



Databases and Application Development

ACCA5026

Amir Reza Tamaddoni Jahromi

2413014

Table of Contents

<i>Introduction</i>	<i>2</i>
<i>Database Design Overview</i>	<i>2</i>
Main Tables and Their Attributes.....	2
Table Relationships and Design Structure	2
Normalisation	3
<i>Database Implementation</i>	<i>3</i>
Creating the Database and Tables.....	3
Table Creation	4
Adding Sample Data	7
Using SQL Queries to Retrieve Data	9
Triggers and Automated Database Actions	13
Example Transaction Using Commit and Rollback	15
<i>Improving Performance with Indexes</i>	<i>16</i>
<i>Learning Materials and References</i>	<i>16</i>
<i>Conclusion</i>	<i>17</i>

Introduction

The Students' Union at the university supports many student clubs and societies, such as academic, cultural, and sports groups. Right now, most information about clubs, members, and events is kept on paper or in spreadsheets. This makes it hard to keep records current and can lead to problems with managing memberships, organizing events, and tracking student participation.

To solve these problems, a **Campus Club Management System (CCMS)** database was created. This system keeps all club-related information in one place. It helps the Students' Union manage clubs, memberships, events, participation, and announcements more easily and efficiently.

This report describes how the CCMS database was designed and built using SQL, including table creation, relationships, joins, transactions, stored procedures, triggers, and indexing.

Database Design Overview

Main Tables and Their Attributes

From the scenario, these main entities were identified:

- **Students:** stores information about each student
- **Clubs:** stores information about student clubs
- **Memberships:** records which students belong to which clubs
- **Events:** stores details of events organized by clubs
- **EventRegistrations:** records student participation in events
- **Announcements:** stores messages sent by clubs to members

Table Relationships and Design Structure

A **student can join many clubs**, and a **club can have many students**. This is handled using the **Memberships** table.

A **club can organise many events**, but each event belongs to only one club.

A **student can register for many events**, and an event can have many students. This is handled using the **EventRegistrations** table.

A **club can send many announcements**, but each announcement belongs to one club.

Normalisation

The database has been normalised to **Third Normal Form (3NF)** so that the data is stored in a clear and organised way. Each table has a primary key to uniquely identify records, and there is no repeated data within the tables.

For example, club membership information is stored in the Memberships table rather than repeating student and club data in one table

Many-to-many relationships, such as students joining multiple clubs, are handled using separate tables. All other data in each table depends only on the primary key. This helps to avoid duplicate data and keeps the database accurate and consistent.

Database Implementation

Creating the Database and Tables

Database Creation screenshot

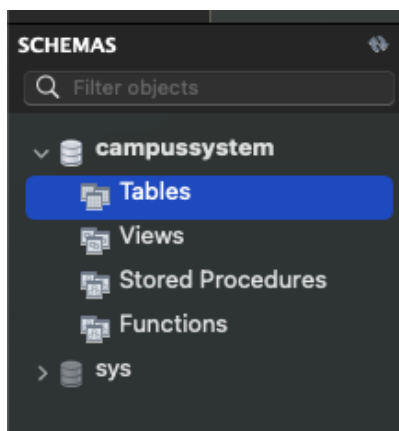
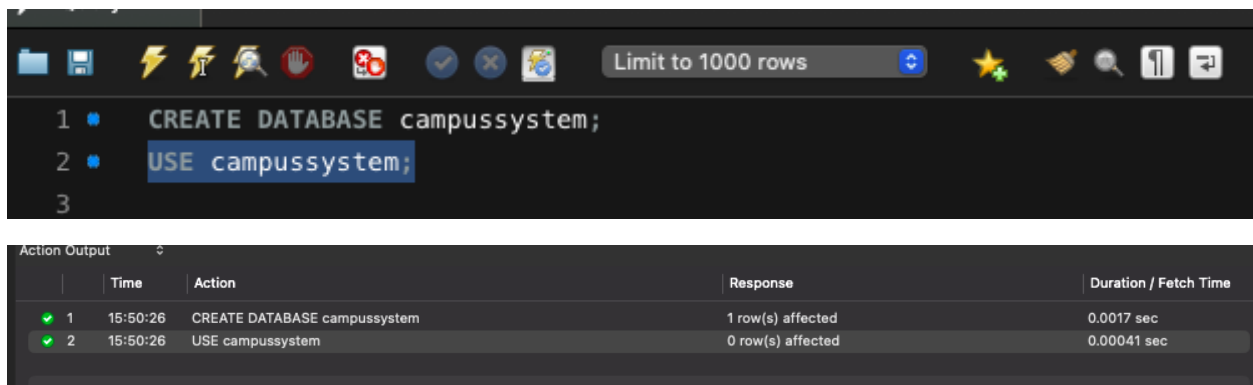


Table Creation

Student Table

```
4 CREATE TABLE Students (  
5     StudentID INT PRIMARY KEY AUTO_INCREMENT,  
6     FirstName VARCHAR(50) NOT NULL,  
7     LastName VARCHAR(50) NOT NULL,  
8     Email VARCHAR(100) NOT NULL UNIQUE,  
9     Programme VARCHAR(80),  
10    YearOfStudy INT CHECK (YearOfStudy BETWEEN 1 AND 6)  
11 );
```

	Time	Action	Response	Duration / Fetch Time
✓ 1	15:50:26	CREATE DATABASE campussystem	1 row(s) affected	0.0017 sec
✓ 2	15:50:26	USE campussystem	0 row(s) affected	0.00041 sec
✓ 3	15:50:59	CREATE TABLE Students (StudentID INT PRIMARY KEY AUTO_INCREMENT,...	0 row(s) affected	0.031 sec

62 DESCRIBE Students;

100% 19:62

Result Grid Filter Rows: Search Export:

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(50)	NO		NULL	
LastName	varchar(50)	NO		NULL	
Email	varchar(100)	NO	UNI	NULL	
Programme	varchar(80)	YES		NULL	
YearOfStudy	int	YES		NULL	

Result 1 Read Only

Clubs Table

```
12  
13 CREATE TABLE Clubs (  
14     ClubID INT PRIMARY KEY AUTO_INCREMENT,  
15     ClubName VARCHAR(80) NOT NULL UNIQUE,  
16     Category VARCHAR(30) NOT NULL,  
17     FoundingDate DATE NOT NULL  
18 );  
19
```

	Time	Action	Response	Duration / Fetch Time
✓ 1	15:50:26	CREATE DATABASE campussystem	1 row(s) affected	0.0017 sec
✓ 2	15:50:26	USE campussystem	0 row(s) affected	0.00041 sec
✓ 3	15:50:59	CREATE TABLE Students (StudentID INT PRIMARY KEY AUTO_INCREMENT,...	0 row(s) affected	0.031 sec
✓ 4	15:51:27	CREATE TABLE Clubs (ClubID INT PRIMARY KEY AUTO_INCREMENT, Clu...	0 row(s) affected	0.039 sec

Membership Table

```

20 CREATE TABLE Memberships (
21     MembershipID INT PRIMARY KEY AUTO_INCREMENT,
22     StudentID INT NOT NULL,
23     ClubID INT NOT NULL,
24     JoinDate DATE NOT NULL,
25     Status ENUM('Active', 'Left') NOT NULL DEFAULT 'Active',
26     CONSTRAINT FK_Member_Student FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
27     CONSTRAINT FK_Member_Club FOREIGN KEY (ClubID) REFERENCES Clubs(ClubID),
28     CONSTRAINT UQ_Student_Club UNIQUE (StudentID, ClubID)
29 );

```

	Time	Action	Response	Duration / Fetch Time
✓ 3	15:50:59	CREATE TABLE Students (StudentID INT PRIMARY KEY AUTO_INCREMENT,...	0 row(s) affected	0.031 sec
✓ 4	15:51:27	CREATE TABLE Clubs (ClubID INT PRIMARY KEY AUTO_INCREMENT, Clu...	0 row(s) affected	0.039 sec
✓ 5	15:51:46	CREATE TABLE Memberships (MembershipID INT PRIMARY KEY AUTO_INC...	0 row(s) affected	0.034 sec

63 DESCRIBE Memberships;

100% 22:63

Result Grid Filter Rows: Search Export:

Field	Type	Null	Key	Default	Extra
MembershipID	int	NO	PRI		auto_increment
StudentID	int	NO	MUL		
ClubID	int	NO	MUL		
JoinDate	date	NO			
Status	enum('Active','Left')	NO		Active	

Result 2 Read Only

Events Table

```

31 CREATE TABLE Events (
32     EventID INT PRIMARY KEY AUTO_INCREMENT,
33     ClubID INT NOT NULL,
34     EventTitle VARCHAR(100) NOT NULL,
35     EventDate DATE NOT NULL,
36     Venue VARCHAR(100) NOT NULL,
37     SeatsAvailable INT NOT NULL CHECK (SeatsAvailable >= 0),
38     MembersOnly BOOLEAN DEFAULT TRUE,
39     CONSTRAINT FK_Event_Club FOREIGN KEY (ClubID) REFERENCES Clubs(ClubID)
40 );
41

```

	Time	Action	Response	Duration / Fetch Time
✓ 4	15:51:27	CREATE TABLE Clubs (ClubID INT PRIMARY KEY AUTO_INCREMENT, Clu...	0 row(s) affected	0.039 sec
✓ 5	15:51:46	CREATE TABLE Memberships (MembershipID INT PRIMARY KEY AUTO_INC...	0 row(s) affected	0.034 sec
✓ 6	15:52:11	CREATE TABLE Events (EventID INT PRIMARY KEY AUTO_INCREMENT, Cl...	0 row(s) affected	0.035 sec

Event Registration Table

```
42 CREATE TABLE EventRegistrations (  
43     RegistrationID INT PRIMARY KEY AUTO_INCREMENT,  
44     EventID INT NOT NULL,  
45     StudentID INT NOT NULL,  
46     RegisteredOn DATE NOT NULL,  
47     Attended BOOLEAN DEFAULT FALSE,  
48     CONSTRAINT FK_Reg_Event FOREIGN KEY (EventID) REFERENCES Events(EventID),  
49     CONSTRAINT FK_Reg_Student FOREIGN KEY (StudentID) REFERENCES Students(StudentID),  
50     CONSTRAINT UQ_Event_Student UNIQUE (EventID, StudentID)  
51 );
```

Action Output				
	Time	Action	Response	Duration / Fetch Time
✓	15:51:46	CREATE TABLE Memberships (MembershipID INT PRIMARY KEY AUTO_INCREMENT, ...	0 row(s) affected	0.034 sec
✓	15:52:11	CREATE TABLE Events (EventID INT PRIMARY KEY AUTO_INCREMENT, Cl...	0 row(s) affected	0.035 sec
✓	15:52:37	CREATE TABLE EventRegistrations (RegistrationID INT PRIMARY KEY AUTO...	0 row(s) affected	0.033 sec

Announcement Table

```
53 CREATE TABLE Announcements (  
54     AnnouncementID INT PRIMARY KEY AUTO_INCREMENT,  
55     ClubID INT NOT NULL,  
56     Subject VARCHAR(100) NOT NULL,  
57     Message TEXT NOT NULL,  
58     SentDate DATE NOT NULL,  
59     CONSTRAINT FK_Ann_Club FOREIGN KEY (ClubID) REFERENCES Clubs(ClubID)  
60 );
```

Action Output				
	Time	Action	Response	Duration / Fetch Time
✓	15:52:11	CREATE TABLE Events (EventID INT PRIMARY KEY AUTO_INCREMENT, Cl...	0 row(s) affected	0.035 sec
✓	15:52:37	CREATE TABLE EventRegistrations (RegistrationID INT PRIMARY KEY AUTO...	0 row(s) affected	0.033 sec
✓	15:53:05	CREATE TABLE Announcements (AnnouncementID INT PRIMARY KEY AUTO...	0 row(s) affected	0.041 sec

Adding Sample Data

Student Table

```
64 |
65 • INSERT INTO Students (FirstName, LastName, Email, Programme, YearOfStudy) VALUES
66 ('Zendaya','Coleman','zendaya.coleman@uni.ac.uk','Drama',2),
67 ('Tom','Holland','tom.holland@uni.ac.uk','Performing Arts',1),
68 ('Ariana','Grande','ariana.grande@uni.ac.uk','Music Technology',3),
69 ('Dwayne','Johnson','dwayne.johnson@uni.ac.uk','Sports Science',4),
70 ('Natalie','Portman','natalie.portman@uni.ac.uk','Psychology',3),
71 ('Keanu','Reeves','keanu.reeves@uni.ac.uk','Philosophy',2),
72 ('Rihanna','Fenty','rihanna.fenty@uni.ac.uk','Fashion',1),
73 ('Robert','Downey','robert.downey@uni.ac.uk','Media Studies',4),
74 ('Lionel','Messi','lionel.messi@uni.ac.uk','Sports Management',2),
75 ('Bill','Gates','bill.gates@uni.ac.uk','Computer Science',4);
76 |
```

100% 1:64

	Time	Action	Response	Duration / Fetch Time
9	15:54:48	DESCRIBE Students	6 row(s) returned	0.033 sec / 0.000025...
10	15:55:48	DESCRIBE Memberships	5 row(s) returned	0.0018 sec / 0.00001...
11	16:15:38	INSERT INTO Students (FirstName, LastName, Email, Programme, YearOfStud...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.051 sec

Clubs Table

```
//
78 • INSERT INTO Clubs (ClubName, Category, FoundingDate) VALUES
79 ('Football Club','Sports','2010-09-15'),
80 ('Art Society','Cultural','2014-02-10'),
81 ('Chess Club','Academic','2008-11-05'),
82 ('Rugby Club','Sports','2005-09-20'),
83 ('Computing Society','Academic','2016-10-01');
84 |
```

	Time	Action	Response	Duration / Fetch Time
10	15:55:48	DESCRIBE Memberships	5 row(s) returned	0.0018 sec / 0.00001...
11	16:15:38	INSERT INTO Students (FirstName, LastName, Email, Programme, YearOfStud...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.051 sec
12	16:29:32	INSERT INTO Clubs (ClubName, Category, FoundingDate) VALUES ('Football C...	5 row(s) affected Records: 5 Duplicates: 0 Warni...	0.027 sec

Membership Table

```
86 • INSERT INTO Memberships (StudentID, ClubID, JoinDate, Status) VALUES
87 (9, 1, '2024-09-12','Active'),
88 (4, 1, '2024-09-14','Active'),
89 (7, 2, '2024-09-16','Active'),
90 (5, 2, '2024-09-18','Active'),
91 (6, 3, '2024-09-17','Active'),
92 (2, 3, '2024-09-19','Active'),
93 (1, 4, '2024-09-15','Active'),
94 (10, 5, '2024-09-13','Active'),
95 (8, 5, '2024-09-20','Active'),
96 (3, 5, '2024-09-21','Active');
87 |
```

Action Output				
	Time	Action	Response	Duration / Fetch Time
11	16:15:38	INSERT INTO Students (FirstName, LastName, Email, Programme, YearOfStud...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.051 sec
12	16:29:32	INSERT INTO Clubs (ClubName, Category, FoundingDate) VALUES ('Football C...	5 row(s) affected Records: 5 Duplicates: 0 Warni...	0.027 sec
13	16:30:19	INSERT INTO Memberships (StudentID, ClubID, JoinDate, Status) VALUES (9, 1...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.025 sec

Events Table

```

97
98 • INSERT INTO Events (ClubID, EventTitle, EventDate, Venue, SeatsAvailable, MembersOnly) VALUES
99 (1, 'Five-a-side Tournament', '2024-10-11', 'Astro Pitch', 20, TRUE),
100 (2, 'Student Art Showcase', '2024-10-09', 'Gallery Room', 8, TRUE),
101 (3, 'Beginner Chess Workshop', '2024-10-10', 'Room C2', 12, TRUE),
102 (4, 'Rugby Skills Training', '2024-10-16', 'Rugby Field', 25, TRUE),
103 (5, 'Hackathon Kickoff', '2024-10-17', 'Computer Lab 3', 6, TRUE);
104

```

Action Output				
	Time	Action	Response	Duration / Fetch Time
12	16:29:32	INSERT INTO Clubs (ClubName, Category, FoundingDate) VALUES ('Football C...	5 row(s) affected Records: 5 Duplicates: 0 Warni...	0.027 sec
13	16:30:19	INSERT INTO Memberships (StudentID, ClubID, JoinDate, Status) VALUES (9, 1...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.025 sec
14	16:32:24	INSERT INTO Events (ClubID, EventTitle, EventDate, Venue, SeatsAvailable, Me...	5 row(s) affected Records: 5 Duplicates: 0 Warni...	0.021 sec

Announcement table

```

104
105 • INSERT INTO Announcements (ClubID, Subject, Message, SentDate) VALUES
106 (1, 'Tournament Notice', 'Five-a-side football tournament on 11 Oct.', '2024-10-02'),
107 (2, 'Art Event', 'Art showcase scheduled for 9 Oct.', '2024-10-01'),
108 (3, 'Workshop Info', 'Beginner chess workshop on 10 Oct.', '2024-10-03'),
109 (4, 'Training Update', 'Rugby training session on 16 Oct.', '2024-10-04'),
110 (5, 'Hackathon Alert', 'Hackathon starts on 17 Oct. Limited seats.', '2024-10-05');
111

```

Action Output				
	Time	Action	Response	Duration / Fetch Time
13	16:30:19	INSERT INTO Memberships (StudentID, ClubID, JoinDate, Status) VALUES (9, 1...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.025 sec
14	16:32:24	INSERT INTO Events (ClubID, EventTitle, EventDate, Venue, SeatsAvailable, Me...	5 row(s) affected Records: 5 Duplicates: 0 Warni...	0.021 sec
15	16:34:14	INSERT INTO Announcements (ClubID, Subject, Message, SentDate) VALUES (...	5 row(s) affected Records: 5 Duplicates: 0 Warni...	0.011 sec

Event Registration table

```
105 • INSERT INTO EventRegistrations (EventID, StudentID, RegisteredOn, Attended) VALUES
106 (1, 9, '2024-10-01', FALSE), -- Messi -> Five-a-Side Tournament
107 (1, 4, '2024-10-02', TRUE), -- Dwayne -> Five-a-Side Tournament
108 (2, 7, '2024-10-03', FALSE), -- Rihanna -> Student Art Showcase (Art member, ok)
109 (2, 5, '2024-10-03', TRUE), -- Natalie -> Student Art Showcase (Art member, ok)
110 (3, 2, '2024-10-04', FALSE), -- Tom -> Chess Workshop (Chess member, ok)
111 (3, 6, '2024-10-04', TRUE), -- Keanu -> Chess Workshop (Chess member, ok)
112 (4, 1, '2024-10-05', FALSE), -- Zendaya -> Rugby Training (Rugby member, ok)
113 (5, 10, '2024-10-06', FALSE), -- Bill -> Hackathon Kickoff (Computing member, ok)
114 (5, 8, '2024-10-06', TRUE); -- Robert -> Hackathon Kickoff (Computing member, ok)
115
```

100% 1:115

Action Output

	Time	Action	Response	Duration / Fetch Time
✓ 18	16:50:28	SELECT c.ClubName, e.EventTitle, e.EventDate, e.Venue, e.SeatsA...	5 row(s) returned	0.0028 sec / 0.00002...
✓ 19	16:52:03	SELECT e.EventTitle, COUNT(r.RegistrationID) AS TotalRegistered FROM...	5 row(s) returned	0.014 sec / 0.000010...
✓ 20	16:55:32	INSERT INTO EventRegistrations (EventID, StudentID, RegisteredOn, Attended)...	9 row(s) affected Records: 9 Duplicates: 0 Warnin...	0.0053 sec

Using SQL Queries to Retrieve Data

Q1: List all clubs

```
124 • SELECT * FROM Clubs;
125
```

Result Grid

ClubID	ClubName	Category	FoundingDate
3	Chess Club	Academic	2008-11-05
4	Rugby Club	Sports	2005-09-20
5	Computing Society	Academic	2016-10-01

Clubs 3

Action Output

	Time	Action	Response	Duration / Fetch Time
✓ 14	16:32:24	INSERT INTO Memberships (ClubID, StudentID, EventDate, Venue, SeatsA...	5 row(s) affected Records: 5 Duplicates: 0 Warnin...	0.021 sec
✓ 15	16:34:14	INSERT INTO Announcements (ClubID, Subject, Message, SentDate) VALUES (...	5 row(s) affected Records: 5 Duplicates: 0 Warnin...	0.011 sec
✓ 16	16:46:05	SELECT * FROM Clubs LIMIT 0, 1000	5 row(s) returned	0.0099 sec / 0.00002...

This query lists all clubs in the database. It is useful for checking that club records have been stored correctly.

Q2: Show all memberships (student name + club name)

```
114 SELECT
115     CONCAT(s.FirstName, ' ', s.LastName) AS StudentName,
116     c.ClubName,
117     m.JoinDate,
118     m.Status
119 FROM Memberships m
120 JOIN Students s ON m.StudentID = s.StudentID
121 JOIN Clubs c ON m.ClubID = c.ClubID;
122
```

100% 45:120

Result Grid Filter Rows: Search Export:

StudentName	ClubName	JoinDate	Status
Rihanna Fenty	Art Society	2024-09-16	Active
Natalie Portman	Art Society	2024-09-18	Active
Keanu Reeves	Chess Club	2024-09-17	Active
Tom Holland	Chess Club	2024-09-19	Active
Bill Gates	Computing Society	2024-09-13	Active
Robert Downey	Computing Society	2024-09-20	Active
Ariana Grande	Computing Society	2024-09-21	Active
Lionel Messi	Football Club	2024-09-12	Active

Result 4 Read Only

Action Output

Time	Action	Response	Duration / Fetch Time
15 16:34:14	INSERT INTO Announcements (ClubID, Subject, Message, SentDate) VALUES (...)	5 row(s) affected Records: 5 Duplicates: 0 Warnin...	0.011 sec
16 16:46:05	SELECT * FROM Clubs LIMIT 0, 1000	5 row(s) returned	0.0099 sec / 0.00002...
17 16:48:23	SELECT CONCAT(s.FirstName, ' ', s.LastName) AS StudentName, c.ClubNa...	10 row(s) returned	0.0089 sec / 0.00001...

This query shows which students are members of which clubs, along with their join dates and membership status, in a clear and easy-to-read way.

Q3: Show events with club name

```
123 SELECT
124     c.ClubName,
125     e.EventTitle,
126     e.EventDate,
127     e.Venue,
128     e.SeatsAvailable,
129     e.MembersOnly
130 FROM Events e
131 JOIN Clubs c ON e.ClubID = c.ClubID
132 ORDER BY e.EventDate;
133
```

100% 18-129

Result Grid Filter Rows: Search Export:

ClubName	EventTitle	EventDate	Venue	SeatsAvailable	MembersOnly
Art Society	Student Art Showcase	2024-10-09	Gallery Room	8	1
Chess Club	Beginner Chess Workshop	2024-10-10	Room C2	12	1
Football Club	Five-a-Side Tournament	2024-10-11	Astro Pitch	20	1
Rugby Club	Rugby Skills Training	2024-10-16	Rugby Field	25	1
Computing Society	Hackathon Kickoff	2024-10-17	Computer Lab 3	6	1

Result 5 Read Only

Action Output

Time	Action	Response	Duration / Fetch Time
16:46:05	SELECT * FROM Clubs LIMIT 0, 1000	5 row(s) returned	0.0099 sec / 0.00002...
16:48:23	SELECT CONCAT(s.FirstName, 's.LastName) AS StudentName, c.ClubNa...	10 row(s) returned	0.0089 sec / 0.00001...
16:50:28	SELECT c.ClubName, e.EventTitle, e.EventDate, e.Venue, e.SeatsA...	5 row(s) returned	0.0028 sec / 0.00002...

This query lists all events with their club's name, date, venue, available seats, and whether they are members-only, ordered by date so the events are easy to follow.

Q4: Count registrations per event

```
146 SELECT
147     e.EventTitle,
148     COUNT(r.RegistrationID) AS TotalRegistered
149 FROM Events e
150 LEFT JOIN EventRegistrations r ON e.EventID = r.EventID
151 GROUP BY e.EventID;
152
```

100% 1:153

Result Grid Filter Rows: Search Export:

EventTitle	TotalRegistered
Five-a-Side Tournament	2
Student Art Showcase	2
Beginner Chess Workshop	2
Rugby Skills Training	1
Hackathon Kickoff	2

Result 8 Read Only

Action Output

Time	Action	Response	Duration / Fetch Time
16:55:32	INSERT INTO EventRegistrations (EventID, StudentID, RegisteredOn, Attended)...	9 row(s) affected Records: 9 Duplicates: 0 Warnin...	0.0053 sec
16:57:52	SELECT e.EventID, e.EventTitle, COUNT(r.RegistrationID) AS TotalRegis...	5 row(s) returned	0.0060 sec / 0.00001...
16:58:31	SELECT e.EventTitle, COUNT(r.RegistrationID) AS TotalRegistered FROM...	5 row(s) returned	0.0065 sec / 0.000...

This query shows each event along with how many students have registered for it, including events with no registrations, by using a left join and grouping the results.

Stored Procedures and Transaction Handling

Scenario: Student 1 registers for Event 1

```
152
153 • START TRANSACTION;
154
155 • INSERT INTO EventRegistrations (EventID, StudentID, RegisteredOn)
156   SELECT 1, 1, CURDATE()
157   WHERE NOT EXISTS (
158     SELECT * FROM EventRegistrations
159     WHERE EventID = 1 AND StudentID = 1
160   );
161
162 • COMMIT;
```

This transaction is used to register a student for an event in a safe way. It starts with `START TRANSACTION` so the registration is treated as a single operation.

The `INSERT` statement adds the student to the `EventRegistrations` table. A `WHERE NOT EXISTS` condition is used to make sure the same student cannot be registered for the same event more than once. If the student is already registered, no data is inserted.

The transaction is completed using `COMMIT`, which saves the change. Seat availability is updated automatically by a trigger when the registration is added, so no manual update is needed. This keeps the database consistent and avoids duplicate or incorrect data.

Scenario: You accidentally register wrong student, then fix it

```
167 • DROP PROCEDURE IF EXISTS MarkAttendance;
168
169 DELIMITER $$
170
171 • CREATE PROCEDURE MarkAttendance(IN in_registration_id INT)
172   BEGIN
173     UPDATE EventRegistrations
174     SET Attended = TRUE
175     WHERE RegistrationID = in_registration_id;
176   END $$
177
178 DELIMITER ;
179
180 • CALL MarkAttendance(1);
181
182
```

24:180

Action Output

	Time	Action	Response	Duration / Fetch Time
✓ 3	17:31:25	COMMIT	0 row(s) affected	0.00048 sec
✗ 4	17:31:25	CREATE PROCEDURE MarkAttendance(IN in_registration_id INT) BEGIN UP...	Error Code: 1304. PROCEDURE MarkAttendance alr...	0.00075 sec
✓ 5	17:33:11	CALL MarkAttendance(1)	0 row(s) affected	0.00070 sec
✓ 6	17:34:32	DROP PROCEDURE IF EXISTS MarkAttendance	0 row(s) affected	0.016 sec
✓ 7	17:34:32	CREATE PROCEDURE MarkAttendance(IN in_registration_id INT) BEGIN UP...	0 row(s) affected	0.0045 sec

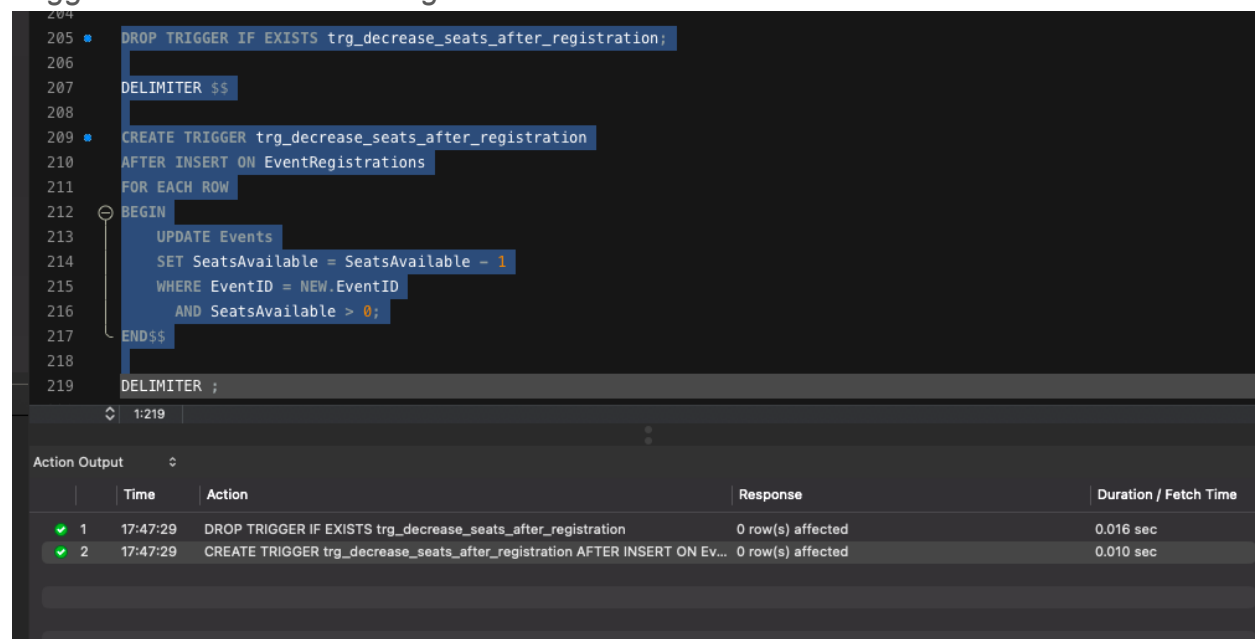
I used `DROP PROCEDURE IF EXISTS` because while testing and rerunning my SQL script, MySQL was showing an error saying the procedure already existed. Dropping it first lets the procedure to be recreated without errors and makes it easier to rerun the script during development.

I chose to use a stored procedure for marking attendance so that this action is controlled in one place rather than updating the table directly each time. This helps keep the database reliable and reduces the chance of mistakes, such as marking the wrong record as attended.

Overall, this approach makes the database easier to manage and reflects how similar tasks would be handled in a real system.

Triggers and Automated Database Actions

Trigger 1: AFTER INSERT registration → decrease SeatsAvailable



```
205 DROP TRIGGER IF EXISTS trg_decrease_seats_after_registration;
206
207 DELIMITER $$
208
209 CREATE TRIGGER trg_decrease_seats_after_registration
210 AFTER INSERT ON EventRegistrations
211 FOR EACH ROW
212 BEGIN
213     UPDATE Events
214     SET SeatsAvailable = SeatsAvailable - 1
215     WHERE EventID = NEW.EventID
216     AND SeatsAvailable > 0;
217 END$$
218
219 DELIMITER ;
```

The screenshot shows a SQL IDE with a script editor and an 'Action Output' panel. The script editor contains SQL code to drop and create a trigger. The 'Action Output' panel shows the execution results of two actions.

	Time	Action	Response	Duration / Fetch Time
1	17:47:29	DROP TRIGGER IF EXISTS trg_decrease_seats_after_registration	0 row(s) affected	0.016 sec
2	17:47:29	CREATE TRIGGER trg_decrease_seats_after_registration AFTER INSERT ON Ev...	0 row(s) affected	0.010 sec

I used this trigger to automatically update the number of available seats whenever a student successfully registers for an event. By placing the logic in a trigger, the seat count is updated immediately after a new registration is added to the `EventRegistrations` table, without needing a separate update query.

The trigger only reduces the seat count if there are seats available, which helps prevent the number of seats from becoming negative. Dropping the trigger before creating it allows the script to be rerun without errors during testing and development.

This approach improves data reliability and reduces the risk of human error by ensuring that seat availability is always kept up to date automatically.

```
221 DELIMITER $$
222
223 CREATE TRIGGER trg_increase_seats_after_cancel
224 AFTER DELETE ON EventRegistrations
225 FOR EACH ROW
226 BEGIN
227     UPDATE Events
228     SET SeatsAvailable = SeatsAvailable + 1
229     WHERE EventID = OLD.EventID;
230 END $$
231
232 DELIMITER ;
```

1:231

	Time	Action	Response	Duration / Fetch Time
✓ 1	17:47:29	DROP TRIGGER IF EXISTS trg_decrease_seats_after_registration	0 row(s) affected	0.016 sec
✓ 2	17:47:29	CREATE TRIGGER trg_decrease_seats_after_registration AFTER INSERT ON Ev...	0 row(s) affected	0.010 sec
✗ 3	17:53:20	ELIMITER \$\$ CREATE TRIGGER trg_increase_seats_after_cancel AFTER DELE...	Error Code: 1064. You have an error in your SQL syn...	0.00027 sec
✓ 4	17:53:29	CREATE TRIGGER trg_increase_seats_after_cancel AFTER DELETE ON EventR...	0 row(s) affected	0.014 sec

I used this trigger to handle the situation where a student's event registration is cancelled or removed. When a registration is deleted from the EventRegistrations table, the trigger automatically increases the number of available seats for that event.

This makes sure that seat availability always stays accurate without needing to manually update the Events table each time a registration is removed. Using OLD.EventID allows the trigger to correctly identify which event the deleted registration belonged to.

Overall, this trigger helps keep data consistency by automatically returning seats when registrations are cancelled, reducing the chance of mistakes and keeping the system consistent.

Example Transaction Using Commit and Rollback

```
169 ■ START TRANSACTION;
170
171 ■ SAVEPOINT BeforeAttempt;
172
173 ■ INSERT IGNORE INTO EventRegistrations (EventID, StudentID, RegisteredOn)
174   VALUES (3, 2, CURRENT_DATE);
175
176 ■ ROLLBACK TO SAVEPOINT BeforeAttempt;
177 ■ INSERT IGNORE INTO EventRegistrations (EventID, StudentID, RegisteredOn)
178   VALUES (3, 4, CURRENT_DATE);
179 ■ UPDATE Events
180   SET SeatsAvailable = SeatsAvailable - 1
181   WHERE EventID = 3
182     AND SeatsAvailable > 0
183     AND EXISTS (
184       SELECT 1 FROM EventRegistrations
185       WHERE EventID = 3 AND StudentID = 4
186     );
187 ■ COMMIT;
188
189
```

37:176

Action Output

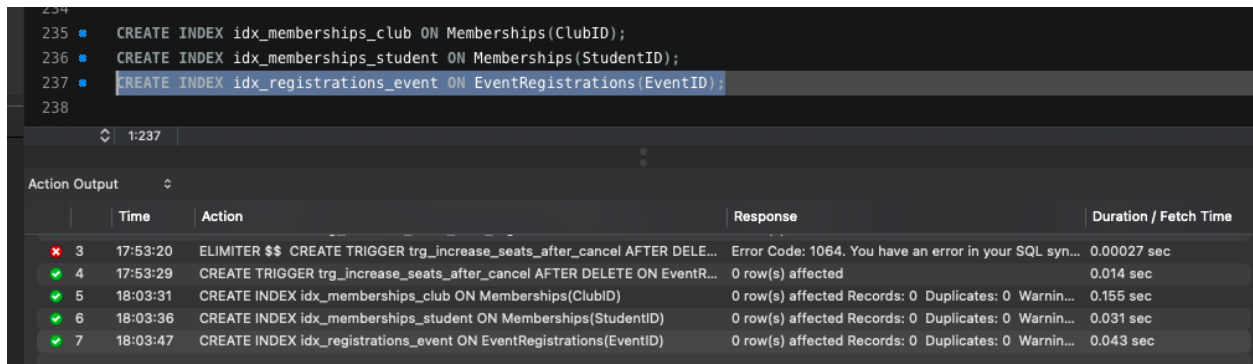
	Time	Action	Response	Duration / Fetch Time
✓ 75	17:23:48	START TRANSACTION	0 row(s) affected	0.012 sec
✓ 76	17:23:48	SAVEPOINT BeforeAttempt	0 row(s) affected	0.00023 sec
⚠ 77	17:23:48	INSERT IGNORE INTO EventRegistrations (EventID, StudentID, RegisteredOn)...	0 row(s) affected, 1 warning(s): 1062 Duplicate entr...	0.0041 sec
✓ 78	17:23:48	ROLLBACK TO SAVEPOINT BeforeAttempt	0 row(s) affected	0.00052 sec
✓ 79	17:23:48	INSERT IGNORE INTO EventRegistrations (EventID, StudentID, RegisteredOn)...	1 row(s) affected	0.00057 sec
✓ 80	17:23:48	UPDATE Events SET SeatsAvailable = SeatsAvailable - 1 WHERE EventID = 3...	1 row(s) affected Rows matched: 1 Changed: 1 War...	0.0031 sec

I used this transaction to deal with a situation where a registration might be entered incorrectly and then corrected straight away. The savepoint allows me to undo just part of the transaction instead of cancelling everything, which makes it easier to fix small mistakes.

INSERT IGNORE is used so that if the student is already registered, the database does not return an error and stop the transaction. After rolling back to the savepoint, the correct student is registered, again without causing duplicate registration errors.

Seat availability is updated automatically by the trigger when the correct registration is inserted. Using a transaction will make sure that all these steps happen together and keeps the data consistent.

Improving Performance with Indexes



The screenshot shows a SQL editor with three commands: `CREATE INDEX idx_memberships_club ON Memberships(ClubID);`, `CREATE INDEX idx_memberships_student ON Memberships(StudentID);`, and `CREATE INDEX idx_registrations_event ON EventRegistrations(EventID);`. Below the editor is an 'Action Output' table with columns: Time, Action, Response, and Duration / Fetch Time.

	Time	Action	Response	Duration / Fetch Time
✖ 3	17:53:20	ELIMITER \$\$ CREATE TRIGGER trg_increase_seats_after_cancel AFTER DELETE ON EventR...	Error Code: 1064. You have an error in your SQL syn...	0.00027 sec
✔ 4	17:53:29	CREATE TRIGGER trg_increase_seats_after_cancel AFTER DELETE ON EventR...	0 row(s) affected	0.014 sec
✔ 5	18:03:31	CREATE INDEX idx_memberships_club ON Memberships(ClubID)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.155 sec
✔ 6	18:03:36	CREATE INDEX idx_memberships_student ON Memberships(StudentID)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.031 sec
✔ 7	18:03:47	CREATE INDEX idx_registrations_event ON EventRegistrations(EventID)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.043 sec

I added these indexes to help the database run more efficiently when searching and joining tables. In my system, the Memberships and EventRegistrations tables are used a lot to link students to clubs and events, so the columns that store these relationships are queried frequently.

By indexing ClubID and StudentID in the Memberships table, the database can find members of a club or clubs a student belongs to more quickly. I also added an index to EventID in the EventRegistrations table because it is usually used when counting registrations or displaying event details.

This doesn't change how the database works, but it improves performance and makes the system more suitable for handling larger amounts of data as the database grows.

Learning Materials and References

From Moodle UWTSD

Doc - Hospital joins tutorial (constraints + joins) Tutorial - SQL queries with joins for hospital management system_solutions'

Doc - Library transactions/procedures/triggers tutorial – 'Tutorial -Illustrating transactions Stored Procedures and trigerrs Questions'

Conclusion

I successfully designed and built the Campus Club Management System database to meet the needs of the Students' Union. The system allows information about clubs, students, memberships, events, registrations, and announcements to be stored and managed in a simple and organised way.

The database uses basic good practices such as primary keys, foreign keys, and junction tables to manage relationships between data. I used SQL queries with joins to get useful information, such as which students belong to which clubs and details about events. Transactions were used to make sure event registrations are handled safely, and triggers were used to automatically update seat availability without manual updates.

Overall, the database works well for managing campus clubs and student participation. Using transactions, stored procedures, triggers, and indexes helps keep the data accurate and ensures the system runs correctly.