


Hibernate - Native SQL

Advertisements

[⬅ Previous Page](#)

[Next Page ➡](#)

You can use native SQL to express database queries if you want to utilize database-specific features such as query hints or the CONNECT keyword in Oracle. Hibernate 3.x allows you to specify handwritten SQL, including stored procedures, for all create, update, delete,  load operations.

Your application will create a native SQL query from the session with the **createSQLQuery()** method on the Session interface –

```
public SQLQuery createSQLQuery(String sqlString) throws HibernateException
```

After you pass a string containing the SQL query to the createSQLQuery() method, you can associate the SQL result with either an existing Hibernate entity, a join, or a scalar result using addEntity(), addJoin(), and addScalar() methods respectively.

Scalar Queries

The most basic SQL query is to get a list of scalars (values) from one or more tables. Following is the syntax for using native SQL for scalar values –

```
String sql = "SELECT first_name, salary FROM EMPLOYEE";
SQLQuery query = session.createSQLQuery(sql);
query.setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP);
List results = query.list();
```

Entity Queries

The above queries were all about returning scalar values, basically returning the "raw" values from the result set. Following is the syntax to get entity objects as a whole from a native sql query via addEntity().

```
String sql = "SELECT * FROM EMPLOYEE";
SQLQuery query = session.createSQLQuery(sql);
```

```
query.addEntity(Employee.class);  
List results = query.list();
```

Named SQL Queries

Following is the syntax to get entity objects from a native sql query via addEntity() and using named SQL query.

```
String sql = "SELECT * FROM EMPLOYEE WHERE id = :employee_id";  
SQLQuery query = session.createSQLQuery(sql);  
query.addEntity(Employee.class);  
query.setParameter("employee_id", 10);  
List results = query.list();
```

Native SQL Example

Consider the following POJO class –

```
public class Employee {  
    private int id;  
    private String firstName;  
    private String lastName;  
    private int salary;  
  
    public Employee() {}  
  
    public Employee(String fname, String lname, int salary) {  
        this.firstName = fname;  
        this.lastName = lname;  
        this.salary = salary;  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    public void setId( int id ) {  
        this.id = id;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName( String first_name ) {  
        this.firstName = first_name;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
  
    public void setLastName( String last_name ) {  
        this.lastName = last_name;  
    }  
}
```

```

    public int getSalary() {
        return salary;
    }

    public void setSalary( int salary ) {
        this.salary = salary;
    }
}

```

Let us create the following EMPLOYEE table to store Employee objects –

```

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)
);

```

Following will be mapping file –

```

<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name = "Employee" table = "EMPLOYEE">

        <meta attribute = "class-description">
            This class contains the employee detail.
        </meta>

        <id name = "id" type = "int" column = "id">
            <generator class="native"/>
        </id>

        <property name = "firstName" column = "first_name" type = "string"/>
        <property name = "lastName" column = "last_name" type = "string"/>
        <property name = "salary" column = "salary" type = "int"/>

    </class>
</hibernate-mapping>

```

Finally, we will create our application class with the main() method to run the application where we will use **Native SQL** queries –

```

import java.util.*;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.SQLQuery;
import org.hibernate.Criteria;
import org.hibernate.Hibernate;
import org.hibernate.cfg.Configuration;

```

```

public class ManageEmployee {
    private static SessionFactory factory;
    public static void main(String[] args) {

        try {
            factory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Failed to create sessionFactory object." + ex);
            throw new ExceptionInInitializerError(ex);
        }

        ManageEmployee ME = new ManageEmployee();

        /* Add few employee records in database */
        Integer empID1 = ME.addEmployee("Zara", "Ali", 2000);
        Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
        Integer empID3 = ME.addEmployee("John", "Paul", 5000);
        Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

        /* List down employees and their salary using Scalar Query */
        ME.listEmployeesScalar();

        /* List down complete employees information using Entity Query */
        ME.listEmployeesEntity();
    }

    /* Method to CREATE an employee in the database */
    public Integer addEmployee(String fname, String lname, int salary){
        Session session = factory.openSession();
        Transaction tx = null;
        Integer employeeID = null;

        try {
            tx = session.beginTransaction();
            Employee employee = new Employee(fname, lname, salary);
            employeeID = (Integer) session.save(employee);
            tx.commit();
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
        return employeeID;
    }

    /* Method to READ all the employees using Scalar Query */
    public void listEmployeesScalar( ){
        Session session = factory.openSession();
        Transaction tx = null;

        try {
            tx = session.beginTransaction();
            String sql = "SELECT first_name, salary FROM EMPLOYEE";
            SQLQuery query = session.createSQLQuery(sql);
            query.setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP);
            List data = query.list();

            for(Object object : data) {
                Map row = (Map)object;
                System.out.print("First Name: " + row.get("first_name"));
            }
        }
    }
}

```

```

        System.out.println(", Salary: " + row.get("salary"));
    }
    tx.commit();
} catch (HibernateException e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
} finally {
    session.close();
}
}

/* Method to READ all the employees using Entity Query */
public void listEmployeesEntity( ){
    Session session = factory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();
        String sql = "SELECT * FROM EMPLOYEE";
        SQLQuery query = session.createSQLQuery(sql);
        query.addEntity(Employee.class);
        List employees = query.list();

        for (Iterator iterator = employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("   Salary: " + employee.getSalary());
        }
        tx.commit();
    } catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    } finally {
        session.close();
    }
}
}

```

Compilation and Execution

Here are the steps to compile and run the above mentioned application. Make sure, you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

Create hibernate.cfg.xml configuration file as explained in configuration chapter.

Create Employee.hbm.xml mapping file as shown above.

Create Employee.java source file as shown above and compile it.

Create ManageEmployee.java source file as shown above and compile it.

Execute ManageEmployee binary to run the program.

You would get the following result, and records would be created in the EMPLOYEE table.

```
$java ManageEmployee
.....VARIOUS LOG MESSAGES WILL DISPLAY HERE.....
```

```
First Name: Zara, Salary: 2000
First Name: Daisy, Salary: 5000
First Name: John, Salary: 5000
First Name: Mohd, Salary: 3000
First Name: Zara Last Name: Ali Salary: 2000
First Name: Daisy Last Name: Das Salary: 5000
First Name: John Last Name: Paul Salary: 5000
First Name: Mohd Last Name: Yasee Salary: 3000
```

If you check your EMPLOYEE table, it should have the following records –

```
mysql> select * from EMPLOYEE;
+----+-----+-----+-----+
| id | first_name | last_name | salary |
+----+-----+-----+-----+
| 26 | Zara      | Ali      | 2000   |
| 27 | Daisy     | Das      | 5000   |
| 28 | John      | Paul     | 5000   |
| 29 | Mohd      | Yasee    | 3000   |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
mysql>
```

[< Previous Page](#)

[Next Page >](#)

Advertisements



[FAQ's](#) [Cookies Policy](#) [Contact](#)

© Copyright 2018. All Rights Reserved.