XIAMEN UNIVERSITY MALAYSIA

Course Code	:	SOF202
Course Name	:	Database
Lecturer	:	Dr. Subashini A/P Ganapathy
Academic Session	:	2025/09
Assessment Title	:	Lab Report
Submission Due Date	:	19 December 2025

Prepared by :

Student ID	Student Name
DSC2409007	Deng Kailong (Leader)
DSC2409009	Du Zhixuan
DSC2409026	Min Yiduo
DSC2409027	Pan Ziliang
DSC2409033	Tao Ouwen

Date Received :

Feedback from Lecturer:

Mark:

XIAMEN UNIVERSITY MALAYSIA

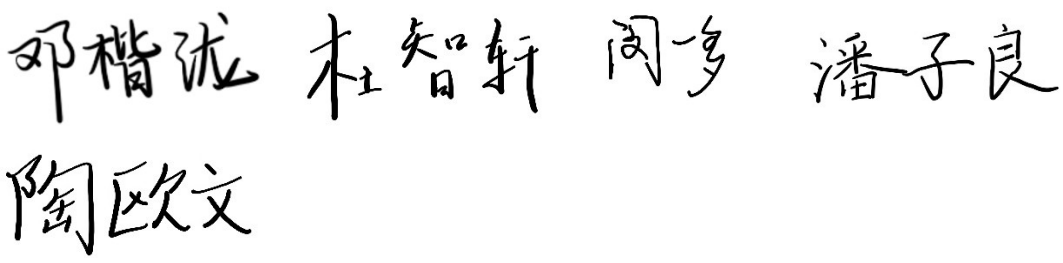
Own Work Declaration

I/We acknowledge that my/our work will be examined for plagiarism or any other form of academic misconduct, and that the digital copy of the work may be retained for future comparisons.

I/We confirm that all sources cited in this work have been correctly acknowledged and that I/we fully understand the serious consequences of any intention'.al or unintentional academic misconduct.

In addition, I/we affirm that this submission does not contain any materials generated by AI tools, including direct copying and pasting of text or paraphrasing. This work is my/our original creation, and it has not been based on any work of other students (past or present), nor has it been previously submitted to any other course or institution.

Signature:



Date: 10 December 2025

Contents

SOF202 Database Lab Report

Group Members: Deng Kailong (Leader), Min Yiduo, Pan Ziliang, Tao Ouwen, Du Zhixuan

Project Start: Wed, 11/19/2025

Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Phase 1 Review & Setup			11/19/25	11/25/25
Review Assignment Schema	All members	100%	11/19/25	11/21/25
Task Assignment	All members	100%	11/20/25	11/21/25
DBMS Environment Setup	Deng Kailong	100%	11/22/25	11/25/25
Milestone 1			11/25/25	11/25/25
Phase 2: Structure Implementation			11/26/25	12/5/25
Task 1: Database Integrity	Pan Ziliang	100%	11/26/25	11/30/25
Task 2A: Create Tables	Deng Kailong	100%	11/26/25	12/1/25
Task 2B: Insert Data (Populat	Deng Kailong	100%	12/1/25	12/5/25
Milestone 2			12/5/25	12/5/25
Phase 3: Functional Development			12/6/25	12/11/25
Task 2C: SQL Queries (Advan	Min Yiduo	100%	12/6/25	12/11/25
Task 3: Triggers Implementat	Tao Ouwen	100%	12/7/25	12/10/25
Task 4: Access Control	Du Zhixuan	100%	12/8/25	12/11/25
Milestone 3			12/11/25	12/11/25
Phase 4 Report Compilation & Review			12/12/25	12/18/25
Report Integration	Deng Kailong,Min Yiduo	100%	12/12/25	12/15/25
Proofreading & Formatting	Deng Kailong,Min Yiduo	100%	12/15/25	12/17/25
Milestone 4			12/17/25	12/17/25
Final Submission			12/18/25	12/18/25

Milestone 1: Environment Ready (11/25/2025)

Confirmed the previous database schema is error-free and the MySQL environment is configured for all group members.

Milestone 2: Database Populated (12/05/2025)

All tables created with correct integrity constraints (PK/FK) and successfully populated with 6-10 rows of sample data.

Milestone 3: Functionality Verified (12/11/2025)

Advanced SQL queries, triggers, and access control mechanisms are implemented, tested, and screenshots are captured.

Milestone 4: Final Submission (12/18/2025)

The final lab report is compiled, formatted according to guidelines, proofread, and submitted via Moodle.

Task 1 Database Integrity

The university sports complex currently relies on outdated methods such as manual registration and spreadsheets to manage equipment, facilities, coaches, and bookings. This has led to issues including double bookings, missing records, and difficulties in tracking equipment maintenance(Frank, Buhmann, & Basin, 2013).

Task 2 SQL

Task 3 Triggering

To ensure data integrity and enforce business rules at the database level, we verified several triggers covering different event types (INSERT and UPDATE). The following sections detail the implementation and verification of these triggers.

3.1 Implemented Triggers

3.1.1 Booking Guardrails

These triggers ensure a member cannot have more than 2 pending bookings, sessions do not exceed 3 hours, and bookings are made within a valid 7-day window.

Listing 3.1: *Trigger: Booking Before Insert Checks*

```
CREATE TRIGGER `booking_bi_guardrails` BEFORE INSERT ON `booking` FOR EACH ROW BEGIN
DECLARE v_pending INT DEFAULT 0;
DECLARE v_reservation_date DATE;
DECLARE v_start TIME;
DECLARE v_end TIME;
DECLARE v_duration_minutes INT;

SELECT COUNT(*) INTO v_pending
FROM `booking`
WHERE `Member_ID` = NEW.Member_ID
    AND `Booking_Status` = 'Pending';

IF NEW.Booking_Status = 'Pending' AND v_pending >= 2 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Member already has 2 pending bookings';
END IF;

SELECT `Reservation_Date`, `Start_Time`, `End_Time`
    INTO v_reservation_date, v_start, v_end
FROM `reservation`
```

```
WHERE `Reservation_ID` = NEW.Reservation_ID;

IF v_reservation_date IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Reservation not found for booking';
END IF;

SET v_duration_minutes = TIMESTAMPDIFF(MINUTE, v_start, v_end);

IF v_duration_minutes > 180 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Single booking session cannot exceed 3 hours';
END IF;

IF v_reservation_date < CURDATE() THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot book for past dates';
END IF;

IF v_reservation_date > DATE_ADD(CURDATE(), INTERVAL 7 DAY) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Bookings can only be made up to 7 days in advance';
END IF;
END;
```

Listing 3.2: *Trigger: Booking Before Update Checks*

```
CREATE TRIGGER `booking_bu_guardrails` BEFORE UPDATE ON `booking` FOR EACH ROW BEGIN
DECLARE v_pending INT DEFAULT 0;
DECLARE v_reservation_date DATE;
DECLARE v_start TIME;
DECLARE v_end TIME;
DECLARE v_duration_minutes INT;

SELECT COUNT(*) INTO v_pending
FROM `booking`
WHERE `Member_ID` = NEW.Member_ID
    AND `Booking_Status` = 'Pending'
    AND `Reservation_ID` <> OLD.Reservation_ID;

IF NEW.Booking_Status = 'Pending' AND v_pending >= 2 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Member already has 2 pending bookings';
END IF;

IF NEW.Reservation_ID <> OLD.Reservation_ID THEN
    SELECT `Reservation_Date`, `Start_Time`, `End_Time`
```

```
    INTO v_reservation_date, v_start, v_end
    FROM `reservation`
    WHERE `Reservation_ID` = NEW.Reservation_ID;

    IF v_reservation_date IS NULL THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Reservation not found for booking';
    END IF;

    SET v_duration_minutes = TIMESTAMPDIFF(MINUTE, v_start, v_end);

    IF v_duration_minutes > 180 THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Single booking session cannot exceed 3 hours';
    END IF;

    IF v_reservation_date < CURDATE() THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot book for past dates';
    END IF;

    IF v_reservation_date > DATE_ADD(CURDATE(), INTERVAL 7 DAY) THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Bookings can only be made up to 7 days in advance';
    END IF;
END IF;
END;
```

3.1.2 Maintenance XOR Constraints

These triggers enforce that a maintenance record targets either a facility OR equipment, but not both and not neither.

Listing 3.3: *Trigger: Maintenance Before Insert XOR Constraint*

```
CREATE TRIGGER `maintenance_bi_xor` BEFORE INSERT ON `maintenance` FOR EACH ROW
  BEGIN
IF (NEW.Facility_ID IS NULL AND NEW.Equipment_ID IS NULL)
    OR (NEW.Facility_ID IS NOT NULL AND NEW.Equipment_ID IS NOT NULL) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Maintenance must target either a facility or equipment (
      exclusively)';
END IF;
END;
```

Listing 3.4: *Trigger: Maintenance Before Update XOR Constraint*

```sql
CREATE TRIGGER `maintenance_bu_xor` BEFORE UPDATE ON `maintenance` FOR EACH ROW
  BEGIN
IF (NEW.Facility_ID IS NULL AND NEW.Equipment_ID IS NULL)
    OR (NEW.Facility_ID IS NOT NULL AND NEW.Equipment_ID IS NOT NULL) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Maintenance must target either a facility or equipment (
      exclusively)';
END IF;
END;
```

3.1.3 Reservation Overlap and Availability

These triggers prevent overlapping reservations for the same facility and ensure the facility is available and not under maintenance.

Listing 3.5: *Trigger: Reservation Before Insert Checks*

```sql
CREATE TRIGGER `reservation_bi_guardrails` BEFORE INSERT ON `reservation` FOR EACH
  ROW BEGIN
DECLARE v_facility_status VARCHAR(20);
DECLARE v_conflicts INT DEFAULT 0;
DECLARE v_maintenance INT DEFAULT 0;

SELECT `Status` INTO v_facility_status
FROM `facility`
WHERE `Facility_ID` = NEW.Facility_ID
FOR UPDATE;

IF v_facility_status IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Facility does not exist for this reservation';
END IF;

IF NEW.Start_Time >= NEW.End_Time THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Reservation start time must be earlier than end time';
END IF;

IF v_facility_status <> 'Available' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Facility is not available for reservation';
END IF;

SELECT COUNT(*) INTO v_maintenance
FROM `maintenance` m
WHERE m.`Facility_ID` = NEW.Facility_ID
    AND m.`Scheduled_Date` = NEW.Reservation_Date
```

```
    AND m.`Status` IN ('Scheduled', 'In_Progress', 'In Progress');

IF v_maintenance > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Facility has maintenance scheduled for the requested date';
END IF;

SELECT COUNT(*) INTO v_conflicts
FROM `reservation` r
WHERE r.`Facility_ID` = NEW.Facility_ID
    AND r.`Reservation_Date` = NEW.Reservation_Date
    AND NOT (NEW.End_Time <= r.`Start_Time` OR NEW.Start_Time >= r.`End_Time`);

IF v_conflicts > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Reservation overlaps with an existing booking for this
      facility';
END IF;
END;
```

Listing 3.6: *Trigger: Reservation Before Update Checks*

```
CREATE TRIGGER `reservation_bu_guardrails` BEFORE UPDATE ON `reservation` FOR EACH
  ROW BEGIN
DECLARE v_facility_status VARCHAR(20);
DECLARE v_conflicts INT DEFAULT 0;
DECLARE v_maintenance INT DEFAULT 0;

SELECT `Status` INTO v_facility_status
FROM `facility`
WHERE `Facility_ID` = NEW.Facility_ID
FOR UPDATE;

IF v_facility_status IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Facility does not exist for this reservation';
END IF;

IF NEW.Start_Time >= NEW.End_Time THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Reservation start time must be earlier than end time';
END IF;

IF v_facility_status <> 'Available' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Facility is not available for reservation';
END IF;
```

```
25
26 SELECT COUNT(*) INTO v_maintenance
27 FROM `maintenance` m
28 WHERE m.`Facility_ID` = NEW.Facility_ID
29     AND m.`Scheduled_Date` = NEW.Reservation_Date
30     AND m.`Status` IN ('Scheduled', 'In_Progress', 'In Progress');
31
32 IF v_maintenance > 0 THEN
33     SIGNAL SQLSTATE '45000'
34     SET MESSAGE_TEXT = 'Facility has maintenance scheduled for the requested date';
35 END IF;
36
37 SELECT COUNT(*) INTO v_conflicts
38 FROM `reservation` r
39 WHERE r.`Facility_ID` = NEW.Facility_ID
40     AND r.`Reservation_Date` = NEW.Reservation_Date
41     AND r.`Reservation_ID` <> OLD.Reservation_ID
42     AND NOT (NEW.End_Time <= r.`Start_Time` OR NEW.Start_Time >= r.`End_Time`);
43
44 IF v_conflicts > 0 THEN
45     SIGNAL SQLSTATE '45000'
46     SET MESSAGE_TEXT = 'Reservation overlaps with an existing booking for this
       facility';
47 END IF;
48 END;
```

3.1.4 Reservation Equipment Availability

These triggers ensure enough equipment stock is available for a reservation, accounting for other reservations and maintenance.

Listing 3.7: *Trigger: Reservation Equipments Before Insert Checks*

```
1 CREATE TRIGGER `reservation_equipments_bi_guardrails` BEFORE INSERT ON `
    reservation_equipments` FOR EACH ROW BEGIN
2 DECLARE v_total INT;
3 DECLARE v_reserved INT DEFAULT 0;
4 DECLARE v_available INT;
5 DECLARE v_status VARCHAR(20);
6 DECLARE v_reservation_date DATE;
7 DECLARE v_start TIME;
8 DECLARE v_end TIME;
9 DECLARE v_maintenance INT DEFAULT 0;
10
11 SELECT r.`Reservation_Date`, r.`Start_Time`, r.`End_Time`
12     INTO v_reservation_date, v_start, v_end
13 FROM `reservation` r
```

```
WHERE r.`Reservation_ID` = NEW.Reservation_ID
FOR UPDATE;

IF v_reservation_date IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Reservation not found for equipment assignment';
END IF;

SELECT e.`Total_Quantity`, e.`Status`
    INTO v_total, v_status
FROM `equipment` e
WHERE e.`Equipment_ID` = NEW.Equipment_ID
FOR UPDATE;

IF v_status IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Equipment does not exist';
END IF;

IF v_status <> 'Available' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Equipment is not available for reservation';
END IF;

IF NEW.Quantity <= 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Quantity must be positive';
END IF;

SELECT COUNT(*) INTO v_maintenance
FROM `maintenance` m
WHERE m.`Equipment_ID` = NEW.Equipment_ID
    AND m.`Scheduled_Date` = v_reservation_date
    AND m.`Status` IN ('Scheduled', 'In_Progress', 'In Progress');

SELECT COALESCE(SUM(re2.`Quantity`), 0) INTO v_reserved
FROM `reservation_equipments` re2
JOIN `reservation` r2 ON r2.`Reservation_ID` = re2.`Reservation_ID`
WHERE re2.`Equipment_ID` = NEW.Equipment_ID
    AND r2.`Reservation_Date` = v_reservation_date
    AND NOT (v_end <= r2.`Start_Time` OR v_start >= r2.`End_Time`);

SET v_available = v_total - v_reserved - v_maintenance;

IF NEW.Quantity > v_available THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Requested equipment exceeds available quantity';
```

```
END IF;
END;
```

Listing 3.8: *Trigger: Reservation Equipments Before Update Checks*

```
CREATE TRIGGER `reservation_equipments_bu_guardrails` BEFORE UPDATE ON `
  reservation_equipments` FOR EACH ROW BEGIN
DECLARE v_total INT;
DECLARE v_reserved INT DEFAULT 0;
DECLARE v_available INT;
DECLARE v_status VARCHAR(20);
DECLARE v_reservation_date DATE;
DECLARE v_start TIME;
DECLARE v_end TIME;
DECLARE v_maintenance INT DEFAULT 0;

SELECT r.`Reservation_Date`, r.`Start_Time`, r.`End_Time`
    INTO v_reservation_date, v_start, v_end
FROM `reservation` r
WHERE r.`Reservation_ID` = NEW.Reservation_ID
FOR UPDATE;

IF v_reservation_date IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Reservation not found for equipment assignment';
END IF;

SELECT e.`Total_Quantity`, e.`Status`
    INTO v_total, v_status
FROM `equipment` e
WHERE e.`Equipment_ID` = NEW.Equipment_ID
FOR UPDATE;

IF v_status IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Equipment does not exist';
END IF;

IF v_status <> 'Available' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Equipment is not available for reservation';
END IF;

IF NEW.Quantity <= 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Quantity must be positive';
END IF;
```

```
42
43 SELECT COUNT(*) INTO v_maintenance
44 FROM `maintenance` m
45 WHERE m.`Equipment_ID` = NEW.Equipment_ID
46     AND m.`Scheduled_Date` = v_reservation_date
47     AND m.`Status` IN ('Scheduled', 'In_Progress', 'In Progress');
48
49 SELECT COALESCE(SUM(re2.`Quantity`), 0) INTO v_reserved
50 FROM `reservation_equipments` re2
51 JOIN `reservation` r2 ON r2.`Reservation_ID` = re2.`Reservation_ID`
52 WHERE re2.`Equipment_ID` = NEW.Equipment_ID
53     AND r2.`Reservation_Date` = v_reservation_date
54     AND NOT (v_end <= r2.`Start_Time` OR v_start >= r2.`End_Time`)
55     AND re2.`id` <> OLD.`id`;
56
57 SET v_available = v_total - v_reserved - v_maintenance;
58
59 IF NEW.Quantity > v_available THEN
60     SIGNAL SQLSTATE '45000'
61     SET MESSAGE_TEXT = 'Requested equipment exceeds available quantity';
62 END IF;
63 END;
```

3.1.5 Session Enrollment Capacity

These triggers prevent enrollment if the training session has reached its maximum capacity.

Listing 3.9: *Trigger: Session Enrollment Before Insert Capacity Check*

```
1 CREATE TRIGGER `session_enrollment_bi_capacity` BEFORE INSERT ON `session_enrollment
    ` FOR EACH ROW BEGIN
2 DECLARE v_capacity INT;
3 DECLARE v_current INT DEFAULT 0;
4
5 SELECT `Max_Capacity` INTO v_capacity
6 FROM `training_session`
7 WHERE `Reservation_ID` = NEW.Reservation_ID
8 FOR UPDATE;
9
10 IF v_capacity IS NULL THEN
11     SIGNAL SQLSTATE '45000'
12     SET MESSAGE_TEXT = 'Training session not found for enrollment';
13 END IF;
14
15 IF v_capacity <= 0 THEN
16     SIGNAL SQLSTATE '45000'
17     SET MESSAGE_TEXT = 'Training session has no available capacity';
```

```
END IF;

SELECT COUNT(*) INTO v_current
FROM `session_enrollment`
WHERE `Reservation_ID` = NEW.Reservation_ID;

IF v_current >= v_capacity THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Training session is full';
END IF;
END;
```

Listing 3.10: *Trigger: Session Enrollment Before Update Capacity Check*

```
CREATE TRIGGER `session_enrollment_bu_capacity` BEFORE UPDATE ON `session_enrollment
  ` FOR EACH ROW BEGIN
DECLARE v_capacity INT;
DECLARE v_current INT DEFAULT 0;

SELECT `Max_Capacity` INTO v_capacity
FROM `training_session`
WHERE `Reservation_ID` = NEW.Reservation_ID
FOR UPDATE;

IF v_capacity IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Training session not found for enrollment';
END IF;

IF v_capacity <= 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Training session has no available capacity';
END IF;

SELECT COUNT(*) INTO v_current
FROM `session_enrollment`
WHERE `Reservation_ID` = NEW.Reservation_ID
    AND `id` <> OLD.`id`;

IF v_current >= v_capacity THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Training session is full';
END IF;
END;
```

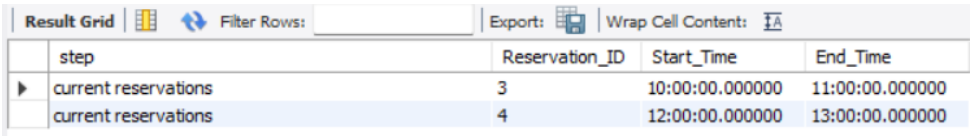
3.2 Evidence Scenarios

3.2.1 Scenario A: Reservation Overlap is Blocked

We first create two non-overlapping reservations for facility 4 (tomorrow).

Listing 3.11: *Scenario A1 SQL: Create two non-overlapping reservations*

```
SET @res_facility := 4;
SET @res_date := DATE_ADD(CURDATE(), INTERVAL 1 DAY);
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@res_date, '10:00:00', '11:00:00', @res_facility);
SET @res_ok_1 := LAST_INSERT_ID();
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@res_date, '12:00:00', '13:00:00', @res_facility);
SET @res_ok_2 := LAST_INSERT_ID();
SELECT 'current reservations' AS step, Reservation_ID, Start_Time, End_Time
FROM reservation WHERE Reservation_ID IN (@res_ok_1, @res_ok_2) ORDER BY
  Reservation_ID;
```



step	Reservation_ID	Start_Time	End_Time
current reservations	3	10:00:00.000000	11:00:00.000000
current reservations	4	12:00:00.000000	13:00:00.000000

Figure 3.1: *Scenario A1: Successful creation of non-overlapping reservations*

Then, we attempt an overlapping insert, which should fail.

Listing 3.12: *Scenario A2 SQL: Attempt overlapping reservation*

```
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@res_date, '10:30:00', '11:30:00', @res_facility);
```

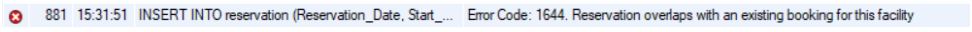


Figure 3.2: *Scenario A2: Overlapping reservation attempt blocked*

Confirmation that only the valid rows remain:

Listing 3.13: *Scenario A3 SQL: Verify reservations found*

```
SELECT Reservation_ID, Start_Time, End_Time
FROM reservation
WHERE Facility_ID = @res_facility AND Reservation_Date = @res_date
ORDER BY Reservation_ID;
```

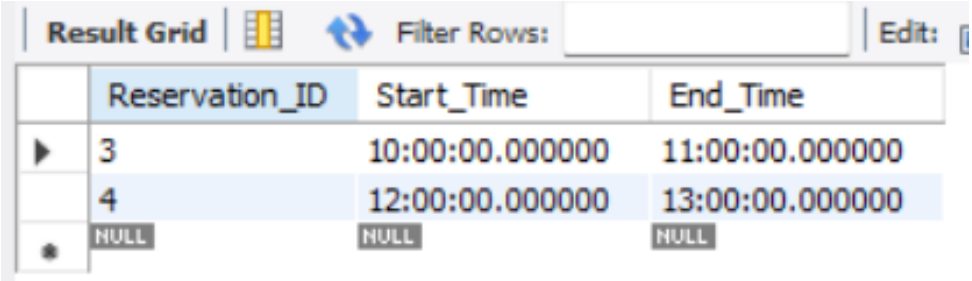


Figure 3.3: *Scenario A3: Verification of existing reservations*

3.2.2 Scenario B: Booking Pending Limit

Limit max 2 pending bookings per member. Prepare three future reservations for member 1:

Listing 3.14: *Scenario B1 SQL: Prepare three reservations*

```
SET @booking_member := 1;
SET @booking_date := DATE_ADD(CURDATE(), INTERVAL 2 DAY);
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@booking_date, '09:00:00', '10:00:00', 5);
SET @book_res_1 := LAST_INSERT_ID();
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@booking_date, '10:10:00', '11:10:00', 5);
SET @book_res_2 := LAST_INSERT_ID();
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@booking_date, '11:20:00', '12:20:00', 5);
SET @book_res_3 := LAST_INSERT_ID();
SELECT 'prep reservations' AS step, Reservation_ID, Reservation_Date, Start_Time,
  End_Time
FROM reservation WHERE Reservation_ID IN (@book_res_1, @book_res_2, @book_res_3)
ORDER BY Reservation_ID;
```

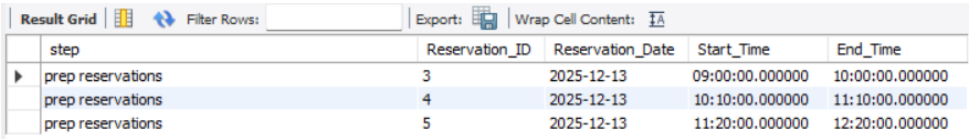


Figure 3.4: *Scenario B1: Preparation of reservations*

Insert two Pending bookings (allowed) and count:

Listing 3.15: *Scenario B2 SQL: Insert two pending bookings*

```
INSERT INTO booking (Reservation_ID, Booking_Status, Member_ID)
VALUES (@book_res_1, 'Pending', @booking_member);
INSERT INTO booking (Reservation_ID, Booking_Status, Member_ID)
VALUES (@book_res_2, 'Pending', @booking_member);
```

```
SELECT 'after two pendings' AS step, COUNT(*) AS pending_count
FROM booking WHERE Member_ID = @booking_member AND Booking_Status = 'Pending';
```

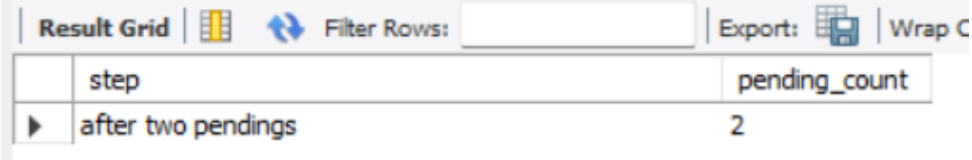


Figure 3.5: *Scenario B2: Two pending bookings successfully created*

Attempt third Pending (should fail) and verify list:

Listing 3.16: *Scenario B3 SQL: Attempt third pending booking*

```
INSERT INTO booking (Reservation_ID, Booking_Status, Member_ID)
VALUES (@book_res_3, 'Pending', @booking_member);
SELECT Reservation_ID, Booking_Status
FROM booking WHERE Member_ID = @booking_member ORDER BY Reservation_ID;
```

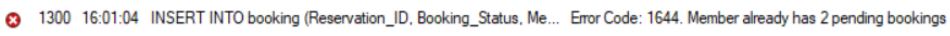


Figure 3.6: *Scenario B3: Third pending booking blocked*

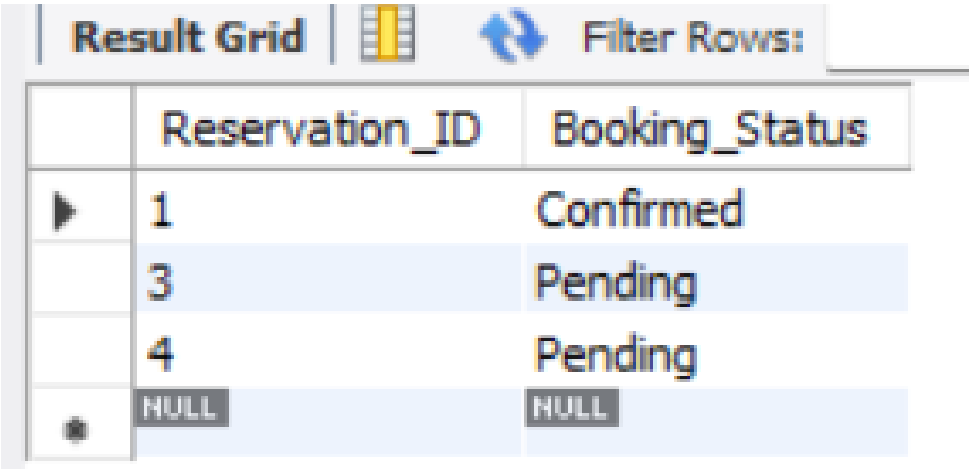


Figure 3.7: *Scenario B4: Verification of booking list*

3.2.3 Scenario C: Maintenance XOR Constraint

Maintenance must target either a facility or equipment, but not both.

Listing 3.17: *Scenario C SQL: Maintenance Insert Tests*

```
INSERT INTO maintenance (Scheduled_Date, Completion_Date, Status, Description,
  Equipment_ID, Facility_ID)
VALUES (DATE_ADD(CURDATE(), INTERVAL 4 DAY), NULL, 'Scheduled', 'Invalid: none',
  NULL, NULL); -- expect error
```

```sql

INSERT INTO maintenance (Scheduled_Date, Completion_Date, Status, Description,
  Equipment_ID, Facility_ID)
VALUES (DATE_ADD(CURDATE(), INTERVAL 4 DAY), NULL, 'Scheduled', 'Invalid: both', 2,
  4); -- expect error

INSERT INTO maintenance (Scheduled_Date, Completion_Date, Status, Description,
  Equipment_ID, Facility_ID)
VALUES (DATE_ADD(CURDATE(), INTERVAL 4 DAY), NULL, 'Scheduled', 'Valid facility
  maintenance', NULL, 4);
SELECT 'valid maintenance inserted' AS step, Maintenance_ID, Equipment_ID,
  Facility_ID, Scheduled_Date, Status
FROM maintenance ORDER BY Maintenance_ID DESC LIMIT 1;
```

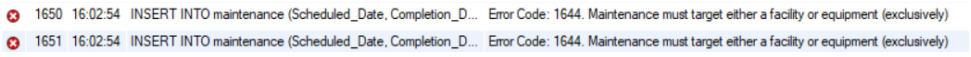


Figure 3.8: *Scenario C1: Invalid maintenance keys blocked*

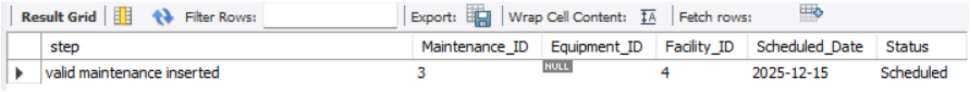


Figure 3.9: *Scenario C2: Valid maintenance entry inserted*

3.2.4 Scenario D: Equipment Availability

Check stock availability during reservation. Setup overlapping reservations for Equipment 2 and check stock:

Listing 3.18: *Scenario D1 SQL: Setup and Stock Check*

```sql
SET @equip_date := DATE_ADD(CURDATE(), INTERVAL 2 DAY);
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@equip_date, '14:00:00', '15:00:00', 8);
SET @equip_res_1 := LAST_INSERT_ID();
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@equip_date, '14:30:00', '15:30:00', 5);
SET @equip_res_2 := LAST_INSERT_ID();
SELECT 'equipment stock snapshot' AS step, Equipment_ID, Equipment_Name, Status,
  Total_Quantity
FROM equipment WHERE Equipment_ID = 2;
```

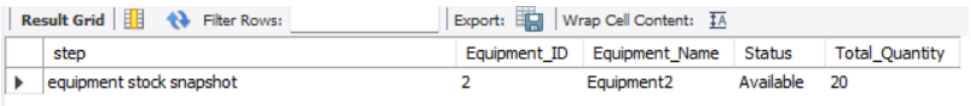


Figure 3.10: *Scenario D1: Equipment stock snapshot*

Valid assign, then over-allocate (should fail):

Listing 3.19: *Scenario D2 SQL: Assignment Tests*

```sql
INSERT INTO reservation_equipments (Quantity, Equipment_ID, Reservation_ID)
VALUES (5, 2, @equip_res_1);
SET @equip_row := LAST_INSERT_ID();
SELECT 'after valid assign' AS step, id, Equipment_ID, Reservation_ID, Quantity
FROM reservation_equipments WHERE id = @equip_row;

INSERT INTO reservation_equipments (Quantity, Equipment_ID, Reservation_ID)
VALUES (16, 2, @equip_res_2); -- expect error
```

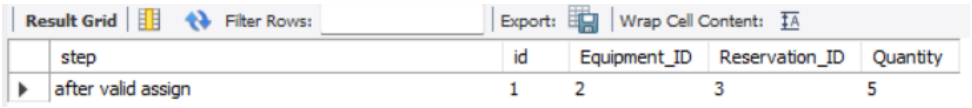


Figure 3.11: *Scenario D2: Valid equipment assignment*

Figure 3.12: *Scenario D3: Over-allocation blocked*

3.2.5 Scenario E: Session Enrollment Capacity

Validate max capacity. Create one-seat session, first enrollment succeeds, second fails:

Listing 3.20: *Scenario E SQL: Session Capacity Tests*

```sql
SET @session_date := DATE_ADD(CURDATE(), INTERVAL 3 DAY);
INSERT INTO reservation (Reservation_Date, Start_Time, End_Time, Facility_ID)
VALUES (@session_date, '16:00:00', '17:00:00', 4);
SET @session_res_full := LAST_INSERT_ID();
INSERT INTO training_session (Reservation_ID, Max_Capacity, Coach_ID)
VALUES (@session_res_full, 1, 1);
SELECT 'session created' AS step, Reservation_ID, Max_Capacity, Coach_ID
FROM training_session WHERE Reservation_ID = @session_res_full;

INSERT INTO session_enrollment (Member_ID, Reservation_ID)
VALUES (1, @session_res_full);
SELECT 'after first enrollment' AS step, id, Member_ID, Reservation_ID
FROM session_enrollment WHERE Reservation_ID = @session_res_full;

INSERT INTO session_enrollment (Member_ID, Reservation_ID)
VALUES (2, @session_res_full); -- expect error
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	step	Reservation_ID	Max_Capacity	Coach_ID
▶	session created	3	1	1

Figure 3.13: *Scenario E1: Session creation*

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	step	id	Member_ID	Reservation_ID
▶	after first enrollment	1	1	3

Figure 3.14: *Scenario E2: First enrollment successful*

2367 16:07:34 INSERT INTO session_enrollment (Member_ID, Reservation_... Error Code: 1644. Training session is full

Figure 3.15: *Scenario E3: Second enrollment blocked due to capacity*

Task 4 Access Control

References

Frank, M., Buhmann, J. M., & Basin, D. (2013). *Role mining with probabilistic models.* Retrieved from https://arxiv.org/abs/1212.4775

APPENDIX 1

MARKING RUBRICS

Component Title	Lab Report (Grouping)				Percentage (%)	24%	
Criteria	Score and Descriptors					Weight	Marks
	Excellent (9.0 – 10.0)	**Good (7.0 – 8.5)**	**Average (5.0 – 6.5)**	**Need Improvement (3.0 – 4.5)**	**Poor (0 – 2.5)**		
Task 1: Database Integrity (20 Marks)	All CREATE TABLE commands are accurate; all integrity constraints correctly applied. Screenshots complete, clear, and well-organized. Explanations are thorough and show strong understanding.	Most tables and constraints are correctly implemented; minor errors. Screenshots provided and generally clear. Explanations show good understanding but lack depth.	Some tables correctly created; constraints may be incomplete or partially wrong. Screenshots provided. Explanations are basic.	Many constraints incorrect or missing. Screenshots unclear or incomplete. Explanations lack clarity and accuracy.	Incomplete or inaccurate SQL statements. Little or no screenshots. Weak or missing explanations.	20	
Task 2: SQL Implementation (30 Marks)	• All remaining CREATE TABLE commands are fully correct and clearly shown with screenshots. • All tables populated with 6–10 valid rows each. • All SQL categories include 3 correct queries each,	• CREATE TABLE commands mostly correct with minor errors. • Tables filled properly with 6–10 rows each. • Most SQL categories include 3 correct queries (some minor mistakes allowed).	• Some CREATE TABLE commands incomplete or inaccurate. • Tables partially populated (some missing rows or incorrect data). • SQL categories include fewer than 3 correct queries or have notable	• CREATE TABLE commands contain multiple mistakes or missing screenshots. • Many tables not properly populated. • Several SQL categories missing or	• Missing or incorrect CREATE TABLE commands. • Very little or no data inserted. SQL queries missing, incorrect, or not executed. • Screenshots mostly missing.	30	

	executed without errors. • Screenshots of queries and outputs are complete, clear, and well-organized. • Demonstrates excellent understanding of SQL, filtering, joins, arithmetic expressions, and formatting.	• Screenshots are provided but may lack full clarity or formatting. • Shows good understanding but misses depth in a few areas.	syntax/output issues. • Screenshots provided but not consistent or clear. • Understanding is present but basic.	incorrectly executed. • Screenshots unclear, incomplete, or missing outputs. • Limited understanding of SQL syntax and application.	• Shows minimal or no understanding of SQL concepts.		
Task 3: Triggering (15 Marks)	Both triggers fully correct and functional. BEFORE/AFTER results clearly demonstrated with proper screenshots. No syntax errors.	Two triggers implemented with minor mistakes. Evidence provided for BEFORE/AFTER. Screenshots mostly clear.	One trigger corrects, the second partially implemented. Limited or basic BEFORE/AFTER evidence.	Triggers contain multiple errors; screenshots unclear or incomplete. Limited understanding shown.	Triggers missing, non-functional, or incorrect. No valid evidence provided.	**15**	
Task 4: Access Control (15 Marks)	Two access control mechanisms implemented correctly. Permissions tested and verified with clear screenshots. Demonstrates strong security understanding.	Both mechanisms implemented with minor issues. Screenshots provided and mostly clear. Verification mostly correct	One mechanism corrects, the other partially correct. Screenshots basic or incomplete.	Incorrect or incomplete access control commands. Screenshots unclear or missing verification	Access control incorrectly implemented or not attempted. No valid evidence.	**15**	
					TOTAL MARKS (/80)		