

Session 6

Hibernate and JPA

Hibernate maps entity classes to tables so that persistence code is annotations and a session or entity manager call rather than hand-written SQL. This session persists the student admission data and performs a batch update.

The `student-admission` project gets a proper schema: five entities, a fresh `student_admission` database, full CRUD on students and a batch approval that assigns enrolment numbers in one transaction.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 25 to 28 of the manual: hibernate and jpa
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✗ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q25	Create a database for the student admission lifecycle and configure Hibernate and JPA...	Complete
Q26	Retrieve Student information using Hibernate and printing the value in the console	Complete
Q27	Create CRUD (Create/Save, Read/Fetch, Edit/Update, Delete) using Spring MVC and...	Complete
Q28	Apply batch update for student's admission approval using Spring MVC and Hibernate	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- Draw the admission life-cycle tables first: Student, Programme, Course, StudentCourse, AdmissionStatus. Map each with `@Entity`, `@Id`, `@OneToMany`, `@ManyToMany`.
- Configure `hibernate.cfg.xml` or Spring's `LocalSessionFactoryBean` with the MySQL dialect and `hbm2ddl.auto=update` for the lab.
- Batch update: loop with `session.update` and flush every 20 records, inside one transaction.

Question 25

Problem Statement

■ Write in lab record

Create a database for the student admission lifecycle and configure Hibernate and JPA with the Spring MVC Project along with all table entities.

Solution

■ Write in lab record

Life cycle: a student applies to one programme and picks courses (status `PENDING`); the office approves or rejects; every change is recorded. Tables and relations:

Table	Key columns	Relation
programme	id, code, name	one programme has many courses
course	id, code, title, programme_id	many courses belong to one programme
student	id, enrolment_no, name, email, mobile, dob, address, hostel_required, programme_id, status, applied_on	many students apply to one programme
student_course	id, student_id, course_id, enrolled_on	join entity between student and course (many-to-many with a date)
admission_status	id, student_id, status, changed_on, remark	one student has many status changes, oldest first

Steps

1. Run `student_admission.sql` in MySQL (Workbench or `mysql -u root -p < student_admission.sql`). It creates the database, five tables and seeds two programmes and six courses.
2. Point `db.properties` at `student_admission`.
3. Replace `JpaConfig.java` (batch size and validation mode are new) and confirm `pom.xml` matches the final listing below.
4. In `com.ignou.lab.admission.entity` replace `Student.java` and add `Programme.java`, `Course.java`, `StudentCourse.java`, `AdmissionStatus.java`.
5. Delete Session 5's `AdmissionController.java`, `admission-form.jsp` and `admission-result.jsp`; Q27 replaces them. Rebuild only after Q27's files are in place, because `AdmissionForm` changes shape.

Program

- Lab record: every tab is one file of the answer. Write all of them.

student_admission.sql

student_admission.sql

```
1 -- Student admission life cycle: one programme has many courses; a student
2 -- applies to one programme, picks courses, and moves PENDING → APPROVED/RE
3 CREATE DATABASE IF NOT EXISTS student_admission;
4 USE student_admission;
5
6 CREATE TABLE programme (
7     id      BIGINT AUTO_INCREMENT PRIMARY KEY,
8     code    VARCHAR(10) NOT NULL UNIQUE,
9     name    VARCHAR(80) NOT NULL
10 );
11
12 CREATE TABLE course (
13     id          BIGINT AUTO_INCREMENT PRIMARY KEY,
14     code        VARCHAR(10) NOT NULL UNIQUE,
15     title       VARCHAR(120) NOT NULL,
16     programme_id BIGINT NOT NULL,
17     FOREIGN KEY (programme_id) REFERENCES programme(id)
18 );
19
20 CREATE TABLE student (
21     id          BIGINT AUTO_INCREMENT PRIMARY KEY,
22     enrolment_no VARCHAR(12) UNIQUE,
23     name        VARCHAR(80) NOT NULL,
24     email       VARCHAR(120) NOT NULL,
25     mobile      VARCHAR(10) NOT NULL,
26     dob         DATE NOT NULL,
27     address     VARCHAR(255) NOT NULL,
28     hostel_required BIT(1) NOT NULL,
29     programme_id BIGINT NOT NULL,
30     status      ENUM('PENDING','APPROVED','REJECTED') NOT NULL DEFAULT 'PE
31     applied_on  DATE NOT NULL,
32     FOREIGN KEY (programme_id) REFERENCES programme(id)
33 );
34
35 -- join table with its own data (when the course was taken), so it is an ent
36 CREATE TABLE student_course (
37     id          BIGINT AUTO_INCREMENT PRIMARY KEY,
38     student_id  BIGINT NOT NULL,
39     course_id   BIGINT NOT NULL,
40     enrolled_on DATE NOT NULL,
41     UNIQUE (student_id, course_id),
42     FOREIGN KEY (student_id) REFERENCES student(id) ON DELETE CASCADE,
43     FOREIGN KEY (course_id) REFERENCES course(id)
```

```

44 );
45
46 -- every status change of every application, oldest first
47 CREATE TABLE admission_status (
48     id          BIGINT AUTO_INCREMENT PRIMARY KEY,
49     student_id  BIGINT NOT NULL,
50     status      ENUM('PENDING','APPROVED','REJECTED') NOT NULL,
51     changed_on  DATETIME(6) NOT NULL,
52     remark      VARCHAR(255),
53     FOREIGN KEY (student_id) REFERENCES student(id) ON DELETE CASCADE
54 );
55
56 INSERT INTO programme (code, name) VALUES
57     ('MCA', 'Master of Computer Applications'),
58     ('BCA', 'Bachelor of Computer Applications');
59
60 INSERT INTO course (code, title, programme_id) VALUES
61     ('MCS-218', 'Data Communication and Computer Networks', 1),
62     ('MCS-219', 'Object Oriented Analysis and Design',      1),
63     ('MCS-220', 'Web Technologies',                          1),
64     ('MCS-221', 'Data Warehousing and Data Mining',         1),
65     ('BCS-011', 'Computer Basics and PC Software',         2),
66     ('BCS-012', 'Basic Mathematics',                        2);

```

Output

Expected, checked by reading, not executed. `SHOW TABLES` in MySQL lists `admission_status`, `course`, `programme`, `student`, `student_course`. On deployment Tomcat's log shows Hibernate starting with the five entities and, because `hbm2ddl.auto=update` finds every table already there, no `create table` statements:

```

1 HHH000412: Hibernate ORM core version 6.6.3.Final
2 HHH10001005: Database info: Database JDBC URL [jdbc:mysql://localhost:3306/s

```

Explanation

- `@Entity` plus `@Table` name the table; `@Id` with `IDENTITY` uses MySQL auto-increment; `@Column` sets name, length and nullability so that generated DDL and the script agree.
- `@ManyToOne` with `@JoinColumn` owns the foreign key (`Course.programme` , `Student.programme` , both sides of `StudentCourse`). `@OneToMany(mappedBy = ...)` is the inverse, read-only view of

the same key.

- Student to Course is many-to-many, but the join row has its own data (`enrolled_on`), so it is modelled as the entity `StudentCourse` rather than `@ManyToMany`. `cascade = ALL, orphanRemoval = true` on `Student.courses` means saving or deleting a student saves or deletes its rows.
- `AdmissionStatus` is the life cycle: one row per change, ordered by `@OrderBy("changedOn")`. `Student.status` keeps the current stage as a column for cheap filtering; `changeStatus()` is the only method that writes it, so column and history never disagree. `@Enumerated(String)` stores `PENDING`, not `0`.
- `JpaConfig` is the Hibernate configuration: data source, entity package, provider, and the `hibernate.*` properties (`hbm2ddl.auto`, `show_sql`, `jdbc.batch_size` for Q28). Hibernate 6 detects the MySQL dialect from the connection, so no `dialect` property is needed. `validation.mode=none` stops Hibernate re-running the Bean Validation the controller already ran.

Question 26

Problem Statement

■ Write in lab record

Retrieve Student information using Hibernate and printing the value in the console.

Solution

■ Write in lab record

Steps

1. Add `ConsoleApp.java` in `com.ignou.lab.admission`. It boots only `JpaConfig`, so no Tomcat is involved.
2. Insert at least two students first (Q27's form, or two `INSERT` statements in MySQL).
3. Run: `mvn -q compile exec:java -Dexec.mainClass=com.ignou.lab.admission.ConsoleApp`, or Run As → Java Application.

Program

ConsoleApp.java

```
1 package com.ignou.lab.admission;
2
3 import java.util.List;
4 import java.util.stream.Collectors;
5 import jakarta.persistence.EntityManagerFactory;
6 import org.hibernate.Session;
7 import org.hibernate.SessionFactory;
8 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
9 import com.ignou.lab.admission.entity.Student;
10
11 /**
12  * Q26: read students with the Hibernate Session API and print them on the c
13  * Run: mvn -q compile exec:java -Dexec.mainClass=com.ignou.lab.admission.Co
14  */
15 public class ConsoleApp {
16
17     public static void main(String[] args) {
18         try (var ctx = new AnnotationConfigApplicationContext(JpaConfig.class);
19             SessionFactory sessionFactory = ctx.getBean(EntityManagerFactory.class);
20             try (Session session = sessionFactory.openSession()) {
21             List<Student> students = session.createQuery(
22                 "select distinct s from Student s join fetch s.programme p"
23                 + " left join fetch s.courses sc left join fetch sc."
24                 + Student.class).getResultList();
25
26             System.out.printf("%-4s %-11s %-18s %-5s %-9s %s\n",
27                 "ID", "ENROLMENT", "NAME", "PROG", "STATUS", "COURSE");
28             for (Student s : students) {
29                 String courses = s.getCourses().stream()
30                     .map(sc -> sc.getCourse().getCode())
31                     .collect(Collectors.joining(", "));
32                 System.out.printf("%-4d %-11s %-18s %-5s %-9s %s\n",
33                     s.getId(), s.getEnrolmentNo() == null ? "-" : s.getEnrolmentNo(),
34                     s.getName(), s.getProgramme().getCode(), s.getStatus());
35             }
36             System.out.println(students.size() + " student(s)");
37         }
38     }
39 }
40 }
```

Output

Expected console output after two applications and one approval, checked by reading, not executed. The `Hibernate:` block with the `SELECT` (one query with three joins) prints first because `show_sql` is on, then:

```
1 ID      ENROLMENT      NAME                PROG  STATUS  COURSES
2 1        20260000001    Asha Verma          MCA   APPROVED MCS-218,MCS-220
3 2        -              Rahul Singh         MCA   PENDING  MCS-219
4 2 student(s)
```

Explanation

- The Spring-managed `EntityManagerFactory` is Hibernate underneath; `unwrap(SessionFactory.class)` exposes the native API, and `openSession()` gives a Hibernate `Session`.
- The query is HQL. `join fetch s.programme` and `left join fetch s.courses sc left join fetch sc.course` load the related rows in the same `SELECT`; without them each `getProgramme()` or `getCourses()` would fire another query (the N+1 problem) or fail with `LazyInitializationException` once the session is closed.
- `select distinct` collapses the duplicate `Student` rows the join produces (one per course).
- `printf` with fixed widths makes the console a table; enrolment shows `-` while it is null, which is every pending application.

Question 27

Problem Statement

■ Write in lab record

Create CRUD (Create/Save, Read/Fetch, Edit/Update, Delete) using Spring MVC and Hibernate.

Solution

■ Write in lab record

Steps

1. Replace `StudentRepository.java` (all persistence), `AdmissionForm.java` (now carries `id`, `programmeId`, `courseIds`) and `StudentController.java` (all URLs under `/students`).
2. Replace `students.jsp` and add `student-form.jsp` in `WEB-INF/views`. `head.jspf` from Session 5 stays.
3. In `home.jsp` change the "Apply for admission" link to `/students/new`.
4. Rebuild, redeploy and walk the cycle: `/students` (empty) → New application → Save → row appears → Edit → change the mobile, tick another course → Save → Delete → confirm → row gone. Watch Tomcat's console for the SQL at each step.

Operation	URL and method	Repository call	SQL Hibernate sends
Create	GET <code>/students/new</code> , POST <code>/students/save</code>	<code>save(form)</code> with <code>id</code> null	INSERT student, INSERT student_course per course, INSERT admission_status
Read	GET <code>/students</code>	<code>findAll()</code>	one SELECT with joins
Update	GET <code>/students/7/edit</code> , POST <code>/students/save</code>	<code>save(form)</code> with <code>id</code> 7	UPDATE student, DELETE and INSERT student_course as needed
Delete	POST <code>/students/7/delete</code>	<code>delete(7)</code>	DELETE admission_status, DELETE student_course, DELETE student

Program

- Lab record: every tab is one file of the answer. Write all of them.

StudentRepository.java

repo/StudentRepository.java

```
1 package com.ignou.lab.admission.repo;
2
3 import java.time.Year;
4 import java.util.List;
5 import java.util.NoSuchElementException;
6 import jakarta.persistence.EntityManager;
7 import jakarta.persistence.PersistenceContext;
8 import org.springframework.stereotype.Repository;
9 import org.springframework.transaction.annotation.Transactional;
10 import com.ignou.lab.admission.entity.AdmissionStatus.Status;
11 import com.ignou.lab.admission.entity.Course;
12 import com.ignou.lab.admission.entity.Programme;
13 import com.ignou.lab.admission.entity.Student;
14 import com.ignou.lab.admission.entity.StudentCourse;
15 import com.ignou.lab.admission.web.AdmissionForm;
16
17 /** Every public method runs in one transaction; commit happens when it returns
18  * @Repository
19  * @Transactional
20  */
21 public class StudentRepository {
22
23     @PersistenceContext
24     private EntityManager em;
25
26     // ---- lookups for the form ----
27
28     public List<Programme> programmes() {
29         return em.createQuery("from Programme p order by p.code", Programme.class);
30     }
31
32     public List<Course> courses() {
33         return em.createQuery("from Course c order by c.code", Course.class);
34     }
35
36     // ---- Read ----
37
38     @Transactional(readOnly = true)
39     public List<Student> findAll() {
40         // join fetch: programme and courses are loaded now, so the JSP never
41         // needs to do a second query
42         return em.createQuery(
43             "select distinct s from Student s join fetch s.programme"
44             + " left join fetch s.courses sc left join fetch sc.course o"
45             + " Student.class).getResultList();
```

```
44     }
45
46     public Student find(Long id) {
47         Student s = em.find(Student.class, id);
48         if (s == null) {
49             throw new NoSuchElementException("No student with id " + id); //
50         }
51         return s;
52     }
53
54     /** the edit page needs the form filled from a managed entity (courses a
55     @Transactional(readOnly = true)
56     public AdmissionForm formFor(Long id) {
57         return AdmissionForm.from(find(id));
58     }
59
60     // ---- Create / Update ----
61
62     public Student save(AdmissionForm f) {
63         Student s = f.getId() == null ? new Student() : find(f.getId());
64         s.setName(f.getName());
65         s.setEmail(f.getEmail());
66         s.setMobile(f.getMobile());
67         s.setDob(f.getDob());
68         s.setAddress(f.getAddress());
69         s.setHostelRequired(f.getHostelRequired());
70         s.setProgramme(em.getReference(Programme.class, f.getProgrammeId()))
71         s.getCourses().clear();
72         for (Long courseId : f.getCourseIds()) {
73             s.getCourses().add(new StudentCourse(s, em.getReference(Course.c
74         }
75         if (s.getId() == null) {
76             s.changeStatus(Status.PENDING, "Application received");
77             em.persist(s);
78         }
79         return s;
80     }
81
82     // ---- Delete ----
83
84     public void delete(Long id) {
85         em.remove(find(id)); // cascades to student_course and admission_sta
86     }
87
```

```

88      // ---- Q28: batch approval ----
89
90      /** Approves every pending id in ONE transaction; UPDATES go to MySQL in
91      public int approve(List<Long> ids) {
92          int approved = 0;
93          for (Long id : ids) {
94              Student s = find(id);
95              if (s.getStatus() != Status.PENDING) {
96                  continue;
97              }
98              s.setEnrolmentNo(String.format("%d%06d", Year.now().getValue(),
99              s.changeStatus(Status.APPROVED, "Approved in batch");
100
101              if (++approved % 20 == 0) {
102
103                  em.flush(); // push this batch of UPDATE/INSERT statements
104
105                  em.clear(); // drop them from memory before loading the next
106
107              }
108          }
109
110          return approved; // commit flushes whatever is left
111      }
112  }

```

Output

Expected, checked by reading, not executed. After saving a new application the list page shows a green alert `Saved Asha Verma (id 1)` and a row:

1	[]	1	-	Asha Verma	MCA	MCS-218, MCS-220	PENDING	2026-09-26	Edit Delete
---	-----	---	---	------------	-----	------------------	---------	------------	-------------

Tomcat's console for that save:

```

1 Hibernate: insert into student (address,applied_on,dob,email,enrolment_no,ho
2 Hibernate: insert into student_course (course_id,enrolled_on,student_id) val
3 Hibernate: insert into student_course (course_id,enrolled_on,student_id) val
4 Hibernate: insert into admission_status (changed_on,remark,status,student_id

```

Editing and saving with one course removed prints one `update student ...` and one `delete from student_course where id=?`. Delete prints the three DELETES in child-first order.

Explanation

- Create and update share one method. `save(form)` either makes a `new Student()` or loads the existing one with `em.find`, copies the fields, and rebuilds the course list. A loaded entity is managed: Hibernate compares it with the snapshot at commit and issues an UPDATE only for what changed (dirty checking). `persist` is called only for a new object.
- `em.getReference(Programme.class, id)` returns a proxy holding just the id; setting it writes the foreign key without a SELECT.
- Clearing `s.getCourses()` and adding new `StudentCourse` objects is enough: `orphanRemoval` deletes rows no longer in the list, `cascade` inserts the new ones.
- `formFor(id)` converts entity to form inside the transaction because `courses` is lazy; converting in the controller would hit a closed session.
- The controller follows POST-redirect-GET: after a POST it redirects to `/students`, so a browser refresh does not save twice; `RedirectAttributes.addFlashAttribute` carries the message across the redirect.
- Delete is a POST button, not a link, so a crawler or a prefetching browser cannot delete rows. The Delete buttons use the HTML `form` attribute to point at small forms outside the approval form, because forms cannot nest.

Question 28

Problem Statement

■ Write in lab record

Apply batch update for student's admission approval using Spring MVC and Hibernate.

Solution

■ Write in lab record

Steps

1. `StudentRepository.approve()` and `StudentController.approve()` are in the Q27 listings; `students.jsp` below adds a checkbox to every `PENDING` row and an "Approve selected" button posting to `/students/approve`.
2. `JpaConfig` (Q25) sets `hibernate.jdbc.batch_size=20` and `hibernate.order_updates=true`.
3. Create five applications, tick three, press Approve selected. The three rows turn green with enrolment numbers; the alert reads `3 application(s) approved in one transaction`.
4. Prove it is one transaction: temporarily change `%06d` in `approve()` to `%s` with a text longer than 12 characters so the third UPDATE fails on column length; after the exception none of the three is approved.

The method under test, from `StudentRepository.java` :

JAVA

```
1 public int approve(List<Long> ids) {
2     int approved = 0;
3     for (Long id : ids) {
4         Student s = find(id);
5         if (s.getStatus() != Status.PENDING) {
6             continue;
7         }
8         s.setEnrolmentNo(String.format("%d%06d", Year.now().getValue(), id));
9         s.changeStatus(Status.APPROVED, "Approved in batch");
10        if (++approved % 20 == 0) {
11            em.flush();
12            em.clear();
13        }
14    }
15    return approved;
16 }
```

Program

WEB-INF/views/students.jsp

```

1 <%@ page contentType="text/html; charset=UTF-8" %>
2 <%@ taglib prefix="c" uri="jakarta.tags.core" %>
3 <!DOCTYPE html>
4 <html lang="en">
5 <head>
6 <%@ include file="head.jspf" %>
7 <title>Students</title>
8 </head>
9 <body>
10 <nav class="navbar navbar-dark"><div class="container"><span class="navbar-b
11 <div class="container mt-4">
12 <c:if test="${not empty message}">
13 <div class="alert alert-success">${message}</div>
14 </c:if>
15
16 <div class="d-flex justify-content-between align-items-center mb-3">
17 <h1 class="h3 m-0">Applications</h1>
18 <a class="btn btn-primary" href="${pageContext.request.contextPath}/stud
19 </div>
20
21 <!-- one form around the table: the ticked ids go to /students/approve -->
22 <form method="post" action="${pageContext.request.contextPath}/students/ap
23 <table class="table table-striped align-middle">
24 <thead>
25 <tr><th></th><th>Id</th><th>Enrolment</th><th>Name</th><th>Programme
26 </thead>
27 <tbody>
28 <c:forEach items="${students}" var="s">
29 <tr>
30 <td><c:if test="${s.status == 'PENDING'}"><input class="form-che
31 <td>${s.id}</td>
32 <td>${empty s.enrolmentNo ? '-' : s.enrolmentNo}</td>
33 <td><c:out value="${s.name}" /></td>
34 <td>${s.programme.code}</td>
35 <td><c:forEach items="${s.courses}" var="sc" varStatus="st">${sc
36 <td><span class="badge ${s.status == 'APPROVED' ? 'text-bg-succe
37 <td>${s.appliedOn}</td>
38 <td class="text-nowrap">
39 <a class="btn btn-sm btn-outline-secondary" href="${pageContex
40 <!-- the form attribute links this button to a form outside th
41 <button class="btn btn-sm btn-outline-danger" type="submit" fo
42 <button onclick="return confirm('Delete student ${s.id}?')">De
43 </td>

```

```

44         </tr>
45     </c:forEach>
46 </tbody>
47 </table>
48     <button type="submit" class="btn btn-success">Approve selected</button>
49 </form>
50
51 <c:forEach items="${students}" var="s">
52     <form id="delete-${s.id}" method="post" action="${pageContext.request.co
53 </c:forEach>
54 </div>
55 </body>
56 </html>

```

Output

Expected, checked by reading, not executed. After approving ids 1, 2 and 4 the list shows:

1	1	2026000001	Asha Verma	MCA	MCS-218, MCS-220	APPROVED	2026-09-2	
2	2	2026000002	Rahul Singh	MCA	MCS-219	APPROVED	2026-09-2	
3	[]	3	-	Priya Nair	BCA	BCS-011	PENDING	2026-09-2
4	4	2026000004	Amit Kumar	MCA	MCS-220, MCS-221	APPROVED	2026-09-2	
5	[]	5	-	Sunita Devi	BCA	BCS-012	PENDING	2026-09-2

Tomcat's console prints the SELECT for each `find`, then at commit three `update student set ... status=? where id=?` statements followed by three `insert into admission_status` statements, grouped because of `order_updates`. `SELECT * FROM admission_status` in MySQL shows two rows for each approved student: `PENDING` from the application and `APPROVED` from this batch.

Explanation

- The browser sends the ticked ids as repeated `ids=1&ids=2&ids=4`; `@RequestParam List<Long> ids` collects them. Untick everything and `ids` is absent, so it is `required = false` with a null check.
- `approve()` is one `@Transactional` method, so the whole batch is one transaction: all rows are approved or, if any UPDATE fails, the rollback leaves every row `PENDING`.
- Inside the loop nothing is sent to MySQL; each managed `Student` is marked dirty. At flush time Hibernate emits the UPDATES, and with `hibernate.jdbc.batch_size=20` the JDBC driver

sends them in groups of 20 statements per round trip instead of one each.

- `flush()` then `clear()` every 20 rows bounds memory: the persistence context does not hold thousands of entities for a long batch. For a lab-sized list the loop ends before the first flush and the commit does it all.
- The status guard skips rows that are already approved or rejected, so re-submitting the same ids is harmless and returns a count of 0.
- The enrolment number is year plus zero-padded id, unique because the id is; a real registry would take the next value from a sequence table inside the same transaction.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** Why is `StudentCourse` an entity instead of `@ManyToMany`? **A:** The join row carries `enrolled_on`; `@ManyToMany` cannot hold extra columns.
- **Q:** What does `orphanRemoval = true` do? **A:** When a child is removed from the collection of a managed parent, Hibernate deletes its row at flush.
- **Q:** Managed, detached, transient: what are they? **A:** Transient: new object, no row. Managed: loaded or persisted inside an open persistence context; changes are tracked. Detached: was managed, the context closed; changes are not tracked.
- **Q:** What is `LazyInitializationException`? **A:** Touching a lazy collection after the session closed. Fix with `join fetch` or by doing the access inside the transaction.
- **Q:** What is the N+1 problem? **A:** One query for the list and one more per row for a relation; `join fetch` turns it into one query.
- **Q:** How does the batch approval stay atomic? **A:** One `@Transactional` method; commit at the end or rollback on exception, nothing in between is visible.
- **Q:** Why `flush` and `clear` every 20? **A:** `flush` sends the pending statements, `clear` frees the managed entities; together they keep memory flat on large batches.
- **Q:** Where is the Hibernate dialect configured? **A:** Nowhere; Hibernate 6 reads the database metadata from the connection and picks `MySQLDialect` itself.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Accessing `student.courses` in the JSP after a `findAll` without `join fetch`, then reporting a `LazyInitializationException` as a Hibernate bug.
- Calling `em.merge` on a detached entity for every update instead of loading and changing the managed one; the collection handling gets unpredictable.
- Putting `@Transactional` on the controller instead of the repository, which drags the JSP rendering into the transaction.
- Deleting through a GET link; browsers prefetch links and delete rows nobody clicked.
- Setting `hbm2ddl.auto=create` in the lab and losing every row on each redeploy.
- Forgetting `distinct` in the fetch-join query and showing each student once per course.

Session Summary

■ Write in lab record

- Question 25: `student_admission` schema (five tables), entities `Programme`, `Course`, `Student`, `StudentCourse`, `AdmissionStatus`, Hibernate configured in `JpaConfig`
- Question 26: `ConsoleApp` reading students through the Hibernate `Session` and HQL fetch joins, printed as a console table
- Question 27: `StudentRepository` and `StudentController` with list, new, edit, save and delete, views `students.jsp` and `student-form.jsp`
- Question 28: batch approval of ticked applications in one transaction with JDBC batching, enrolment numbers assigned, history rows written

[✎ Edit this page](#)

[< Previous](#)
Session 5

Session 7 [Next >](#)