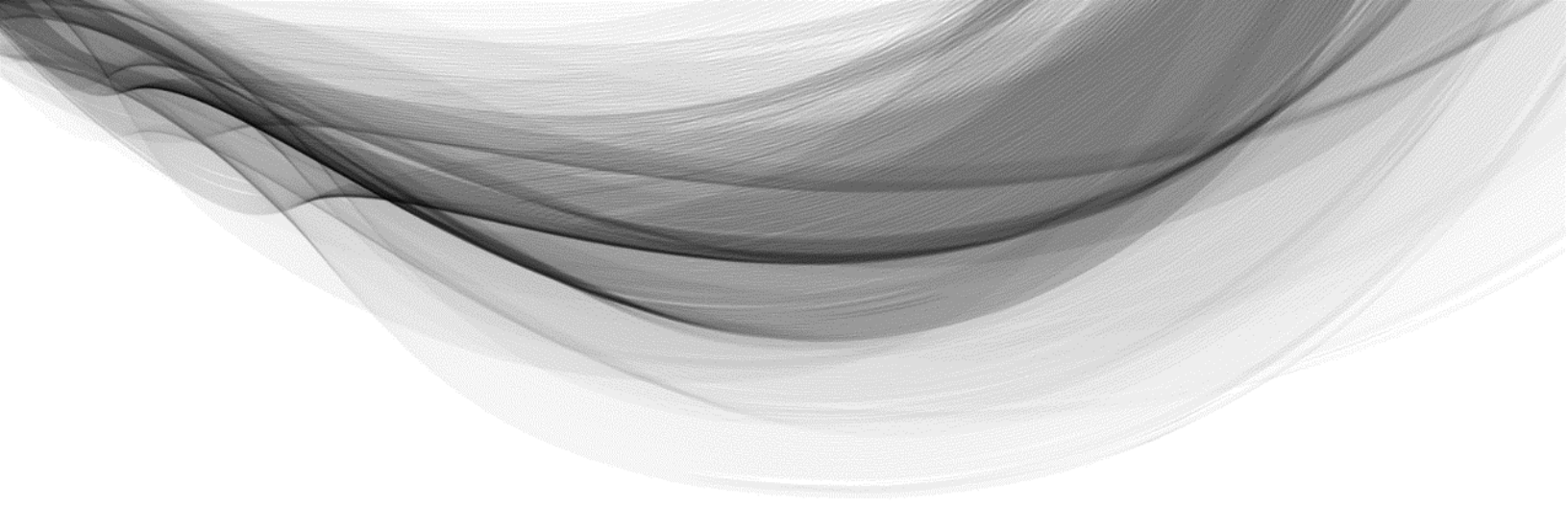




Part A: Database Design and Implementation

L5 – Database and Application Development

Module Code: ACCA5026

BSc Computing (Software Engineering)

Contents

1. Introduction	3
2. Database Design	3
2.1 Entity Identification and Relationships	3
2.2 Normalisation to 3NF	4
3. Implementation Documentation	4
3.1 Table Creation and Constraints	4
3.2 Sample Data Insertion	5
3.3 Stored Procedures for CRUD Operations	5
3.4 Triggers and Auditing	6
4. Stored Procedures and Transactional Scenario	7
5. Performance Tuning and Indexing	7
6. Conclusion	8
7. References	9

1. Introduction

A university’s Students’ Union plays an essential role in student life, enhancing engagement through a wide range of clubs and societies, including academia, sporting, social and special-interest groups. As Students’ Unions expand their support for such clubs, reliance on paper-based records and isolated spreadsheets to manage club-related operations becomes increasingly inadequate. This fragmented approach leads to difficulty in tracking memberships, organising events, communicating announcements, and evaluating overall engagement.

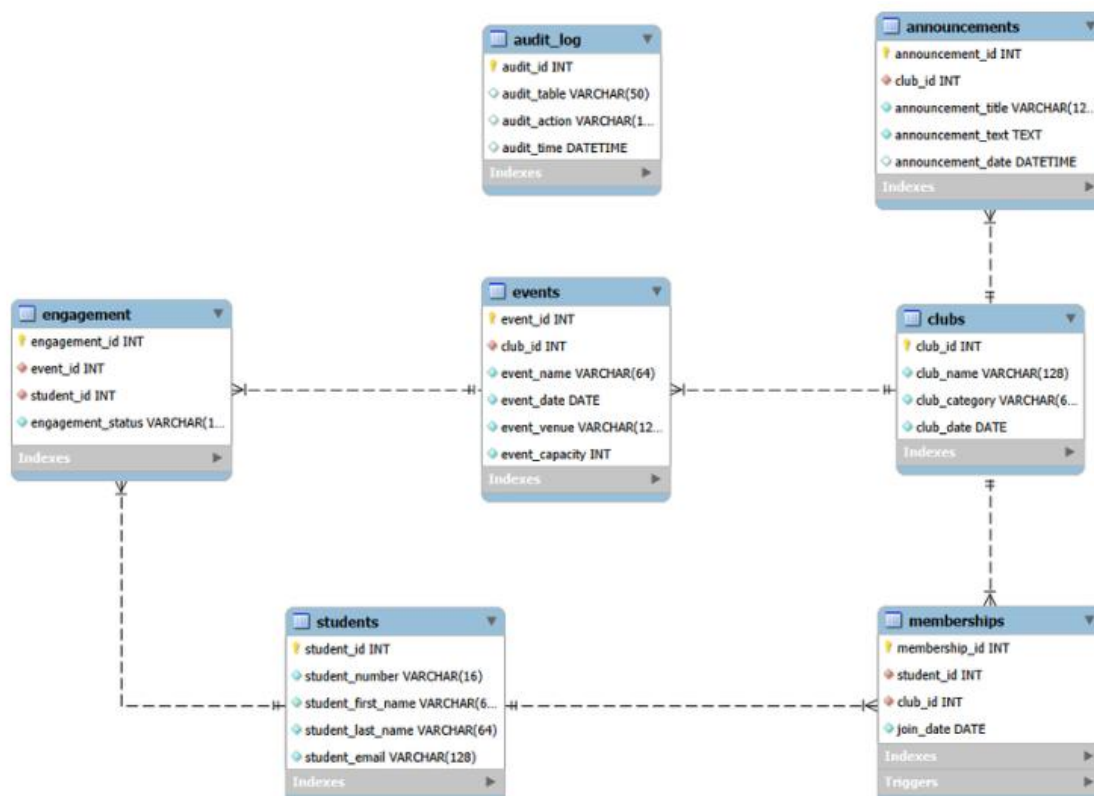
This report examines the implementation of a Campus Club Management System (CCMS), a structured database solution that aims to centralise club operations while improving the efficiency of club and society management. The proposed solution is implemented as a relational database system that supports data integrity, minimises redundancy and demonstrates its functionality using appropriate test data and documentation.

2. Database Design

This section aims to identify the key entities and relationships within the CCMS while explaining how the design meets Third Normal Form (3NF) principles.

2.1 Entity Identification and Relationships

Based on the functional requirements of the CCMS, both a set of key entities and relationships were identified, supporting the management of clubs, students, memberships, events, announcements, engagement and audit logging. Figure 1 contains an Entity Relationship Diagram (ERD) illustrating these entities and their relationships.



[Figure. 1]

The seven entities identified include:

- **clubs:** The clubs entity forms the foundation of the relational database system, storing key information relating to club and society names, categories and founding dates.
- **students:** The students entity supports club memberships and event engagement, storing student specific information such as university identification numbers and contact details.
- **memberships:** The memberships entity represents student enrolment within individual clubs and societies, containing foreign keys for unique student and club ids.
- **events:** The events entity contains data relating to club specific events, storing event information and a foreign key referencing associated clubs.
- **announcements:** The announcement entity stores club specific announcements, containing related information and a foreign key referencing club ids.
- **engagement:** The engagement entity records student engagement within registered events. Foreign keys such as event and student id are stored along side student specific engagement status.
- **audit_log:** The audit_log entity tracks changes made to tables within the database, such as membership insertion and deletion. Data related to the affected tables, actions and times are all stored.

Within the proposed design, there are multiple many-to-many relationships. A student may join multiple clubs, and each club may have multiple students. This relationship is connected via the memberships table, recording individual memberships and using foreign keys to associate a single student to a single club. Similarly, student participation within events is connected via the engagement table, allowing multiple students to be linked to multiple events while recording engagement status and maintaining data integrity.

One-to-many relationships can also be observed within Figure 1. Each individual club may organise many events or issue multiple announcements. Furthermore, each event may be related to multiple engagement records, allowing for accurate tracking of individual student participation.

2.2 Normalisation to 3NF

The proposed design for the CCMS was normalised to Third Normal Form (3NF), organising data in order to reduce redundancy [1]. First Normal Form (1NF) is achieved by ensuring all entries in a column are atomic and all rows are unique. Second Normal Form (2NF) is met by ensuring there is no partial dependency within connecting tables such as memberships and engagement. 3NF is satisfied by removing transitive dependency, meaning non-key values depend on only the primary key [2]. As a result, student, event and club data are stored within separate entities rather than duplicated across multiple tables. This approach reduces anomalies and supports consistency within data management.

3. Implementation Documentation

This section documents the implementation of the proposed relational database system.

3.1 Table Creation and Constraints

The relational database system was implemented via the use of MySQL in alignment with the design presented in the ERD shown within Figure 1.

```

32 • CREATE TABLE memberships (
33     membership_id INT AUTO_INCREMENT PRIMARY KEY,
34     student_id INT NOT NULL,
35     club_id INT NOT NULL,
36     join_date DATE NOT NULL,
37     FOREIGN KEY (student_id) REFERENCES students(student_id),
38     FOREIGN KEY (club_id) REFERENCES clubs(club_id),
39     UNIQUE (student_id, club_id)
40 );

```

[Figure. 2]

As illustrated in Figure 2, the CCMS database schema was implemented by translating each unique entity into a corresponding table containing a primary key for unique record identification. In order to enforce relationships between individual tables, foreign keys were also applied to tables, ensuring consistency across the database. In addition, unique constraints were also defined in order to prevent duplicate records, further supporting data integrity.

3.2 Sample Data Insertion

With the implementation of tables, realistic sample data was inserted in order to simulate a real-world application of the CCMS. As shown in Figure 3 this included students, as well as several clubs, memberships, events, announcements and student engagement with said events. This inserted data was later used to validate table relationships and confirm that both primary and foreign key function as intended.

```

13 • INSERT INTO students (student_number, student_first_name, student_last_name, student_email)
14     VALUES
15     ('2317034', 'Jacob', 'Moseley', '2317034@student.uwtsd.ac.uk'),
16     ('2235034', 'Noah', 'Saponara', '2235034@student.uwtsd.ac.uk'),
17     ('2457454', 'Timothy', 'Bohm', '2457454@student.uwtsd.ac.uk'),
18     ('2312354', 'Samuel', 'Balaz', '2312354@student.uwtsd.ac.uk'),
19     ('2457734', 'Beatriz', 'Linder', '2457734@student.uwtsd.ac.uk'),
20     ('2356034', 'Andreas', 'Wustinger', '2356034@student.uwtsd.ac.uk'),
21     ('2843434', 'Antonious', 'Antonio', '2843434@student.uwtsd.ac.uk'),
22     ('3445653', 'Giovanni', 'Galvao', '3445653@student.uwtsd.ac.uk'),
23     ('3477453', 'Scarlett', 'Moseley', '3477453@student.uwtsd.ac.uk');

```

[Figure. 3]

3.3 Stored Procedures for CRUD Operations

Stored procedures were used to implement CRUD operations, supporting maintainability and reducing the risk of inconsistent or incorrect queries being executed within the relational database system. Figure 4 demonstrates the delete_membership stored procedure, which removes membership records based on the provided membership ID.

The procedure accepts membership_id, before starting a transaction to ensure the operation's entire sequence is executed atomically [3]. After the delete operation, the number of affected rows is checked using a conditional statement. In the event no records are deleted a rollback is executed to prevent the invalid operation from being committed. If the deletion is a success, the transaction is committed.

```

1 • USE ccms;
2
3 DELIMITER $$
4
5 • CREATE PROCEDURE delete_membership (
6     IN delete_membership_id INT
7 )
8
9 BEGIN
10     START TRANSACTION;
11     DELETE FROM memberships
12     WHERE membership_id = delete_membership_id;
13     IF ROW_COUNT() = 0 THEN
14         ROLLBACK;
15     ELSE
16         COMMIT;
17     END IF;
18 END $$
19
20 DELIMITER ;

```

[Figure. 4]

Figure 5 displays the output of a successful execution of the stored procedure illustrated within Figure 4.

✓	16	05:00:57	SELECT * FROM memberships LIMIT 0, 1000	8 row(s) returned
✓	17	05:00:57	CALL delete_membership(15)	0 row(s) affected
✓	18	05:00:57	SELECT * FROM memberships LIMIT 0, 1000	7 row(s) returned

[Figure. 5]

3.4 Triggers and Auditing

Auditing features were implemented using database triggers to automatically detect data modification events such as insertions and deletions. These triggers populate the audit_log table and ensure changes are both logged consistently, and accurately. Figure 6 illustrates the membership_insert trigger.

```

1 • USE ccms;
2
3 DELIMITER $$
4
5 • CREATE TRIGGER membership_insert
6 AFTER INSERT
7 ON memberships
8 FOR EACH ROW
9 BEGIN
10     INSERT INTO audit_log (audit_table, audit_action)
11     VALUES ('memberships', 'INSERT');
12 END $$
13
14 DELIMITER ;

```

[Figure. 6]

Figure 7 illustrates the audit_log table and its associated attributes.

	audit_id	audit_table	audit_action	audit_time
▶	1	memberships	INSERT	2026-01-14 03:23:13
	2	memberships	INSERT	2026-01-15 05:21:38
	3	memberships	DELETE	2026-01-15 05:29:59
✱	NULL	NULL	NULL	NULL

[Figure. 7]

4. Stored Procedure and Transactional Scenario

This section outlines a transactional scenario implemented within the proposed CCMS. As illustrated in Figure 8, this scenario applies to event registration, inserting a registration record and reducing the remaining event capacity by one. Both commands are executed within a single transaction to ensure atomicity and prevent partial updates. A conditional statement is used to ensure that an insert is only executed when capacity is available. If an event has reached full capacity a rollback is executed ensuring the preservation of data integrity. This approach prevents registrations being recorded without capacity being updated.

```
1 • USE ccms;
2
3 DELIMITER $$
4
5 • CREATE PROCEDURE event_registration (
6     IN registration_event_id INT,
7     IN registration_student_id INT
8 )
9 BEGIN
10     START TRANSACTION;
11     IF (SELECT event_capacity FROM events WHERE event_id = registration_event_id) > 0 THEN
12         INSERT INTO engagement (event_id, student_id, engagement_status)
13         VALUES (registration_event_id, registration_student_id, 'registered');
14
15         UPDATE EVENTS
16         SET event_capacity = event_capacity - 1
17         WHERE event_id = registration_event_id;
18
19         COMMIT;
20     ELSE
21         ROLLBACK;
22     END IF;
23 END $$
24
25 DELIMITER ;
```

[Figure. 8]

5. Performance Tuning and Indexing

In order to improve the speed of data retrieval, indexes were applied to related columns that are frequently used in both join and search operations. These include queries such as retrieving a student's membership, listing participants for a specific event and filtering events by clubs. This reduces the need for full table scans and significantly reduces query execution time as the number of records within the CCMS [4]. Implementation of indexes is illustrated within Figure 9, where indexes were created on foreign key columns such as student_id, event_id, and club_id.

```
1 • USE ccms;
2
3 • CREATE INDEX idx_memberships_student ON memberships(student_id);
4 • CREATE INDEX idx_engagement_event ON engagement(event_id);
5 • CREATE INDEX idx_events_club_date ON events(club_id);
6 • CREATE INDEX idx_memberships_club ON memberships(club_id);
```

[Figure. 9]

6. Conclusion

In conclusion, this report has documented the design and implementation of a relational database system that supports the management of both clubs and societies within the CCMS. Through the implementation of a database schema, primary and foreign keys, stored procedures, transactions, triggers and indexing, the system ensures data integrity and efficient operation.

Overall, the implementation effectively meets the functional requirements of the Students' Union, supporting both scalability and maintainability. Future improvements could include the addition of more stored procedures supporting more advanced administrative queries.

7. References

- [1] M. Demba, "Database security and privacy challenges," *International Journal of Database Management Systems (IJDMS)*, vol. 5, no. 3, Jun. 2013, doi: 10.5121/ijdms.2013.5303, https://d1wqtxts1xzle7.cloudfront.net/37950350/5313ijdms03-libre.pdf?1434779031=&response-content-disposition=inline%3B+filename%3DALGORITHM_FOR_RELATIONAL_DATABASE_NORMAL.pdf&Expires=1768447540&Signature=SmdlpK0FVg6Ftjpx8UB7Zf22BxITUC-MOVfwwMW5eXDYAkfAM3e361rq4rUPI6d5uE1WYr-9XNS9uPkfC2grUfdow7mrhGV28KACg201117wiEp-oloPpDNfNfGDpXZ0E9DN-JUfRifve0rsUUg-wAtCmRkCHurEbwMPNO1LME7m9JXd26Gjh4ldF9FFKtmKp6TsqA0Dt16q2C-6bGG25OhBt7Zff9G3ctCbWVNp8tGRuiLToU07M6Y8iq07P6EovO4bL7DSVwwTMGniVRbONizh8sDxSL75X0MeQnpPhBEI WccjVkJ7ebQVsPQBe9g1FUURJ4NCQ5NbmuNR58kuQ_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA (Accessed Jan. 15)
- [2] University of Wales Trinity Saint David, *Normalisation*, Lecture slides, Week 2: Functional Dependency and Normalisation, Swansea, UK, Sep. 30, 2019 (Accessed Jan. 15)
- [3] University of Wales Trinity Saint David, *Advanced MySQL Concepts – Understanding Database Transactions*, Lecture slides, Swansea, UK, 2022 (Accessed Jan. 15)
- [4] University of Wales Trinity Saint David, *Enhancing Database Design*, Lecture slides, Swansea, UK, 2022 (Accessed Jan. 15)