**Objectives**

* Demonstrate writing Hibernate Query Language and Native Query
  + HQL stands for Hibernate Query Language, JPQL stands for Java Persistence Query Language, Compare HQL and JPQL, @Query annotation, HQL fetch keyword, aggregate functions in HQL, Native Query, nativeQuery attribute
    - Reference - https://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html/ch11.html
    - Features of JPA Query - https://www.baeldung.com/spring-data-jpa-query

* Explain the need and benefit of Criteria Query
  + Scenarios where Criteria Query helps, CriteriaBuilder, Criteria Query, Root, TypedQuery
    - Reference - https://docs.oracle.com/javaee/6/tutorial/doc/gjrij.html

**Hands on 1**

**Introduction to HQL and JPQL** 

* HQL stands for Hibernate Query Language
* JPQL stands for Java Persistence Query Language
* Both HQL and JPQL are object focused query language similar to SQL
* JPQL is a subset of HQL
* All JPQL queries are valid HQL query, but the reverse is not true
* Both HQL and JPQL allows SELECT, UPDATE and DELETE
* HQL additionally allows INSERT statement

Reference - <https://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html/ch11.html>

**ANSWER**

// This is a theoretical Hands-On — nothing to implement.

// Summary:

// HQL is Hibernate's query language, supports INSERT.

// JPQL is JPA-compliant, supports SELECT, UPDATE, DELETE.

**Hands on 2**

**Get all permanent employees using HQL**   
  
Using HQL get all permanent employees. When retrieving the employee details it should also retrieve respective department and skill list as well.  
  
**HQL Solution**

* Include a new method definition in EmployeeRepository with @Query annotation

    @Query(value="SELECT e FROM Employee e WHERE e.permanent = 1")

    List<Employee> getAllPermanentEmployees();

  // NOTE: HQL looks like SQL, instead of table, Java classes and it's

  // instance variables are addressed here

* Include appropriate service method
* Include a new test method and Invoke the service method in OrmLearnApplication.java. Refer test method below that logs all employee details and each employee's skill details.

    public static void testGetAllPermanentEmployees() {

        LOGGER.info("Start");

        List<Employee> employees = employeeService.getAllPermanentEmployees();

        LOGGER.debug("Permanent Employees:{}", employees);

        employees.forEach(e -> LOGGER.debug("Skills:{}", e.getSkillList()));

        LOGGER.info("End");

    }

* Check the list of SQL queries executed in the log file. Following queries would have got executed.

select employee0\_.em\_id as em\_id1\_2\_0\_, department1\_.dp\_id as dp\_id1\_1\_1\_,

skill3\_.sk\_id as sk\_id1\_4\_2\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_0\_,

employee0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employee0\_.em\_name as em\_name3\_2\_0\_,

employee0\_.em\_permanent as em\_perma4\_2\_0\_,

employee0\_.em\_salary as em\_salar5\_2\_0\_, department1\_.dp\_name as dp\_name2\_1\_1\_,

skill3\_.sk\_name as sk\_name2\_4\_2\_, skilllist2\_.es\_em\_id as es\_em\_id1\_3\_0\_\_,

skilllist2\_.es\_sk\_id as es\_sk\_id2\_3\_0\_\_

from employee employee0\_ left outer join department department1\_ on

employee0\_.em\_dp\_id=department1\_.dp\_id left outer join employee\_skill skilllist2\_ on

employee0\_.em\_id=skilllist2\_.es\_em\_id left outer join skill skill3\_ on

skilllist2\_.es\_sk\_id=skill3\_.sk\_id

where employee0\_.em\_permanent=1

select employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employeeli0\_.em\_id as em\_id1\_2\_0\_,

employeeli0\_.em\_id as em\_id1\_2\_1\_, employeeli0\_.em\_date\_of\_birth as em\_date\_2\_2\_1\_,

employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_1\_, employeeli0\_.em\_name as em\_name3\_2\_1\_,

employeeli0\_.em\_permanent as em\_perma4\_2\_1\_, employeeli0\_.em\_salary as em\_salar5\_2\_1\_

from employee employeeli0\_ where employeeli0\_.em\_dp\_id=3

select employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employeeli0\_.em\_id as em\_id1\_2\_0\_,

employeeli0\_.em\_id as em\_id1\_2\_1\_, employeeli0\_.em\_date\_of\_birth as em\_date\_2\_2\_1\_,

employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_1\_, employeeli0\_.em\_name as em\_name3\_2\_1\_,

employeeli0\_.em\_permanent as em\_perma4\_2\_1\_, employeeli0\_.em\_salary as em\_salar5\_2\_1\_

from employee employeeli0\_ where employeeli0\_.em\_dp\_id=2

select skilllist0\_.es\_em\_id as es\_em\_id1\_3\_0\_, skilllist0\_.es\_sk\_id as es\_sk\_id2\_3\_0\_,

skill1\_.sk\_id as sk\_id1\_4\_1\_, skill1\_.sk\_name as sk\_name2\_4\_1\_

from employee\_skill skilllist0\_ inner join skill skill1\_

on skilllist0\_.es\_sk\_id=skill1\_.sk\_id

where skilllist0\_.es\_em\_id=2

**Optimizing HQL Solution by removing the EAGER fetch configuration**

* An optimal solution should not execute multiple queries, we have defined unnecessary fetch configuration in
* Eager fetch configuration is defined for employeeList in Department.java and skillList of Employee.java
* Remove these two eager fetch configurations and check the logs. The following queries would have got executed. It would have failed when getting the skill list. Since we have remove the eager fetch skill is not retrieved.

select employee0\_.em\_id as em\_id1\_2\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_,

employee0\_.em\_dp\_id as em\_dp\_id6\_2\_, employee0\_.em\_name as em\_name3\_2\_,

  employee0\_.em\_permanent as em\_perma4\_2\_, employee0\_.em\_salary as em\_salar5\_2\_

from employee employee0\_

where employee0\_.em\_permanent=1

select department0\_.dp\_id as dp\_id1\_1\_0\_, department0\_.dp\_name as dp\_name2\_1\_0\_

from department department0\_

where department0\_.dp\_id=2

select department0\_.dp\_id as dp\_id1\_1\_0\_, department0\_.dp\_name as dp\_name2\_1\_0\_

from department department0\_

where department0\_.dp\_id=3

* There are two issues in this approach:
  + We did not get the skill details
  + Still the query is not optimal as we have three queries executed

**Optimizing HQL solution by using 'fetch'**

* Change the query in EmployeeRepository.java as specified below:

@Query(value="SELECT e FROM Employee e left join e.department d left join e.skillList WHERE e.permanent = 1")

* The above query still fails to get skill details. Include fetch after each join. Wherever data is required we can include fetch, which will populate the respective data. Change the query as specified below:

@Query(value="SELECT e FROM Employee e left join fetch e.department d left join fetch e.skillList WHERE e.permanent = 1")

* Following the single query generated for the above HQL:

select employee0\_.em\_id as em\_id1\_2\_0\_, department1\_.dp\_id as dp\_id1\_1\_1\_,

skill3\_.sk\_id as sk\_id1\_4\_2\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_0\_,

employee0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employee0\_.em\_name as em\_name3\_2\_0\_,

  employee0\_.em\_permanent as em\_perma4\_2\_0\_, employee0\_.em\_salary as em\_salar5\_2\_0\_,

department1\_.dp\_name as dp\_name2\_1\_1\_, skill3\_.sk\_name as sk\_name2\_4\_2\_,

skilllist2\_.es\_em\_id as es\_em\_id1\_3\_0\_\_, skilllist2\_.es\_sk\_id as es\_sk\_id2\_3\_0\_\_

from employee employee0\_ left outer join department department1\_ on

  employee0\_.em\_dp\_id=department1\_.dp\_id left outer join employee\_skill skilllist2\_ on

  employee0\_.em\_id=skilllist2\_.es\_em\_id left outer join skill skill3\_ on

  skilllist2\_.es\_sk\_id=skill3\_.sk\_id

where employee0\_.em\_permanent=1

**IMPORTANT TAKEAWAY:** Join keyword links the table, but does not populate the beans. Fetch ensures that the beans are populated. Based on our need wherever we need data, we can define fetch. When joining table data is not needed the fetch can be ignored.

**ANSWER**

// EmployeeRepository.java

@Query("SELECT e FROM Employee e LEFT JOIN FETCH e.department d LEFT JOIN FETCH e.skillList WHERE e.permanent = true")

List<Employee> getAllPermanentEmployees();

// EmployeeService.java

@Transactional

public List<Employee> getAllPermanentEmployees() {

return employeeRepository.getAllPermanentEmployees();

}

// OrmLearnApplication.java

private static void testGetAllPermanentEmployees() {

LOGGER.info("Start");

List<Employee> employees = employeeService.getAllPermanentEmployees();

employees.forEach(e -> {

LOGGER.debug("Employee: {}", e);

LOGGER.debug("Department: {}", e.getDepartment());

LOGGER.debug("Skills: {}", e.getSkillList());

});

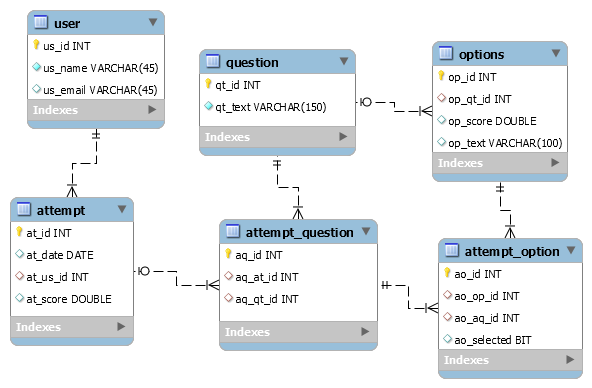
LOGGER.info("End");

}

**Hands on 3**

**Fetch quiz attempt details using HQL**   
  
In a quiz application there is a requirement for admin to view details of a quiz that an user had attempted. This view should include the following details:

* Username
* Attempted Date
* All questions as part of the attempt
* List of options under each quiz
* The option that is correct answer
* The score for correct answer

**Schema Diagram**  
   
Notes on Schema:

* Tables user, question and options are self explanatory. They hold the respective master data.
* Tables attempt, attempt\_question and attempt\_option are used to hold the data of attempts made by each user.

**Follow steps below to setup the schema:**

* Go to spring-data-jpa-files folder in windows explorer
* Open file quiz.mwb in MySQL Workbench
* Generate SQL file using File > Export > Forward Engineer SQL CREATE Script
* Click Browse and select the file name and folder for the saving the generated SQL file
* Select the check box "Generate INSERT Statements for Tables"
* Click Next > Next > Finish to generate the SQL file
* Execute the SQL file in ormlearn schema and check the data in the tables

**Steps to get this implemented:**

* Create necessary entity class for each table defined above
* Define necessary O/R mapping based on the schema defined above​​​
* Create a Repository and Service class:
  + AttemptRepository
    - public Attempt getAttempt(int userId, int attemptId)
  + AttemptService
    - public Attempt getAttempt(int userId, int attemptId)
* Modify OrmLearnApplication.java to include a new test method and test AttemptService.getAttemptDetail() method
* Create HQL that joins the tables in the below order:
  + user
  + attempt
  + attempt\_question
  + question
  + attempt\_option
  + options
* In the HQL include where class for userId and attemptId
* Include 'fetch' in HQL wherever there is one-to-many or many-to-many relationship
* In OrmLearnApplication.java test method ​​​​​​get the attempts details, iterate through the details and display the data in the following format. The second column in each option denotes the score from question table. The last column in each option denotes the answer selected by the user.

What is the extension of the hyper text markup language file?

1) .xhtm       0.0     false

2) .ht         0.0     false

3) .html       1.0     true

4) .htmx       0.0     false

What is the maximum level of heading tag can be used in a HTML page?

1) 5           0.0   false

2) 3           0.0   true

3) 4         0.0   false

4) 6   1.0   false

The HTML document itself begins with <html> and ends </html>. State True of False

1) false        0.0   false

2) true       1.0   true

Choose the right option to store text value value in a variable

1) 'John'     0.5    true

2) John       0.0    false

3) "John" 0.5    false

4) /John/ 0.0    false

**ANSWER**

@Query("SELECT a FROM Attempt a "

+ "JOIN FETCH a.user u "

+ "JOIN FETCH a.attemptQuestionList aq "

+ "JOIN FETCH aq.question q "

+ "JOIN FETCH aq.attemptOptionList ao "

+ "JOIN FETCH ao.option o "

+ "WHERE a.id = :attemptId AND a.user.id = :userId")

Attempt getAttemptDetails(@Param("userId") int userId, @Param("attemptId") int attemptId);

// AttemptService.java

@Transactional

public Attempt getAttemptDetails(int userId, int attemptId) {

return attemptRepository.getAttemptDetails(userId, attemptId);

}

// OrmLearnApplication.java

private static void testGetAttemptDetails() {

Attempt attempt = attemptService.getAttemptDetails(1, 1); // example

LOGGER.info("User: {}", attempt.getUser().getName());

for (AttemptQuestion aq : attempt.getAttemptQuestionList()) {

LOGGER.info(aq.getQuestion().getText());

for (AttemptOption ao : aq.getAttemptOptionList()) {

Option o = ao.getOption();

LOGGER.info(" {}: {} {} {}",

o.getId(), o.getText(), o.getScore(), ao.isSelected());

}

}

}

**Hands on 4**

**Get average salary using HQL**   
  
Compute the average salary of a department using HQL.  
  
Refer steps below to implement:

* Define HQL in EmployeeRepository

    @Query(value="SELECT AVG(e.salary) FROM Employee e")

    double getAverageSalary();

* Include new method with above signature in EmployeeService and include test method in OrmLearnApplication
* The above query does not filter the result based on department id. Modify the query and method signature as specified below to accept department.

@Query(value="SELECT AVG(e.salary) FROM Employee e where e.department.id = :id")

double getAverageSalary(@Param("id") int id);

**NOTES:**

* Observe how department id is referred from 'e'
* Make note of the colon (:) used to define a parameter within a query
* @Param annotation helps in binding the input department id with the query parameter
* Similar to AVG(), all other aggregate functions can be used

**ANSWER**

// EmployeeRepository.java

@Query("SELECT AVG(e.salary) FROM Employee e WHERE e.department.id = :id")

double getAverageSalary(@Param("id") int deptId);

// EmployeeService.java

@Transactional

public double getAverageSalary(int deptId) {

return employeeRepository.getAverageSalary(deptId);

}

// OrmLearnApplication.java

private static void testAverageSalary() {

double avg = employeeService.getAverageSalary(2); // departmentId

LOGGER.info("Average Salary = {}", avg);

}

**Hands on 5**

**Get all employees using Native Query**   
  
About Native Queries

* Native queries are direct SQL queries to the database instead of using HQL
* Try to avoid Native Queries and make it minimal.
* Avoiding native queries helps in easier portability of database

Follow steps below to implement

* Define a new native query method in EmployeeRepository

    @Query(value="SELECT \* FROM employee", nativeQuery = true)

    List<Employee> getAllEmployeesNative();

* Define relevant method in service and OrmLearnApplication and test it

**ANSWER**

// EmployeeRepository.java

@Query(value = "SELECT \* FROM employee", nativeQuery = true)

List<Employee> getAllEmployeesNative();

// EmployeeService.java

@Transactional

public List<Employee> getAllEmployeesNative() {

return employeeRepository.getAllEmployeesNative();

}

// OrmLearnApplication.java

private static void testNativeQueryEmployees() {

List<Employee> list = employeeService.getAllEmployeesNative();

list.forEach(e -> LOGGER.debug("Employee: {}", e));

}

**Hands on 6**

**Criteria Query**   
  
Find below an online retail user scenario

* User goes to Amazon
* Searches with keyword "laptop"
* The left hand size contains the following filter criteria categories:
  + Customer review
  + Hard Disk Size
  + RAM Size
  + CPU Speed
  + Operating System
  + Weight
  + CPU
* The user might select options available in one or more of the criteria and try a fresh search.

In the above given scenario, what will be the where clause of the HQL query that you will run on the product?  
  
The where clause varies based on the criteria selected by the user. We have to dynamically frame the where clause filters based on the criteria selected by user.  
  
Criteria Query helps in handling this scenario in a better way. The filter criteria can be programmatically added, rather than fixing the HQL Statement.  
  
Go through the examples in the link below to understand how Criteria Query has to be implemented.  
<https://howtodoinjava.com/hibernate/hibernate-criteria-queries-tutorial/>

**ANSWER**

// ProductService.java (example filter: brand, RAM, CPU)

@Transactional

public List<Product> searchProducts(String brand, Integer ram, String cpu) {

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

CriteriaQuery<Product> cq = cb.createQuery(Product.class);

Root<Product> product = cq.from(Product.class);

List<Predicate> predicates = new ArrayList<>();

if (brand != null) {

predicates.add(cb.equal(product.get("brand"), brand));

}

if (ram != null) {

predicates.add(cb.equal(product.get("ram"), ram));

}

if (cpu != null) {

predicates.add(cb.equal(product.get("cpu"), cpu));

}

cq.select(product).where(cb.and(predicates.toArray(new Predicate[0])));

return entityManager.createQuery(cq).getResultList();

}

// OrmLearnApplication.java

private static void testCriteriaQuery() {

List<Product> products = productService.searchProducts("Dell", 16, null);

products.forEach(p -> LOGGER.debug("Product: {}", p));

}