

Week 4 Lab: Joins, Procedures, Transactions, Triggers and Indexes

Time: about 50 minutes · Environment: MySQL in Uniform Server

Before you start

1. Start Uniform Server and MySQL. Open the MySQL console (or phpMyAdmin).
2. Run `week4_lab_setup.sql` once. In the console:
`SOURCE C:/your/path/week4_lab_setup.sql;`
3. Open `week4_lab_tasks.sql` in a text editor. Write each answer under its task, then run it.
4. Work through the parts **in order**. Parts D and E change the data. If things get confused, re-run the setup file and repeat the part.
5. Every task uses something from the lecture. Each part names the slides to look back at.
6. phpMyAdmin users: if `DELIMITER` gives an error, remove the `DELIMITER` lines and set the *Delimiter* box under the SQL editor to `//`.

The database

A small sports centre runs weekly classes. Members book classes and pay from prepaid credit.

Table	Columns (PK underlined, FK in italics)
Instructors	<u>instructor_id</u> , name, specialism
Classes	<u>class_id</u> , class_name, <i>instructor_id</i> (may be NULL), day_of_week, capacity, price
Members	<u>member_id</u> , first_name, last_name, email (unique), credit, joined_on, last_updated
Bookings	<u>booking_id</u> , <i>member_id</i> , <i>class_id</i> (one booking per member per class)
Rooms	<u>room_id</u> , room_name
TimeSlots	<u>slot_id</u> , start_time

Start by running `SELECT *` on Instructors, Classes, Members and Bookings, and get to know the data. Some rows have no match on purpose. Spotting them is part of the lab.

Part A Cartesian product

slides 28–35 · 5 min

A1 The centre wants a blank timetable grid: every room paired with every time slot. Predict the number of rows, then write the query using the comma syntax from slide 35.

Check: 9 rows.

A2 Write the same query using `CROSS JOIN`.

A3 Predict, then run, the Cartesian product of Instructors and Classes. Why is this result not useful on its own?

Part B Inner joins

slides 8–13, 45 · 8 min

B1 List every booking, showing the member's first name, last name and the class name. Bookings is the bridge table, like Enrollments on slide 8.

Check: 8 rows.

B2 List each class with its day and its instructor's name. Which class is missing, and why?

B3 List the first name, last name and class name of every member booked on a class taught by 'Rhys Evans'. You need four tables and a **WHERE** clause.

Check: 4 rows.

Part C Outer joins

slides 14–16, 47–62 · 12 min

C1 List *all* members and the class names they booked, including members with no bookings (like slide 15). Which member has never booked? How can you tell?

Check: 9 rows.

C2 Using **LEFT JOIN**, list every class with its instructor's name, including the class that has no instructor.

Check: 5 rows.

C3 Using **RIGHT JOIN** (Classes on the left, Instructors on the right), list every instructor and the classes they teach, including any instructor who teaches nothing.

Check: 5 rows.

C4 Combine C2 and C3 into a full outer join using **UNION** (slide 55).

Check: 6 rows.

Part D Stored procedures

slides 17–22, 63–66 · 8 min

D1 Create **ViewAllClasses()** that shows every class (like slide 17). Call it.

D2 Create **ViewMemberByID(IN p_member_id INT)** that shows one member (like slide 19). Call it for member 3.

D3 Create **GetMemberBookings(IN p_member_id INT)** that returns the class name, day and price of every class the member has booked. Call it for member 1, then member 5.

Check: member 1 has 2 classes, member 5 has none.

D4 Create **AddMember** with five **IN** parameters (first name, last name, email, credit, joined_on) that inserts a new member (like slide 65). Add Elin, Parry, `elin.parry@example.com`, 10.00, '2026-09-01'.

Check: **SELECT * FROM Members** shows 7 members.

D5 Use the two commands on slide 19 to list your procedures and to show the code of **AddMember**.

Part E Transactions

slides 23–27, 72–80 · 10 min

E1 Move 5.00 credit from Cerys (member 1) to Ffion (member 3) inside a transaction (like slides 75–78). Look at both balances, then **ROLLBACK** and look again. Then repeat the transfer and **COMMIT** it.

Check: after the commit, Cerys 15.00 and Ffion 5.00.

E2 Create **AddWeekendClasses()** that inserts these two classes inside a transaction with an exit handler (like slide 24), then call it:

- 'Weekend Yoga', instructor 1, 'Saturday', capacity 3, price 6.00
- 'Family Swim', no instructor (NULL), 'Sunday', capacity 10, price 4.00

Check: 7 classes.

E3 Create `AddMoreClassesWithError()` (like slide 27). Inside a transaction with an exit handler, insert two valid classes, then a class whose `class_name` is NULL. `class_name` is NOT NULL, so this fails. Call it.

Check: still 7 classes. The two valid inserts were rolled back too.

E4 Create

`TransferCredit(IN p_from INT, IN p_to INT, IN p_amount DECIMAL(6,2))`

that moves credit between two members and rolls back if the sender's credit would go below 0 (like slide 80). Test it:

```
CALL TransferCredit(5, 6, 10.00);  -- should commit
CALL TransferCredit(2, 6, 50.00);  -- should roll back
```

Check: Gareth 20.00, Nia 22.00, Dylan 5.00.

Part F Triggers

slides 67–71 · 5 min

F1 Create a BEFORE UPDATE trigger on Members that sets `last_updated` to `CURRENT_TIMESTAMP` (like slide 69). Test it: change Cerys's email, then `SELECT * FROM Members`.

Check: only Cerys has a `last_updated` value. Why not the members you updated in Part E?

Part G Indexes

slides 81–84 · 2 min

G1 Create an index on `Members(last_name)`, then find the member whose last name is 'Davies'.

G2 Show the indexes on Members. There is an index on `email` that you did not create. Where did it come from?

G3 Drop your index.

Stretch tasks

if you finish early, or for homework

S1 List every instructor, their classes and the first and last names of members booked on them. Keep instructors with no classes and classes with no bookings. Which classes still do not appear, and why?

S2 Create `BookClass(IN p_member_id INT, IN p_class_id INT)`. In one transaction with an exit handler it must take the class price from the member's credit, insert the booking, and roll back if the member's credit would go below 0. Otherwise commit. *Hint: credit = credit - (SELECT price FROM Classes WHERE ...)*

```
CALL BookClass(5, 2);  -- should succeed
CALL BookClass(3, 4);  -- not enough credit
CALL BookClass(5, 2);  -- already booked: the handler should fire
```

Check: Gareth 12.50, Ffion 5.00.

Before you leave

Save your `week4_lab_tasks.sql` with your answers. Keep it for revision: next week builds on transactions (ACID properties).