

The background of the slide is a scenic photograph of a calm lake reflecting the surrounding green mountains and a clear blue sky. The reflection is sharp and clear.

# Hibernate Framework Session-2

- HQL

# Hibernate Query Language (HQL)

- 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 we can use SQL statements directly with Hibernate using Native SQL but it is recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies.
- Keywords like SELECT , FROM and WHERE etc. are not case sensitive but properties like table and column names are case sensitive in HQL.

# Hibernate Query Language (HQL)

## FROM Clause

- Use **FROM** clause to load 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();
```

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

```
String hql = "FROM org.asr.hibernate.criteria.Employee";  
Query query = session.createQuery(hql);  
List results = query.list();
```

# Hibernate Query Language (HQL)

## AS Clause

**AS** clause can be used to assign aliases to the classes in HQL queries, specially when we 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();
```

The **AS** keyword is optional, we can also specify the alias directly after the class name, as follows:

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

# Hibernate Query Language (HQL)

## SELECT Clause

The **SELECT** clause provides more control over the result set than the from clause. 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 firstName field of the Employee object:*

```
String hql = "SELECT E.firstName FROM Employee E";  
String hq1 = "SELECT O.firstName, E.birthDate FROM EmployeePersonal E , EmployeeOfficial O  
              WHERE E.empno=O.empno";  
Query query = session.createQuery(hql);  
List results = query.list();
```

Note that Employee.firstName is a property of Employee object rather than a field of the EMPLOYEE table.

# Hibernate Query Language (HQL)

## WHERE Clause

To narrow the specific objects that are returned from storage, use 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 HQL query's results, use the **ORDER BY** clause. We 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();
```

*To sort by more than one property:*

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

## JPA

```
String sql="From Customer";
```

```
Query query=entityManager.createQuery(sql);  
List<Customer> customerList=query.getResultList();
```

# Hibernate Query Language (HQL)

## 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.deptno FROM Employee E " + "GROUP BY  
E.deptno";  
Query query = session.createQuery(hql);  
List results = query.list();
```

# Hibernate Query Language (HQL)

## Using Named Parameters

Hibernate supports *named parameters* in its HQL queries. This makes writing HQL queries that accept input from the user easy.

*Following is the simple syntax of using named parameters:*

```
String hql = "FROM Employee E WHERE E.id = :employee_id";  
Query query = session.createQuery(hql);  
query.setParameter("employee_id", 10);  
List results = query.list();
```



# Hibernate Query Language (HQL)

## UPDATE Clause

The Query interface 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 = :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);
```

# Hibernate Query Language (HQL)

## 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.

*Following is the simple syntax of using INSERT INTO clause:*

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

# Hibernate Query Language (HQL)

## 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

The **distinct** keyword only counts the unique values in the row set. The following query will return only unique count:

```
String hql = "SELECT count(distinct E.firstName) FROM Employee E";  
Query query = session.createQuery(hql);  
List results = query.list();
```

# Hibernate Query Language (HQL)

## Pagination using Query

There are two methods of the Query interface for pagination.

S.N.	Method & Description
1	<b><u>Query setFirstResult(int startPosition)</u></b> This method takes an integer that represents the first row in your result set, starting with row 0.
2	<b><u>Query setMaxResults(int maxResult)</u></b> This method tells Hibernate to retrieve a fixed number <u>maxResults</u> of objects.

Using above two methods together, we can construct a paging component in our web or Swing application.

*Following is the example which you can extend to fetch 10 rows at a time:*

```
String hql = "FROM Employee";  
Query query = session.createQuery(hql);  
query.setFirstResult(1);  
query.setMaxResults(10);  
List results = query.list();
```

# Hibernate Criteria Queries

Hibernate provides alternate ways of manipulating objects and in turn data available in RDBMS tables.

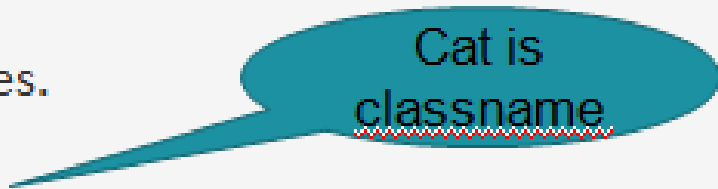
One of the methods is **Criteria API** which allows us to build up a criteria query object programmatically where we can apply filtration rules and logical conditions.

## Creating a Criteria instance

The interface `org.hibernate.Criteria` represents a query against a particular persistent class.

The Session is a factory for Criteria instances.

```
Criteria criteria = session.createCriteria(Cat.class);  
criteria.setMaxResults(50);  
List list = crit.list();
```



Cat is  
classname

# Hibernate Criteria Queries

## Restrictions with Criteria:

We can use **add()** method available for **Criteria** object to add restriction for a criteria query.

*Following is the example to add a restriction to return the records with salary is equal to 2000:*

```
Criteria criteria = session.createCriteria(Employee.class);
criteria.add(Restrictions.eq("salary", 2000));
List results = criteria.list();
```

```
List cats = session.createCriteria(Cat.class) .add(
Restrictions.like("name", "Fritz%")) .add(
Restrictions.between("weight", minWeight, maxWeight)
).list();
```

*Following are the few more examples covering different scenarios and can be used as per requirement:*

```
Criteria criteria = session.createCriteria(Employee.class);
criteria.add(Restrictions.gt("salary", 2000)); // To get records having salary more than 2000
criteria.add(Restrictions.lt("salary", 2000)); // To get records having salary less than 2000
criteria.add(Restrictions.like("firstName", "zara%")); // To get records having firstName starting with zara
criteria.add(Restrictions.ilike("firstName", "zara%")); // Case sensitive form of the above restriction.
criteria.add(Restrictions.between("salary", 1000, 2000)); // To get records having salary in between 1000 and 2000
criteria.add(Restrictions.isNull("salary")); // To check if the given property is null
criteria.add(Restrictions.isNotNull("salary")); // To check if the given property is not null
criteria.add(Restrictions.isEmpty("salary")); // To check if the given property is empty
criteria.add(Restrictions.isNotEmpty("salary")); // To check if the given property is not empty
```



Thank You!