur, but there’s no absolute requirement for this to be true.

Named queries are created with the @NamedQueries annotation, which contains an array of @NamedQuery sets; each has a query and a name.

An example of named queries may look like this:

| @NamedQueries({  @NamedQuery(name = "supplier.findAll", query = "from Supplier s"),  @NamedQuery(name = "supplier.findByName", query = "from Supplier s where s.name=:name"),  }) |
| --- |

Executing the above named query is even simpler.

| Query query = session.getNamedQuery("supplier.findAll");  List<Supplier> suppliers = query.list(); |
| --- |

# 

# Enabling Logs and Comments

Hibernate can output the underlying SQL behind your HQL queries into your application’s log file. This is especially useful if the HQL query does not give the results you expect, or if the query takes longer than you wanted. This is not a feature you will have to use frequently, but it is useful should you have to turn to your database administrators for help in tuning your Hibernate application.

## HQL Logs

The easiest way to see the SQL for a Hibernate HQL query is to enable SQL output in the logs with the “*show\_sql*” property. Set this property to true in your hibernate.cfg.xml configuration file, and Hibernate will output the SQL into the logs. When you look in your application’s output for the Hibernate SQL statements, they will be prefixed with “Hibernate:”.

If you turn your log4j logging up to debug for the Hibernate classes, you will see SQL statements in your log files, along with lots of information about how Hibernate parsed your HQL query and translated it into SQL.

## HQL Comments

Tracing your HQL statements through to the generated SQL can be difficult, so Hibernate provides a commenting facility on the Query object that lets you apply a comment to a specific query. The Query interface has a setComment() method that takes a String object as an argument, as follows:

public Query setComment(String comment)

Hibernate will not add comments to your SQL statements without some additional configuration, even if you use the setComment() method. You will also need to set a Hibernate property, hibernate.use\_sql\_comments, to true in your Hibernate configuration.

If you set this property but do not set a comment on the query programmatically, Hibernate will include the HQL used to generate the SQL call in the comment. I find this to be very useful for debugging HQL.

Use commenting to identify the SQL output in your application’s logs if SQL logging is enabled.

# What does the session.lock() method in Hibernate do?

session.lock() method is used to reattach a detached object to the session. session.lock() method does not check for any data synchronization between the database and the object in the persistence context and hence this reattachment might lead to loss of data synchronization

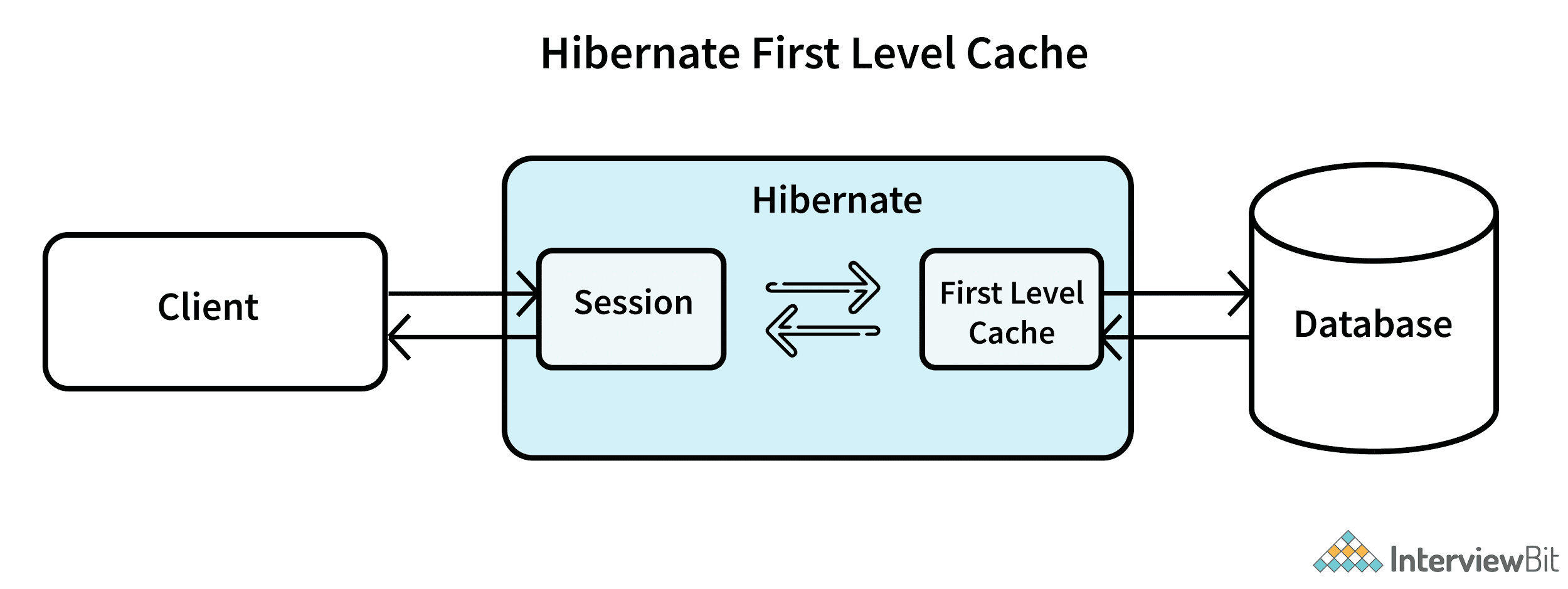
# What is hibernate caching?

Hibernate caching is the strategy for improving the application performance by pooling objects in the cache so that the queries are executed faster. Hibernate caching is particularly useful when fetching the same data that is executed multiple times. Rather than hitting the database, we can just access the data from the cache. This results in reduced throughput time of the application.

# Types of Hibernate Caching

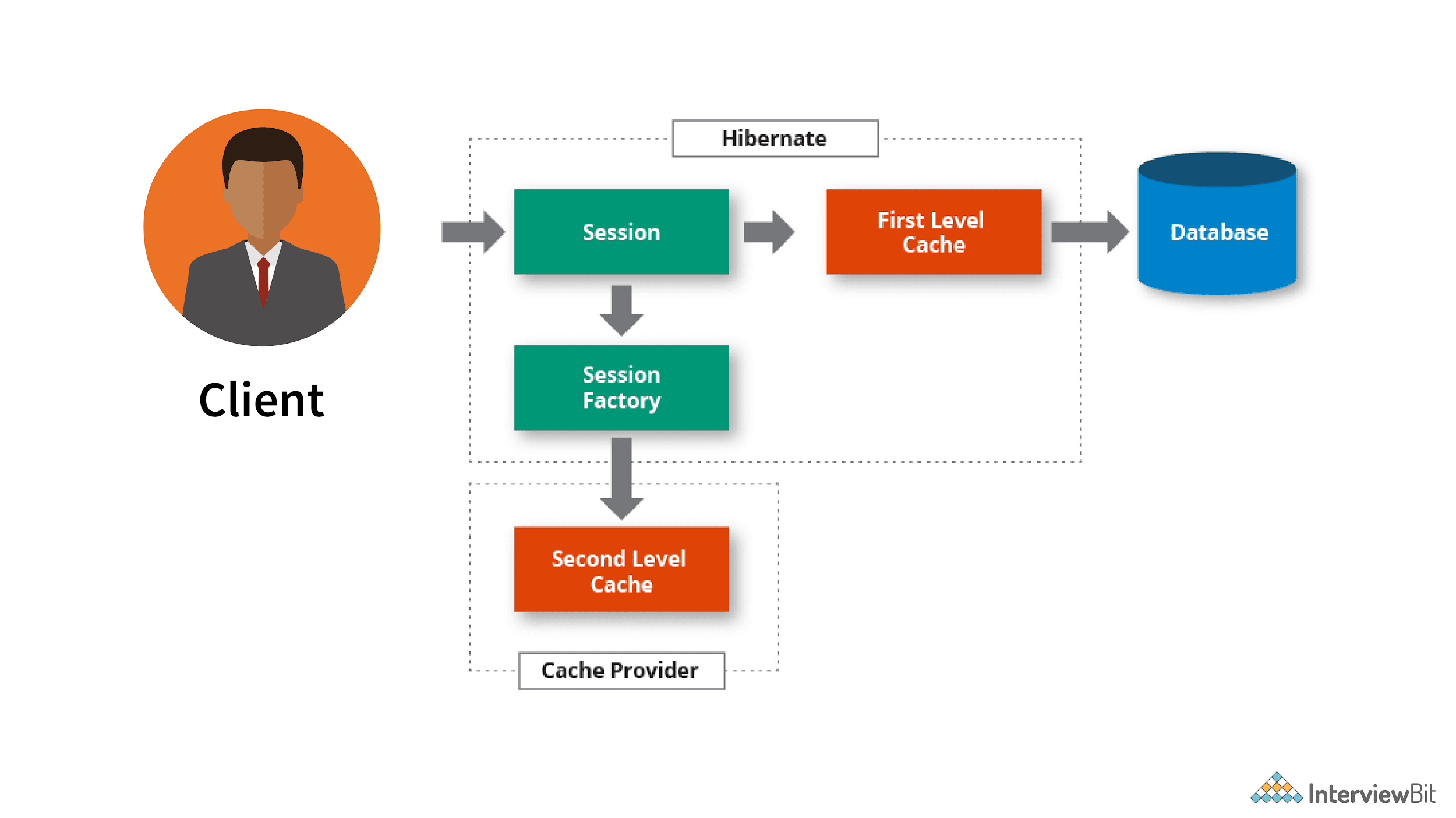
## First Level Cache:

* This level is enabled by default.
* The first level cache resides in the hibernate session object.
* Since it belongs to the session object, the scope of the data stored here will not be available to the entire application as an application can make use of multiple session objects.



## Second Level Cache:

* Second level cache resides in the SessionFactory object and due to this, the data is accessible by the entire application.
* This is not available by default. It has to be enabled explicitly.
* EH (Easy Hibernate) Cache, Swarm Cache, OS Cache, JBoss Cache are some example cache providers.



## merge() method of the hibernate session useful?

Merge() method can be used for updating existing values. The specialty of this method is, once the existing values are updated, the method creates a copy from the entity object and returns it. This result object goes into the persistent context and is then tracked for any changes. The object that was initially used is not tracked.