

HIBERNATE

Prepared By: Prof Shital Pathar

OVERVIEW

- Hibernate is a high-performance Object/Relational persistence and query service which is licensed under the open source GNU Lesser General Public License (LGPL) and is free to download.
- Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities.

What is JDBC?

- JDBC stands for **Java Database Connectivity** and 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.
- JDBC provides a flexible architecture to write a database independent application that can run on different platforms and interact with different DBMS without any modification.

Pros and Cons of JDBC

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

Why Object Relational Mapping (ORM)?

- When we work with an object-oriented systems, there's a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.

JAVA PROGRAM

- public class Employee
{ private int id;
 private String first_name;
 private String last_name;
 private int salary;
 public Employee() {}
 public Employee(String fname, String lname, int salary) {
 this.first_name = fname; this.last_name =
lname; this.salary = salary; }
 public int setId()

RDBMS table:

- create table EMPLOYEE (
 id INT NOT NULL auto_increment,
 first_name VARCHAR(20) default NULL,
 last_name VARCHAR(20) default NULL,
 salary INT default NULL, PRIMARY KEY
 (id));

Problem

1. what if we need to modify the design of our database after having developed few pages or our application?
2. Loading and storing objects in a relational database exposes us to the following five mismatch problems.

Mismatch

Mismatch	Description
Granularity	Sometimes you will have an object model which has more classes than the number of corresponding tables in the database.
Inheritance	RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages.
Identity	A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (<code>a==b</code>) and object equality (<code>a.equals(b)</code>).
Associations	Object-oriented languages represent associations using object references where as an RDBMS represents an association as a foreign key column.
Navigation	The ways you access objects in Java and in a RDBMS are fundamentally different.

What is ORM?

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

ADVANTAGES OF ORM

- Let's business code access objects rather than DB tables.
- Hides details of SQL queries from OO logic.
- Based on JDBC 'under the hood'
- No need to deal with the database implementation.
- Entities based on business concepts rather than database structure.
- Transaction management and automatic key generation.

ORM solution

- An API to perform basic CRUD operations on objects of persistent classes.
- A language or API to specify queries that refer to classes and properties of classes.
- A configurable facility for specifying mapping metadata.
- A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions.

Java ORM Frameworks:

- Enterprise JavaBeans Entity Beans
- Java Data Objects
- Castor
- TopLink
- Spring DAO
- Hibernate
- And many more

Why Hibernate and not JDBC?

- JDBC maps Java classes to database tables (and from Java data types to SQL data types)
- Hibernate automatically generates the SQL queries.
- Hibernate provides data query and retrieval facilities and can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC.
- Makes an application portable to all SQL databases.

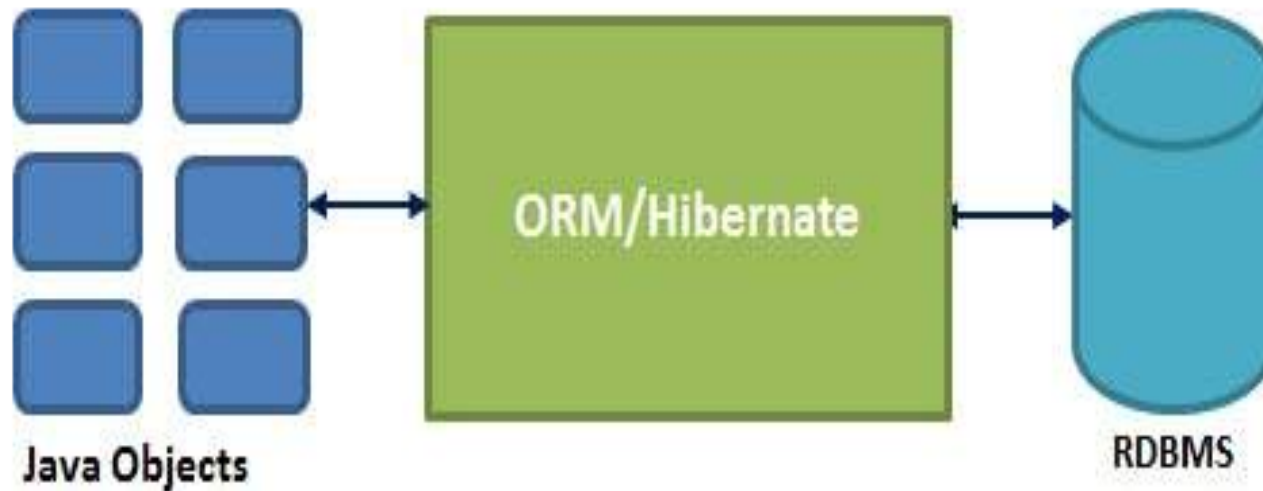
These are some advantages of hibernate.

Hibernate vs. JDBC (an example)

- *JDBC tuple insertion* –
st.executeUpdate(“INSERT INTO book
VALUES(“Harry Potter”,”J.K.Rowling”)”);
- *Hibernate tuple insertion* –
session.save(book1);

Hibernate - Overview

- Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.
- Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming



- Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.

Hibernate Advantages:

- Hibernate takes care of **mapping Java classes to database tables using XML files** and without writing any line of code.
- Provides **simple APIs for storing and retrieving Java objects** directly to and from the database.
- If there is change in Database or in any table then the only need to **change XML file** properties.
- **Abstract** away the unfamiliar SQL types and provide us to work around familiar Java Objects.

Supported Databases:

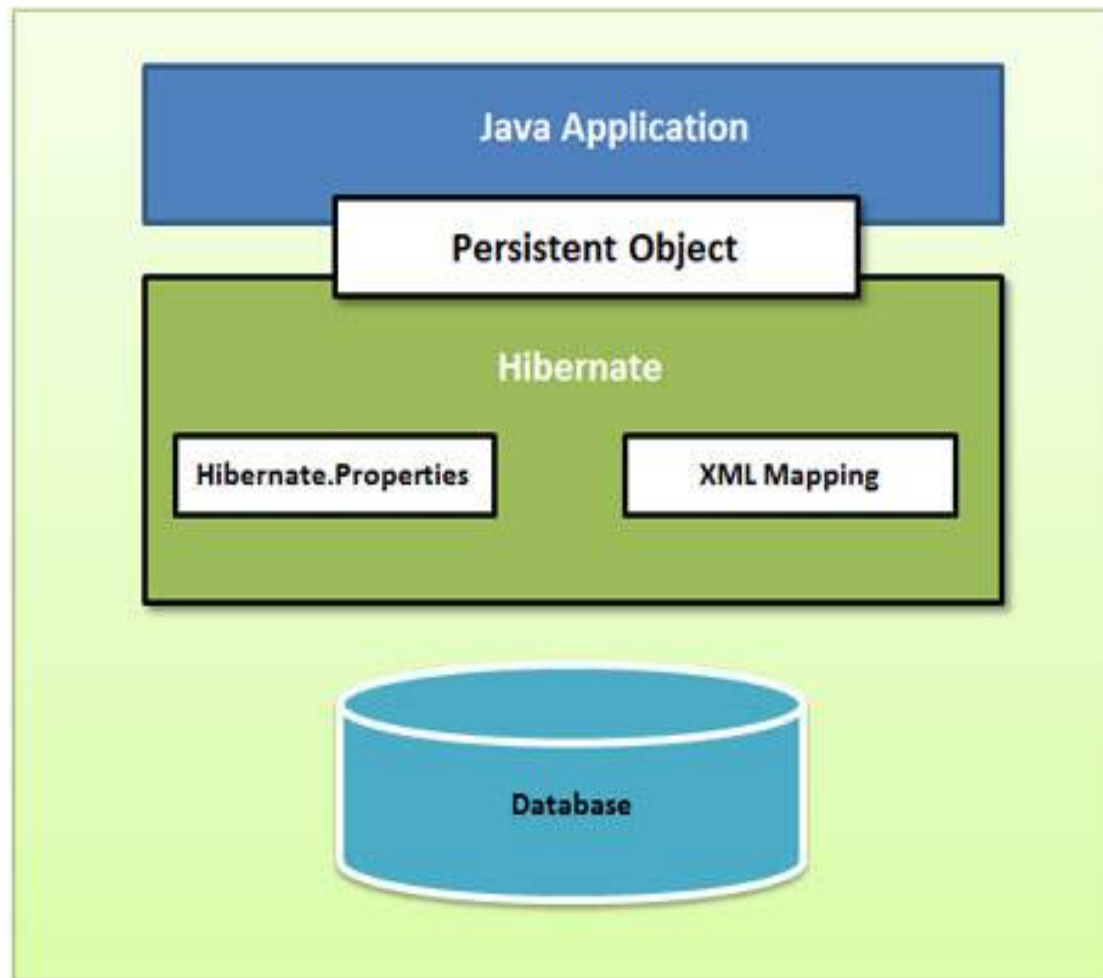
- HSQL Database Engine
- DB2/NT
- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

Supported Technologies:

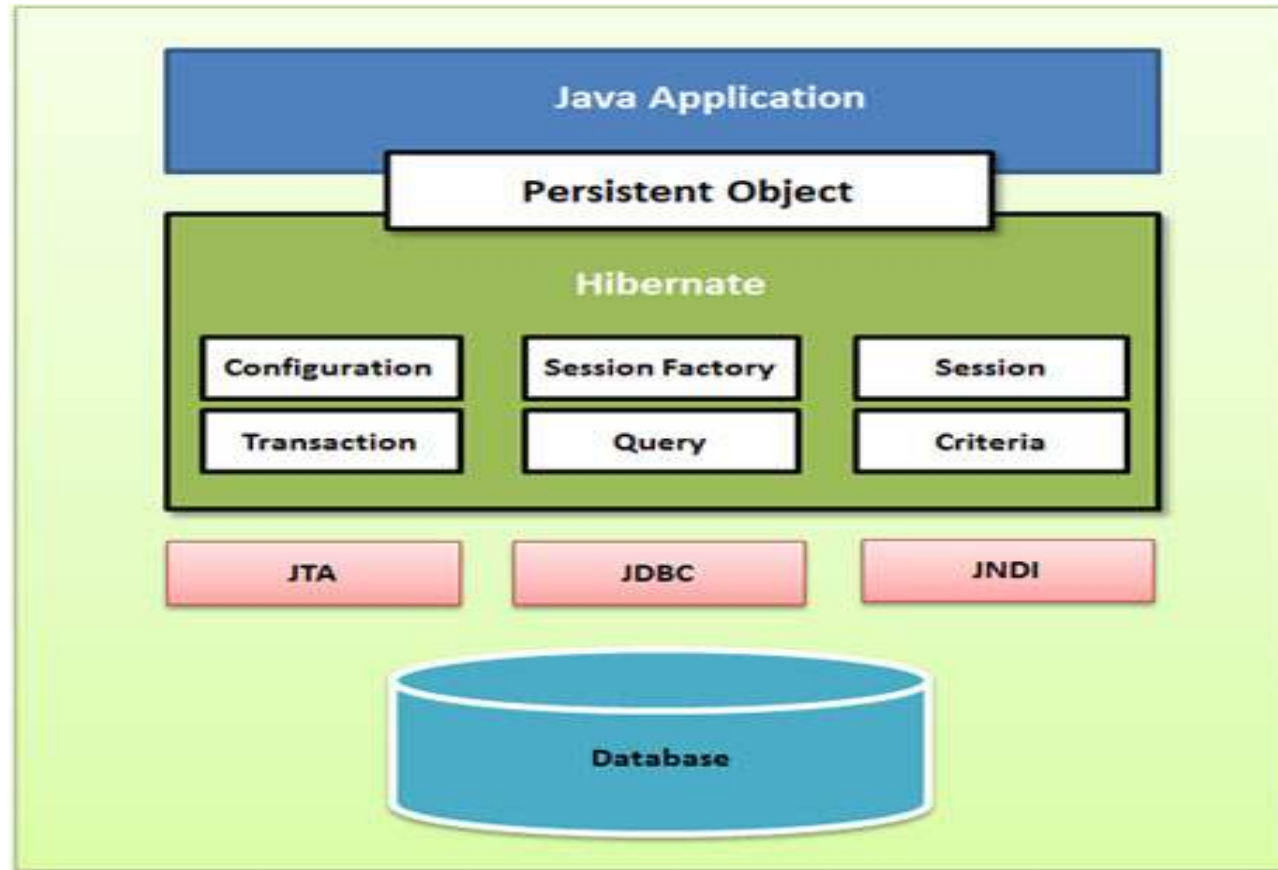
- XDoclet Spring
- J2EE
- Eclipse plug-ins
- Maven

Hibernate - Architecture

- The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.



detailed view of the Hibernate Application Architecture



- Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI).
- JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Configuration Object:

- The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two key components:
- **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**

SessionFactory Object:

- Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.
- The SessionFactory is a heavyweight object so usually it is created during application start up and kept for later use. You would need one

Session Object:

- A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.
- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed

Transaction Object:

- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).
- This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

Query Object:

- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. 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 object are used to create and execute object oriented criteria queries to retrieve objects.

Hibernate - Mapping Types

- When you prepare a Hibernate mapping document, we have seen that you map Java data types into RDBMS data types. The **types** declared and used in the mapping files are not Java data types; they are not SQL database types either. These types are called Hibernate mapping types, which can translate from Java to SQL data types and vice versa.

Primitive types:

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

Date and time types:

Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

Binary and large object types:

Mapping type	Java type	ANSI SQL Type
binary	byte[]	VARBINARY (or BLOB)
text	java.lang.String	CLOB
serializable	any Java class that implements java.io.Serializable	VARBINARY (or BLOB)
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB

JDK-related types:

Mapping type	Java type	ANSI SQL Type
binary	byte[]	VARBINARY (or BLOB)
text	java.lang.String	CLOB
serializable	any Java class that implements java.io.Serializable	VARBINARY (or BLOB)
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB

Hibernate O/R Mapping

1. **Collections Mappings**
2. **Association Mappings**
3. **Component Mappings**





Collections Mappings:

- If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java. Hibernate can persist instances of
 - **>java.util.Map**
 - **> java.util.Set**
 - **java.util.SortedMap**
 - **> java.util.SortedSet**
 - **> java.util.List,**
 - any **array** of persistent entities or values.

Collection type	Mapping and Description
java.util.Set ↗	This is mapped with a <set> element and initialized with java.util.HashSet
java.util.SortedSet ↗	This is mapped with a <set> element and initialized with java.util.TreeSet. The sort attribute can be set to either a comparator or natural ordering.
java.util.List ↗	This is mapped with a <list> element and initialized with java.util.ArrayList
java.util.Collection ↗	This is mapped with a <bag> or <ibag> element and initialized with java.util.ArrayList
java.util.Map ↗	This is mapped with a <map> element and initialized with java.util.HashMap
java.util.SortedMap ↗	This is mapped with a <map> element and initialized with java.util.TreeMap. The sort attribute can be set to either a comparator or natural ordering.

Association Mappings:

- The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways

Mapping type	Description
Many-to-One 	Mapping many-to-one relationship using Hibernate
One-to-One 	Mapping one-to-one relationship using Hibernate
One-to-Many 	Mapping one-to-many relationship using Hibernate
Many-to-Many 	Mapping many-to-many relationship using Hibernate

Component Mappings

- It is very much possible that an Entity class can have a reference to another class as a member variable. If the referred class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence, therefore is called as the

Mapping type	Description
Component Mappings ↗	Mapping for a class having a reference to another class as a member variable.

Hibernate - Annotations

- Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.
- If you going to make your application portable to other EJB 3 compliant ORM applications, you must use annotations to represent the

Environment Setup for Hibernate Annotation

- First of all you would have to make sure that you are using JDK 5.0 otherwise you need to upgrade your JDK to JDK 5.0 to take advantage of the native support for annotations.
- Second, you will need to install the Hibernate 3.x annotations distribution package, available from the nd copy **hibernate-annotations.jar**, **lib/hibernate-comons-annotations.jar** and **lib/hib2_persistence.jar** from the Hibernate

Various Annotations

1. **@Entity Annotation:**
2. **@Table Annotation:**
3. **@Id and @GeneratedValue Annotations:**
4. **@Column Annotation:**

- `@Entity`

```
@org.hibernate.annotations.Entity(optimisticLock = OptimisticLockType.ALL)
```

```
@Table(name = "Employee", uniqueConstraints = {
```

```
    @UniqueConstraint(columnNames = "ID"),
```

```
    @UniqueConstraint(columnNames = "EMAIL") })
```

```
public class EmployeeEntity implements Serializable {
```

```
    private static final long serialVersionUID =
```

***@Entity* Annotation:**

- We use the *@Entity* annotation to the particular class which marks that class as an entity bean.

@Table Annotation:

- The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.
- The @Table annotation provides four attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table.

@Id

- Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.
- By default, the **@Id** annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the **@GeneratedValue** annotation which takes

@Column Annotation:

- The `@Column` annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes:
- **name** attribute permits the name of the column to be explicitly specified.
- **length** attribute permits the size of the column used to map a value particularly for a String value.

11

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 which in turns perform action on database.
- Although you can use SQL statements directly with Hibernate using Native SQL but I would recommend to use HQL whenever possible to avoid database portability hassles and to take

Advantages of HQL

1. The HQL can perform bulk of operations at a time on hibernate.
2. HQL supports object oriented features such as inheritance ,polymorphism,association and so on.
3. Instead of returning plain data HQL queries returns the objects .These objects can be easily accessed ,or programmed.
4. HQL is database independent.The same HQL can be executed on different databases

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

1.FROM Clause

- You will use **FROM** clause if you want to load a complete persistent objects into memory.

Following is the simple syntax of using FROM clause:

- `String hql = "FROM Employee";`
- `Query query = session.createQuery(hql);`
- `List results = query.list();`

2.AS Clause

- The **AS** clause can be used to assign aliases to the classes in your HQL queries, specially when you have long queries. For instance, our previous simple example would be the following:
- `String hql = "FROM Employee AS E";`
- `Query query = session.createQuery(hql);`
- `List results = query.list();`

3.SELECT Clause

- The **SELECT** clause provides more control over the result set than the from clause. If you want to obtain few properties of objects instead of the complete object, use the SELECT clause. Following is the simple syntax of using SELECT clause to get just first_name field of the Employee object:
- `String hql = "SELECT E.firstName FROM Employee E";`
- `Query query = session.createQuery(hql);`

return query.list();

4.WHERE Clause

- If you want to narrow the specific objects that are returned from storage, you use the WHERE clause. Following is the simple syntax of using WHERE clause:
- `String hql = "FROM Employee E WHERE E.id = 10";`
- `Query query = session.createQuery(hql);`
- `List results = query.list();`

5.ORDER BY Clause

- To sort your HQL query's results, you will need to use the **ORDER BY** clause. You can order the results by any property on the objects in the result set either ascending (ASC) or descending (DESC). Following is the simple syntax of using ORDER BY clause:
- `String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";`
- `Query query = session.createQuery(hql);`
- `List results = query.list();`

6.GROUP BY Clause

- This clause lets Hibernate pull information from the database and group it based on a value of an attribute and, typically, use the result to include an aggregate value. Following is the simple syntax of using GROUP BY clause:
- `String hql = "SELECT SUM(E.salary),
E.firtName FROM Employee E " + "GROUP
BY E.firstName";`
- `Query query = session.createQuery(hql);`

return query.list();

7.UPDATE Clause

- Bulk updates are new to HQL with Hibernate 3, and deletes work differently in Hibernate 3 than they did in Hibernate 2. The Query interface now contains a method called `executeUpdate()` for executing HQL UPDATE or DELETE statements.
- The **UPDATE** clause can be used to update one or more properties of an one or more objects. Following is the simple syntax of using UPDATE clause:
 - `String hql = "UPDATE Employee set salary =`

8.DELETE Clause

- The **DELETE** clause can be used to delete one or more objects. Following is the simple syntax of using DELETE clause:
- `String hql = "DELETE FROM Employee " + "WHERE id = :employee_id";`
- `Query query = session.createQuery(hql);`
- `query.setParameter("employee_id", 10);`
- `int result = query.executeUpdate();`
- `System.out.println("Rows affected: " + result);`

9.INSERT Clause

- HQL supports **INSERT INTO** clause only where records can be inserted from one object to another object. Following is the simple syntax of using INSERT INTO clause:
- `String hql = "INSERT INTO Employee(firstName, lastName, salary)" + "SELECT firstName, lastName, salary FROM old_employee";`
- `Query query = session.createQuery(hql);`
- `int result = query.executeUpdate();`

Aggregate Methods

- HQL supports a range of aggregate methods, similar to SQL. They work the same way in HQL as in SQL and following is the list of the available functions:

S.N.	Functions	Description
1	avg(property name)	The average of a property's value
2	count(property name or *)	The number of times a property occurs in the results
3	max(property name)	The maximum value of the property values
4	min(property name)	The minimum value of the property values
5	sum(property name)	The sum total of the property values

THANK YOU