**HIBERNATE**

* **ORM technology**

**Overview:**

Any enterprise application performs database operations by storing and retrieving large amounts of data w.r.t database. Generally, Java developers use lots of code to interact with the database using plain JDBC. Therefore, it is advisable to use Object Relational Mapping (ORM) to reduce the burden of interacting with the database. ORM forms a bridge between object models (Java program) and relational models (database program) like JDBC.

What is JDBC?

JDBC stands for Java Database Connectivity. It provides a set of Java API for accessing the relational databases from Java program. These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database.

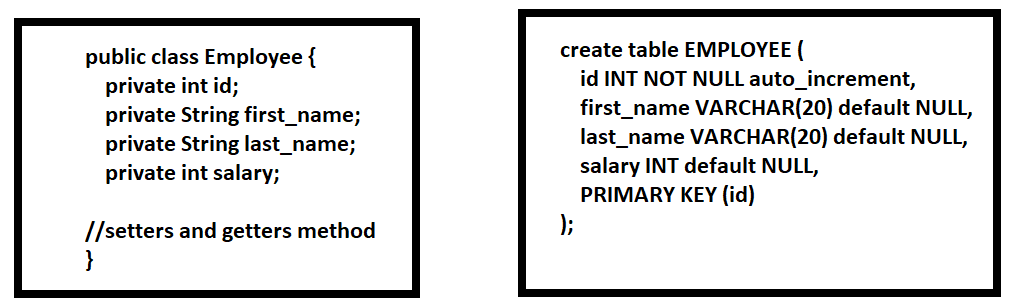
**Pros and Cons of JDBC**

|  |  |
| --- | --- |
| **PROS** | **CONS** |
| Clean and simple SQL processing. | Complex if it is used in large projects. |
| Good performance with large data. | Large programming overhead |
| Very good for small applications | No encapsulation. |
| Simple syntax so easy to learn | Hard to implement MVC concept. Query is DBMS specific. |

Why Object Relational Mapping (ORM)?

When we work with an object-oriented system like JDBC, there may be a mismatch between the **object model** and the **relational database table**. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java represent data in objects.

**Ex:**

****

**Following problems may arise with direct interaction b/w object models and DB tables:**

1. what if we need to modify the design of our database after having developed a few pages or our application?
2. loading and storing objects in a relational database exposes us to five mismatch problems:
   1. **Granularity**: Having a greater number of java classes than DB tables.
   2. **Inheritance**: RDBMSs do not define anything like Inheritance.
   3. **Identity**: An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).
   4. **Associations**: Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column.
   5. **Navigation:** The ways you access objects in Java and in RDBMS are fundamentally different.

The Object-Relational Mapping (ORM) is the solution to handle all the above mismatches.

**What is ORM?**

ORM stands for Object-Relational Mapping (ORM) is a programming technique **for converting data between relational databases and object-oriented** programming languages such as Java, C#, etc.

An ORM system has the following **advantages** over plain JDBC:

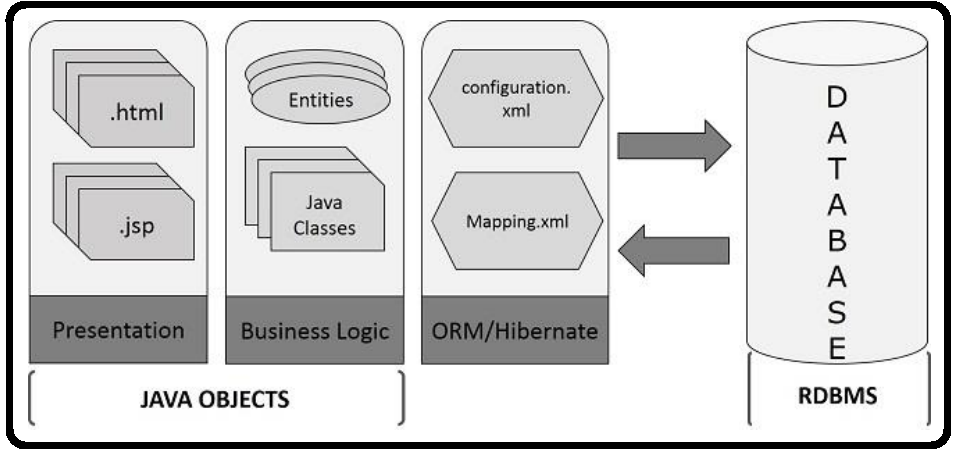
1. ORM reduces boilerplate code, making database interactions more concise and readable.
2. ORM aligns with the object-oriented paradigm, making it more natural for Java developers.
3. Streamlines database interactions, reducing boilerplate code and accelerating development.
4. ORM frameworks can generate, and update database schemas based on the entity classes, reducing manual schema management.
5. ORM provides high-level query languages (HQL in case of Hibernate) that are more natural for developers.
6. Fast development of application.
7. Hides details of SQL queries from OO logic.

**Different ORM Technologies:**

1. Hibernate
2. Spring JPA
3. Castor
4. TopLink and more...

**Hibernate Overview:**

1. Hibernate is an Object-Relational Mapping (ORM) solution for JAVA.
2. Hibernate maps Java classes to database tables.
3. Hibernate maps Java data types to SQL data types.
4. Hibernate relieves the developer from most common data persistence related programming tasks.
5. Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects.



**Hibernate Advantages:**

1. Hibernate takes care of mapping Java classes to database tables using XML files or annotations.
2. Provides simple APIs (classes and methods) for storing and retrieving Java objects directly to and from the database.
3. Minimizes database access with smart fetching strategies.
4. Hibernate has its own query language. That is Hibernate Query Language (HQL) which helps for DB communications for complex queries.
5. Hibernate supports caching mechanism.
6. Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java data types.
7. Hibernate Supports only unchecked exceptions, so no need to write try, catch, or throws blocks. Generally, we have a Hibernate translator which converts Checked exceptions to Unchecked.

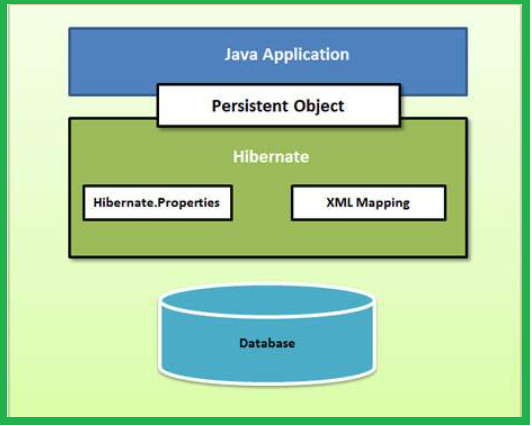
**Supported Databases:**

1. Oracle
2. MySql
3. PostgreSql
4. DB2
5. Microsoft SQL Server
6. FrontBase and most of the RDMS databases.

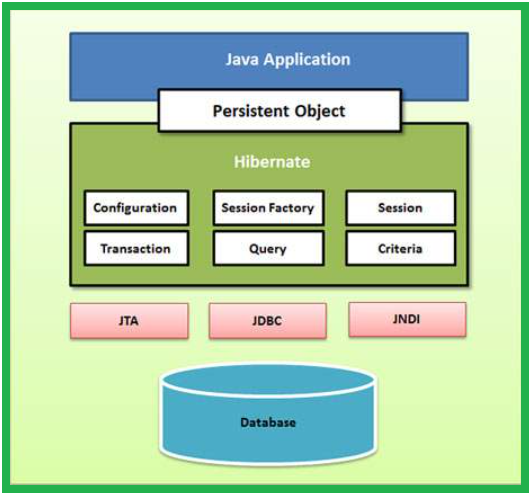
**Hibernate Architecture:**

Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs.

Below is the high level Hibernate Arch:



Below is the detailed Hibernate Arch:



Configuration Object:

It is the first Hibernate object that you need to create in any Hibernate application.

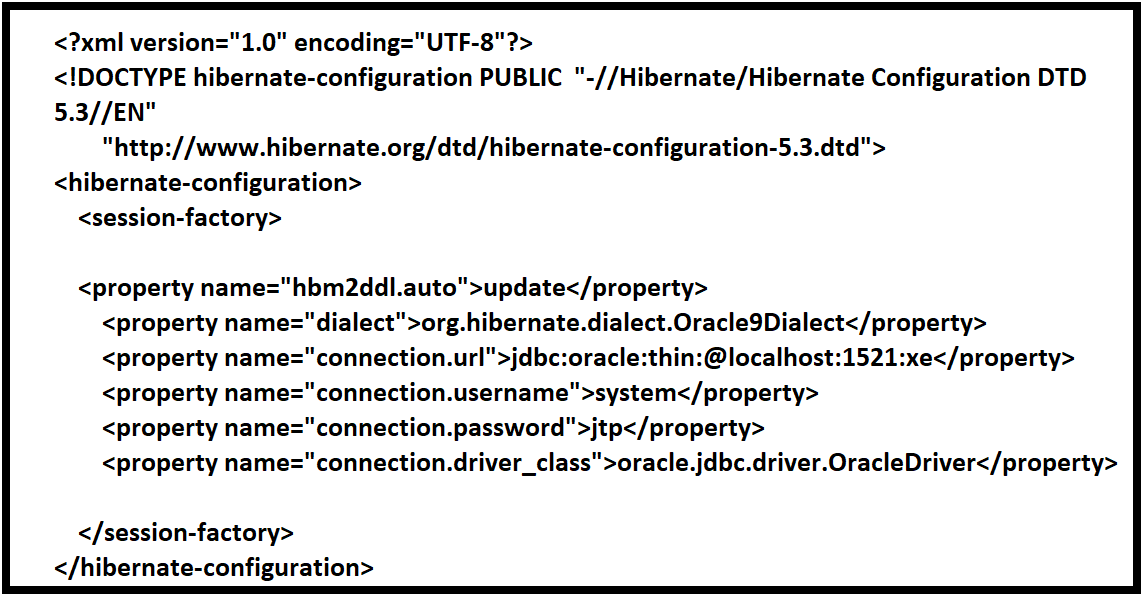
It is created only once during application initialization.

It specifies the properties and mapping files to be used by the application.

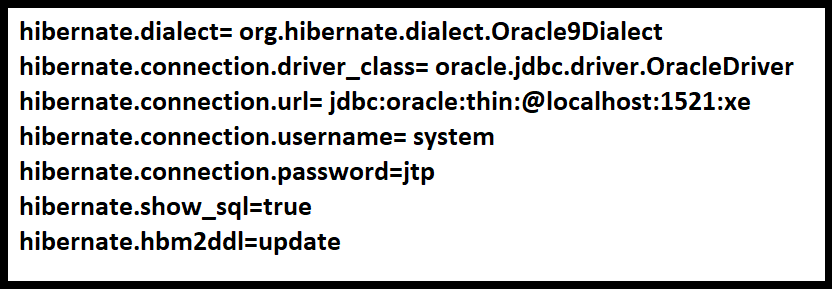
Configuration object provides two key components:

**Database Connection:** A database connection handled through one or more configuration files supported by Hibernate. Those are **hibernate.properties** file and **hibernate.cfg.xml** file.

Sample hibernate.cfg.xml:

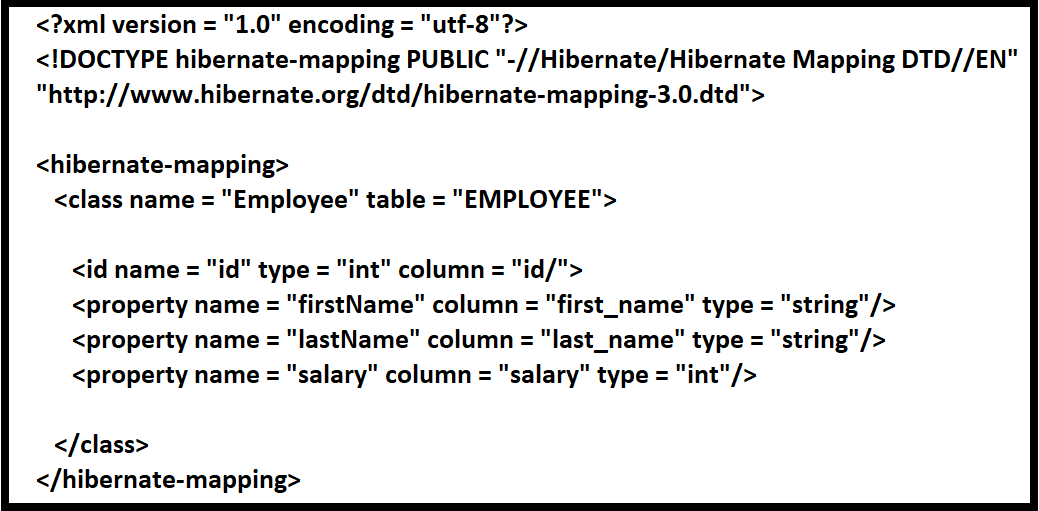


Sample hibernate.properties file:



**Mapping Setup:** This file creates the connection between the Java classes and database tables. Which means it creates mapping between each entity java class and each table in the database. We define these mappings in .hbm.xml files.

Ex: employee.hbm.xml



**SessionFactory Object:**

1. SessionFactory is a Factory Interface used to create Session instances.
2. After adding the properties and Mapping files to the Configuration object, it is used to create a SessionFactory object.
3. SessionFactory is a thread-safe object and used by all the threads of an application.
4. It is a heavyweight object, usually created during application start-up.
5. We need one SessionFactory object per database in our application.
6. if we are using multiple databases in our application, then you would have to create multiple SessionFactory objects.

**Session Object:**

1. Session is an Interface that wraps the JDBC connection. That means, it creates a physical connection between the application and a database.
2. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.
3. Persistent objects are saved and retrieved through a Session object.
4. The Lifecycle of a Session is bounded by the **beginning and end** of a logical transaction.
5. Session object has three states:
   1. Transient: never persistent, currently not associated with any Session.
   2. Persistent: currently associated with unique Session.
   3. Detached: previously persistent, currently not associated with any Session.

**Note:** The session objects should not be kept open for a long time because they are not usually thread safe. They should be created and destroyed them as needed.

**Transaction Object:**

1. Transaction is an Interface, and it represents a unit of work with the database and most of the RDBMS supports transaction functionality.
2. Transactions in Hibernate are handled by an underlying transaction manager.

**Query Object:**

Query is an interface, and it is used in SQL or Hibernate Query Language (HQL) to retrieve data from the database.

A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

**Criteria Object:**

Criteria is an interface and it is used for retrieving entity data by composing Criterion objects.

**Criterion** Objects work like a condition (WHERE and IF) in the SQL query, all the criterion objects (conditions) are added to the Criteria Object and that object will be executed and used for retrieving entity data in objects.

**Environment SetUp :** TRY yourself.. Google it.

**Hibernate Configuration: Detailed.**

Hibernate requires finding the mapping information that defines how your Java classes relate to the database tables and set of configuration settings to identify the respective database and mapping files. All such information is usually containing in 3 types of files:

1. Connection properties
2. Hibernate properties.
3. Mapping file names

**Note:** Most of the properties take their default values and it is not required to specify them in the property file unless it is really required.

**Hibernate Properties:**

1. **hibernate.dialect:**
   1. This property uses to determine which database is being used so that it can switch to the database specific SQL generator code.
2. **hibernate.connection.driver\_class :** The JDBC driver class.
3. **hibernate.connection.url:** The JDBC URL to the database instance.
4. **hibernate.connection.username :** The database username.
5. **hibernate.connection.password :** The database password.
6. **hibernate.connection.pool\_size:** Limits the number of connections waiting in the Hibernate database connection pool.
7. **hibernate.connection.autocommit:** Allows auto-commit mode to be used for the JDBC connection.

**Properties to configure with an application server for JNDI:**

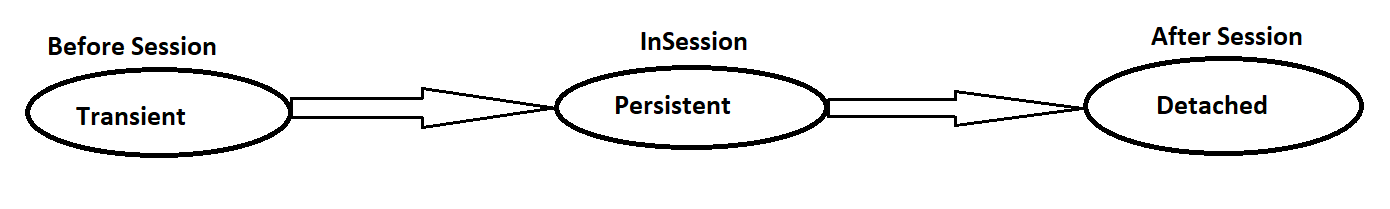
1. **hibernate.connection.datasource:** This property specifies the JNDI name defined in the application server context.
2. **hibernate.jndi.class** : Defines the InitialContext class for JNDI.
3. **hibernate.jndi.<JNDI\_Property\_Name> :** Passes any JNDI property you like to the JNDI InitialContext.
4. **hibernate.jndi.url:** Provides the URL for JNDI.
5. **hibernate.connection.username:** Defines the database username.
6. **hibernate.connection.password:** Defines the databasepassword.

EX: Hibernate with Oracle as follows.

* **Dialect**: org.hibernate.dialect.Oracle10gDialect
* **Driver Class**: oracle.jdbc.OracleDriver
* **Connection URL**: jdbc:oracle:thin:@localhost:1521:dbname
* **User Name**: scott
* **Password**: tiger
* **Mapping file**: employee.hbm.xml

**Session: Detailed**

* A Session is an **Interface** that is used to establish a **physical connection** with a database.
* A Session in Hibernate works like a Connection in JDBC.
* The session opens a single database connection when it is created and holds the connection until the session is closed.
* Every entity class object that is loaded by Hibernate to and from the database server is associated with the session.
* Session object is lightweight.
* Session object instantiated each time an interaction is needed with the database.
* Persistent objects are saved and retrieved through a Session object.
* The main function of a Session object is to create, read, update, and delete operations for instances of mapped entity classes.
* Instances may exist in one of the following three states at a given point in time:

****

**Transient:** A new instance of a persistent class, which is not associated with any Session is considered as transient state.

**Persistent:** You can make a transient instance persistent by associating it with a Session. We have methods on Session object to make an object move from transient state to persistent state.

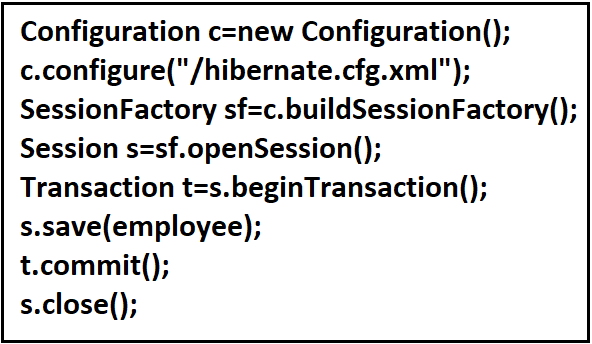
**Detached:** Once we close the Hibernate Session, the persistent instance will become a detached instance.

**Note:** A Session is a Serializable interface, so Session instance is also serializable.

**Sample Code:**

**Steps:**

* The instance of a Configuration class is parsing the hibernate configuration file (hibernate.cfg.xml) to configure the hibernate application.
* Using the Configuration object, we create SessionFactory Object. One SessionFactory object for application.
* Using this SessionFactoy instance we can create a Session object (s) by calling "sf.openSession();"
* we can create a transaction object through the Session object to maintain consistency.

****

**Session Interface Methods:**

The Session interface provides several methods, few important methods are here:

1. beginTransaction(): To start the transaction.
2. cancelQuery(): Cancel the execution of the query.
3. Clear(): Completely clear the session object.
4. Close(): End the session by releasing the JDBC connection and cleaning up.
5. createCriteria(Class persistentClass): Create a new Criteria instance for the given entity class.
6. createCriteria(String entityName): Create a new Criteria instance, for the given entity name.
7. getIdentifier(Object object): Return the identifier value of the given entity as associated with this session.
8. createQuery(String queryString): Create a new instance of Query for the given HQL query string.
9. createSQLQuery(String queryString): Create a new instance of SQLQuery for the given SQL query string.
10. delete(Object object): Remove a persistent instance from the datastore.
11. getSessionFactory(): Get the session factory, which created this session.
12. refresh(Object object): Re-read the state of the given instance from the underlying database.
13. getTransaction(): Get the Transaction instance associated with this session.
14. isConnected(): Check if the session is currently connected.
15. isDirty(): Does this session contain any changes, which must be synchronized with the database?
16. isOpen(): Check if the session is still open.
17. save(Object object): Persist the given transient instance with a generated Identifier.
18. saveOrUpdate (Object object): Either save(Object) or update(Object) the given instance.
19. update (Object object): Update the persistent instance.

**Hibernate Persistent Class:**