HIBERNATE

# HQL

**SUBJECT:**
Advanced Java Programming

**TOPIC:**
Hibernate (HQL)

# Introduction

**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 database.

JDBC provides a flexible architecture to write a database independent application that can run on different platforms and interact with different DBMS.

# Pros and Cons of JDBC

**Pros of JDBC**

- Clean and simple SQL processing
- Good performance with large data
- Very good for small applications
- Simple syntax so easy to learn

**Cons of JDBC**

- Complex if it is used in large projects
- Large programming overhead
- No encapsulation
- Hard to implement MVC concept
- Query is DBMS specific

# Example

```
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;
}
```

# Cont.

```
public int getId() {
    return id;      }
public String getFirstName() {
    return first_name;          }
public String getLastName() {
    return last_name;           }
public int getSalary() {
    return salary;      } }
```

# 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)
);
```

# Problems

First problem, what if we need to modify the design of our database after having developed few pages or our application?

Second, Loading and storing objects in a relational database exposes us to the following five mismatch problems.

# Problems??

**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.

# Problems??

Identity : A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).

Associations : Object-oriented languages represent associations using object references where as am RDBMS represents an association as a foreign key column.

Navigation : The ways you access objects in Java and in a RDBMS are fundamentally different.

# Hibernate

Hibernate is a high-performance Object/Relational 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.

# 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.

# What is ORM?

**O**bject-**R**elational **M**apping (ORM) is the solution to handle all the mismatches.

ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc.

# Cont.

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 tasks.

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.

# Cont.

# 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.

# Architecture

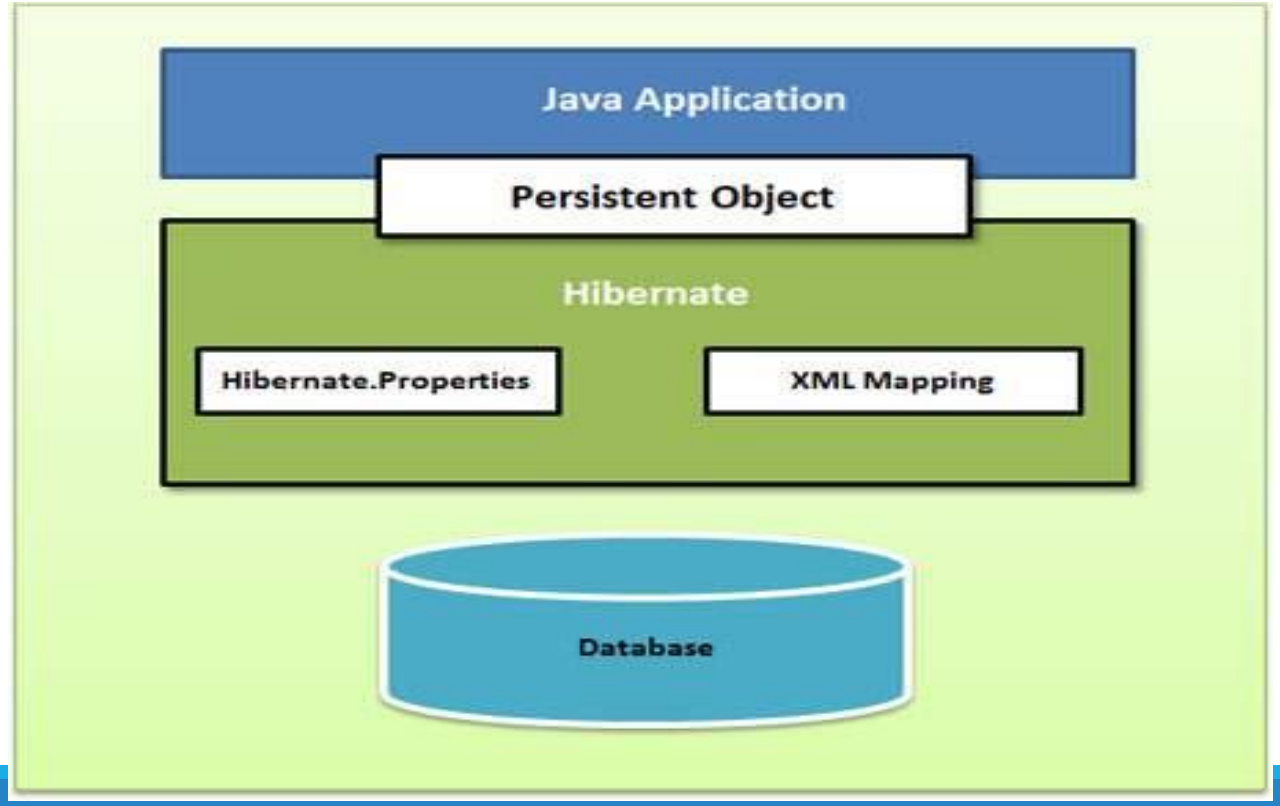
The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs.

Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Persistent Objects are the ones who give an indication that the state of an object would be permanently stored even after the execution of the program.

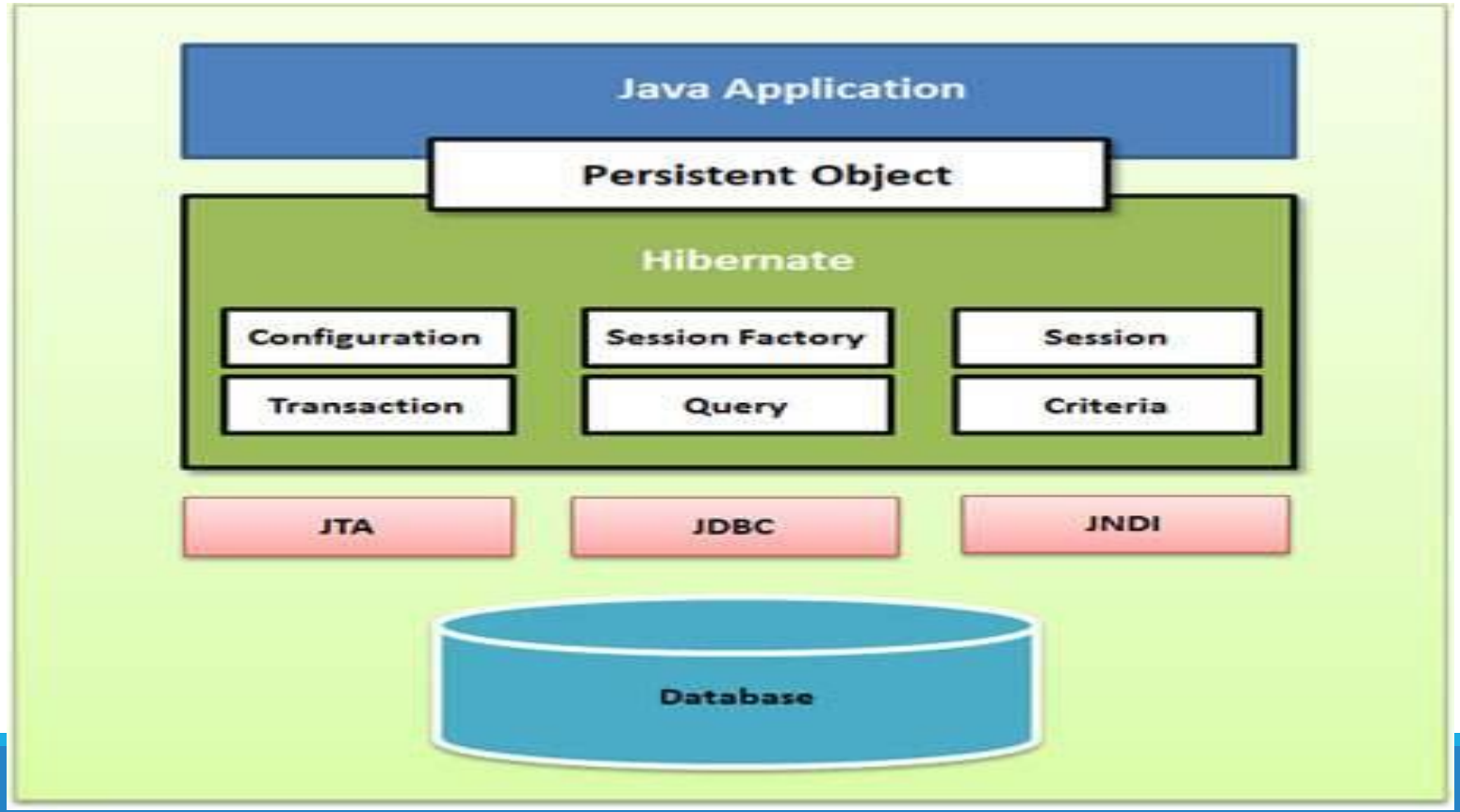
# Cont.

Following is a very high level view of the Hibernate Application Architecture.



# Cont.

Following is a detailed view of the Hibernate Application Architecture with few important core classes.



# Cont.

Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI).

JDBC provides 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.

# Cont.

## Configuration Object:

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization.

## SessionFactory Object:

Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.

# Cont.

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.

# Cont.

## 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 - O/R Mappings

These are the 3 mapping.

1. Mapping of collections,
2. Mapping of associations between entity classes
3. Component Mappings.

# Cont.

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, and any array of persistent entities or values.

# Cont

| 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. |

# Cont.

If you want to map a user defined collection interfaces which is not directly supported by Hibernate, you need to tell Hibernate about the semantics of your custom collections

# Cont.

Association Mappings:

The mapping of associations between entity classes and the relationships between tables is the soul of ORM.

Following are the four ways in which the cardinality of the relationship between the objects can be expressed.

An association mapping can be unidirectional as well as bidirectional.

# Cont.

| 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 |

# Cont.

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 it's 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 Component class.

# Cont.

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

# Hibernate - Mapping Files

An Object/relational mappings are usually defined in an XML document.

This mapping file instructs Hibernate how to map the defined class or classes to the database tables.

# Cont.

The important elements of the Hibernate mapping file are as follows.

- **DOCTYPE:** Refers to the Hibernate mapping Document Type Declaration (DTD) that should be declared in every mapping file for syntactic validation of the XML.
- **<hibernate-mapping> element:** Refers as the first or root element of Hibernate, inside <hibernate-mapping> tag any number of class may be present.
- **<class> element:** Maps a class object with its corresponding entity in the database.
- **<id> element :** Serves as a unique identifier used to map primary key column of the database table.

# Cont.

- **<generator> element:** Helps in generation of the primary key for the new record, following are some of the commonly used generators.
  - **Increment**
  - **Sequence**
  - **Assigned**
  - **Identity**
  - **Hilo**
  - **Native**
- **<property> element:** Defines standard Java attributes and their mapping into database schema.

# 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.

# Cont.

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 |

# Cont.

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 |

# Cont.

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 |

# Cont.

JDK-related types:

| Mapping type | Java type | ANSI SQL Type |
|---|---|---|
| class | java.lang.Class | VARCHAR |
| locale | java.util.Locale | VARCHAR |
| timezone | java.util.TimeZone | VARCHAR |
| currency | java.util.Currency | VARCHAR |

# 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.

# Cont.

Although you can use SQL statements directly with Hibernate using Native SQL but It would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation.

Keywords like SELECT , FROM and WHERE etc. are not case sensitive but properties like table and column names are case sensitive in HQL.

# FROM Clause

You will use FROM clause if you want to load a complete persistent objects into memory. Following is the simple

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

If you need to fully qualify a class name in HQL, just specify the package and class name as follows:

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

# 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);
- List results = query.list();

# 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();

# ORDER BY Clause

To sort your HQL query's results, you will need to use the ORDER BY clause. 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();

If you wanted to sort by more than one property, clause, separated by commas as follows:

- String hql = "FROM Employee E WHERE E.id > 10 " +
- "ORDER BY E.firstName DESC, E.salary DESC ";
- Query query = session.createQuery(hql);
- List results = query.list();

# 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

- String hql = "SELECT SUM(E.salary), E.firstName FROM Employee E " +
- "GROUP BY E.firstName";
- Query query = session.createQuery(hql);
- List results = query.list();

# UPDATE Clause

The Query interface now contains a method called executeUpdate() for executing HQL UPDATE or DELETE statements.

- String hql = "UPDATE Employee set salary = :salary " +
- "WHERE id = :employee_id";
- Query query = session.createQuery(hql);
- query.setParameter("salary", 1000);
- query.setParameter("employee_id", 10);
- int result = query.executeUpdate();
- System.out.println("Rows affected: " + result);

# 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);
```

# INSERT Clause

HQL supports INSERT INTO clause only where records can be inserted from one object to another object.

- String hql = "INSERT INTO Employee(firstName, lastName, salary)" +
- "SELECT firstName, lastName, salary FROM old_employee";
- Query query = session.createQuery(hql);
- int result = query.executeUpdate();
- System.out.println("Rows affected: " + result);

# 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:

| Functions | Description |
|---|---|
| avg(property name) | The average of a property's value |
| count(property name or *) | The number of times a property occurs in the results |
| max(property name) | The maximum value of the property values |
| min(property name) | The minimum value of the property values |
| sum(property name) | The sum total of the property values |