University Student Data Management System
ER, MySQL and MongoDB Implementation

BY

SACHITH M. GUNAWARDANE

2024

TABLE OF CONTENTS

1 Objective

The primary objective of this assignment is to design and implement a University Student Data Management System using two distinct types of database systems: MySQL, a relational database, and MongoDB, a NoSQL database. This initiative will:

- **Explore and Document the Application of Each Database System:**
  We will assess how each database system manages data effectively, highlighting their unique approaches and benefits in handling various aspects of data management within a university setting.
- **Focus on a Defined Scope of Entities:**
  Although the potential scope of a University Student Data Management System is vast and capable of incorporating entities such as Instructors, Exams, and more, this assignment will concentrate on a specific set of entities and relationships which were discussed within the assessment. The chosen scope includes Students, Courses, Enrollments, Departments, and Semesters. This deliberate limitation allows for a detailed exploration of these core components without the complexity introduced by additional entities.
- **Utilise Flexible Data Modeling Tools:**
  The entities are designed to maximise flexibility in data capture, allowing for the use of a broad range of tools to create comprehensive Entity-Relationship (ER) diagrams. This approach ensures that the data model is robust and adaptable, capable of evolving to meet future needs within the university framework.

2 Task 1: Database Design

2.1 MySQL Design

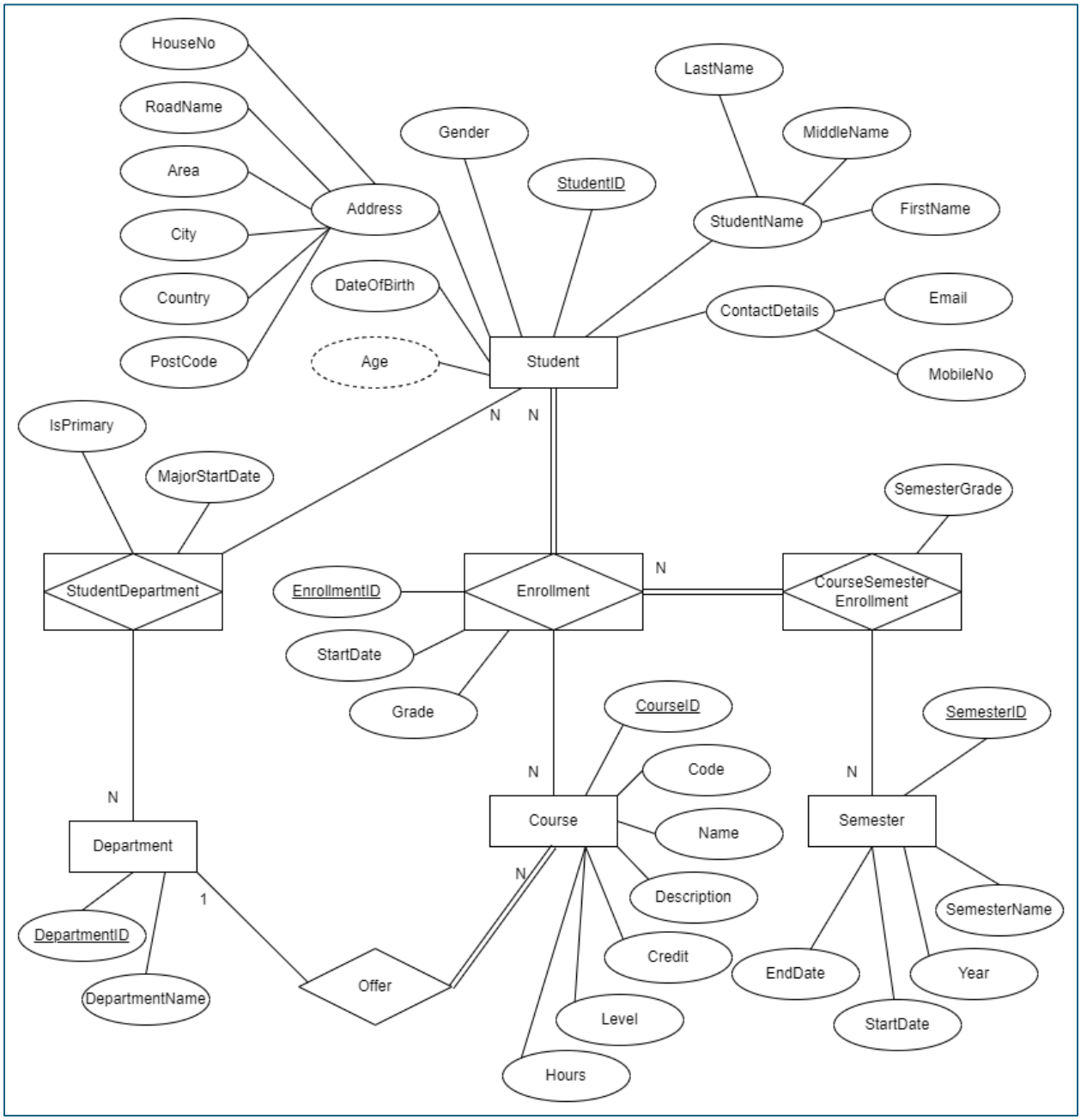


Figure 1: ER Diagram

Entities and Relationships Overview

1. Student Entity

Attributes: The Student entity includes a unique StudentID and composite attributes such as StudentName (FirstName, MiddleName, LastName), Address (HouseNo, RoadName, Area, City, Country, PostCode), and ContactDetails (MobileNo, Email). Gender and DateOfBirth are also key attributes.

Performance Considerations: The nested composite attributes are flattened in relational databases for enhanced performance and manageability. In contrast, MongoDB's document-based system optimally handles these nested structures.

Derived Attributes: The student's age is a derived attribute calculated from the DateOfBirth, and thus is not stored in the database directly but managed at the application's business logic layer.

2. Course Entity:

Attributes: Each course is defined by a CourseID, Code, Name, Description, Credit, Level, and Hours.

Relationships: The Course entity has a many-to-many relationship with the Student entity through an associative entity called Enrollment. This structure allows the flexibility of having courses with no current enrollments.

3. Enrollment Associative Entity:

Attributes: Includes EnrollmentID, Grade, and StartDate.

Key Design: The introduction of a primary key in Enrollment is strategic to manage its complex many-to-many relationship with the Semester entity, simplified through the use of the CourseSemesterEnrollment associative entity.

Notes: The grade captured at the enrollment level represents the overall academic performance up to that date, as referred to in assessment task 3.

4. Semester Entity:

Attributes: Comprises SemesterID, SemesterName, Year, StartDate, and EndDate.

Relationships: Enrollments are linked to Semesters via the CourseSemesterEnrollment associative entity, indicating that a course enrollment may extend over multiple semesters. This assumes potential future semesters without current enrollments.

5. CourseSemesterEnrollment:

Attributes: This primarily includes SemesterGrade, which provides a means to capture semester-specific performance for each course enrollment.

6. Department Entity:

Attributes: Defined by DepartmentID and DepartmentName.

Relationships: Manages a one-to-many relationship with Courses through an entity called Offer, where each course is associated with only one department, though a department can offer multiple courses.

7. StudentDepartment Associative Entity:

Attributes: Includes MajorStartDate and IsPrimary.

Functionality: Facilitates tracking multiple majors for students, with IsPrimary indicating the primary major. This setup enhances flexibility in student academic pathways.

Figure 2 displays the logical diagram, which has been meticulously designed based on the Entity-Relationship (ER) diagram depicted in Figure 1, incorporating the assumptions and established facts discussed previously.

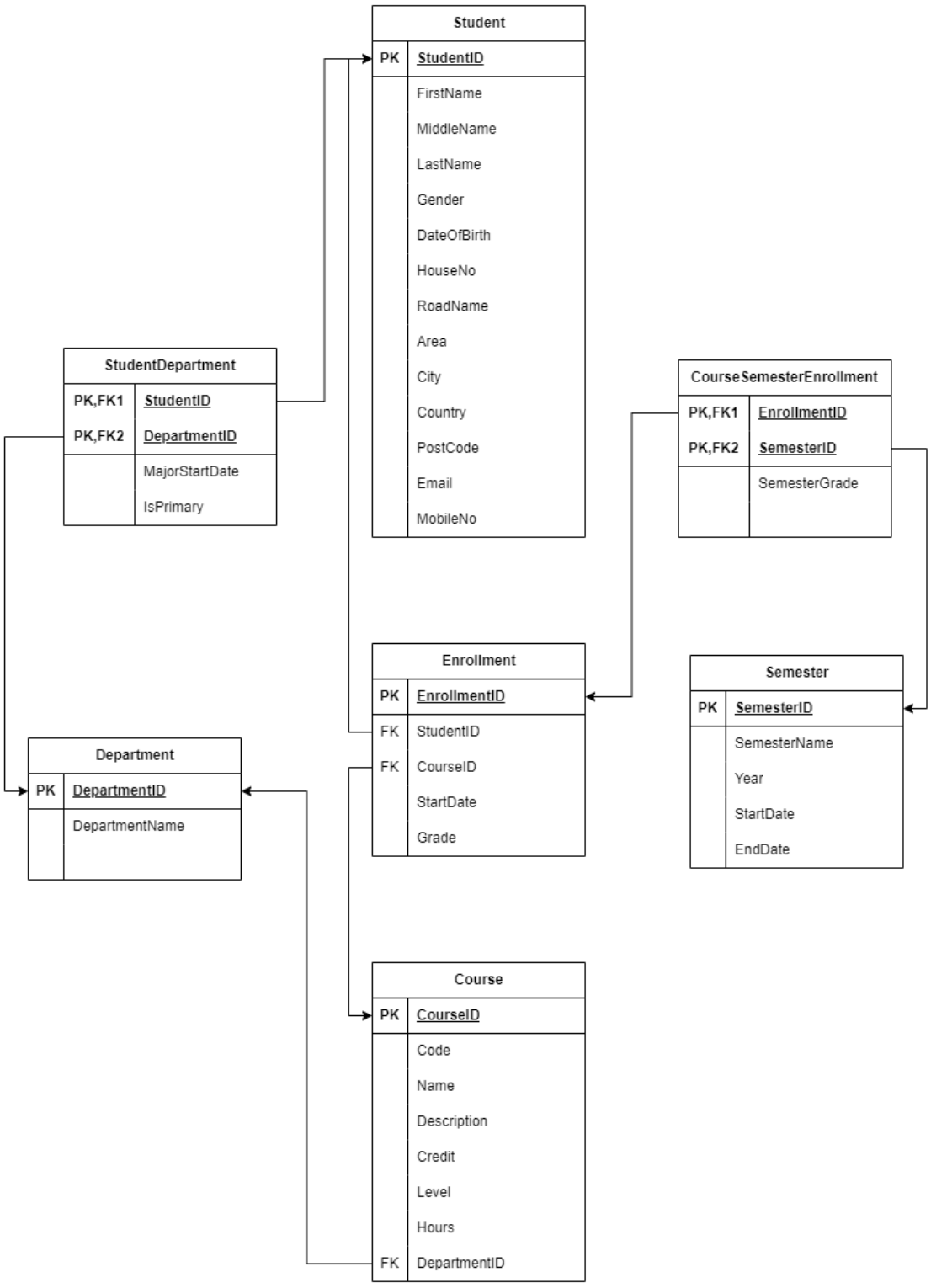


Figure 2: Logical Diagram

2.2 Mongo DB Design

Goal: Enhance flexibility and scalability, especially for applications with large data volumes, distributed architecture, and no strict schema requirements.

Structure: Uses collections and documents instead of tables. A document can store related data that is often accessed together.

Schema-less: No defined schema requirements. Documents in the same collection can have different fields, offering data representation flexibility.

Denormalisation:

Often employs denormalised data models, sometimes embedding related data directly within a single document, reducing the need for joins.

Schema in MongoDB

Purpose: Document versioning is employed to enhance the flexibility of maintenance processes. In relational databases, implementing structural changes often incurs high maintenance costs because these changes must propagate to associated applications. However, by utilising versioning, applications can reference documents by specific versions, thereby circumventing the need for immediate updates at the application level whenever database structures are modified. This approach allows smoother transitions and reduces the direct impact of database changes on application functionality.

Embedding Documents

Advantages:

Performance: Retrieval of all related data in a single database operation without joins.

Atomicity: Changes to a single document are atomic, which is crucial for maintaining data consistency without complex transactions.

Disadvantages:

Data Redundancy: Duplication of embedded data can lead to inconsistencies.

Linking (Referencing) Documents:

Advantages:

Normalisation: Helps avoid data duplication and maintain data integrity by referencing data stored in different documents.

Scalability: Handling large datasets and relationships that might span vast collections is easier.

Disadvantages:

Performance: Requires multiple queries or complex aggregations to retrieve related data, affecting performance.

Flexibility and Efficiency of NoSQL Documents

Flexibility: Documents in MongoDB can contain varied structures, allowing them to adapt quickly to the application's needs.

Less Reliance on Joins:

By storing related data within the same document, MongoDB can retrieve full sets of related data without joining, which is particularly beneficial for read-heavy applications.

Considering the design factors previously discussed, the University Student Data Management system has been structured using a NoSQL approach with four primary collections: **Students**, **Courses**, **Departments**, and **Enrollments**. The document structures for each collection are detailed below.

2.2.1 Students

```
{
  "_id": ObjectId("unique_student_id"),
  "schema" : 1,
  "first_name": "Liam",
  "middle_name": "Tāne",
  "last_name": "Taylor",
  "gender": "Male",
  "dob": "2001-04-23",
  "address": {
    "house_no": "87",
    "road_name": "Queen St",
    "area": "CBD",
    "city": "Auckland",
    "country": "New Zealand",
    "postcode": "1010"
  },
  "contact_details": {
    "mobile_no": "021-987-6543",
    "email": "liamtaylor@nzuni.ac.nz"
  },
  "departments": [
    {
      "department_id": ObjectId("department_id"),
      "is_primary": true,
      "major_start_date": "2020-01-15"
    }
  ]
}
```

Figure 3: Sample Document for Students Collection

2.2.2 Courses

```
{
  "_id": ObjectId("unique_course_id"),
  "schema" : 1,
  "code": "CS101",
  "name": "Introduction to Computer Science",
  "description": "A foundational course in computer science.",
  "credit": 3,
  "level": 100,
  "hours": 45,
  "department_id": ObjectId("department_id")  // Link to the Department document
}
```

Figure 4: Sample Document for Courses Collection

2.2.3 Departments

```
{
  "_id": ObjectId("department_id"),
  "schema" : 1,
  "dname": "Computer Science"
}
```

Figure 5: Sample Document for Department Collection

2.2.4 Enrollments

```
{
  "_id": ObjectId("unique_enrollment_id"),
  "schema" : 1,
  "student_id": ObjectId("unique_student_id"),
  "course_id": ObjectId("unique_course_id"),
  "semester": [
      {
        "semester_name": "2024-August",
        "year": 2024,
        "start_date": "2024-08-01",
        "end_date": "2024-12-15",
        "semester_grade": "A"
      }
  ],
  "grade": "A"
}
```

Figure 6: Sample Document for Enrollments Collection

2.2.5 Design Decisions

- **Nested Structures for Composite Attributes:**

  The composite attributes from the ER diagram, such as address and contact_details in the student document, have been structured as nested documents. This approach leverages MongoDB's document-oriented model to encapsulate related data effectively, enhancing readability and data access patterns.

- **Centralized Department Information:**

  The information about the departments is linked to both students and courses. This linkage ensures that all related entities refer to a single source of truth for department data, thus maintaining data integrity and reducing redundancy.

- **Embedded Semester Information in Enrollment:**
  In the design, semester details are embedded directly within the Enrollment documents. This embedding strategy centralises all relevant Enrollment and semester information within a single document, simplifying data retrieval processes and reducing the need for joins.
- **Separate Enrollment Collection:**
  Considering the potential growth in the volume of enrollment data over time, enrollments are maintained as a separate collection rather than embedded within the Student documents. This separation addresses potential issues related to document size and performance degradation. Storing enrollments independently allows for more efficient data management practices, such as archiving older records and optimising storage usage across the database system.

3 Task 2: Implementation

3.1 MySQL Implementation

MySQL scripts are available in the "mysql_script.sql" file and can be executed on any MySQL instance. This script is designed to create the "final_DB" database and includes steps for table creation, data population, and querying in accordance with the assessment's requirements. Screenshots of these scripts are included in this document to preserve the formatting of the developer tool, making them easier to read and understand.

3.1.1 Department

```
CREATE TABLE Department (
  DepartmentID INT  PRIMARY KEY,
  DepartmentName VARCHAR(255) NOT NULL
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 7: Structure of Department Table

```
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| DepartmentID   | int          | NO   | PRI | NULL    |       |
| DepartmentName | varchar(255) | NO   |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
2 rows in set (0.04 sec)
```

Figure 8: Department Table Description

```
INSERT INTO Department (DepartmentID, DepartmentName) VALUES
(1, 'Computer Engineering'),
(2, 'Computer Science'),
(3, 'Mechanical Engineering'),
(4, 'Electrical Engineering'),
(5, 'Humanities');
```

Figure 9: Insert Scripts for Department Table

```
+--------------+-----------------------+
| DepartmentID | DepartmentName        |
+--------------+-----------------------+
|            1 | Computer Engineering  |
|            2 | Computer Science      |
|            3 | Mechanical Engineering |
|            4 | Electrical Engineering |
|            5 | Humanities            |
+--------------+-----------------------+
```

Figure 10: Select query of Department Table

3.1.2 Student

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    MiddleName VARCHAR(50),
    LastName VARCHAR(50) NOT NULL,
    Gender ENUM('Male', 'Female', 'Other') NOT NULL,
    DateOfBirth DATE NOT NULL,
    HouseNo VARCHAR(50),
    RoadName VARCHAR(255),
    Area VARCHAR(255),
    City VARCHAR(50),
    Country VARCHAR(50),
    PostCode VARCHAR(10),
    Email VARCHAR(100),
    MobileNo VARCHAR(15)
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 11: Structure of Student Table

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	
FirstName	varchar(50)	NO		NULL	
MiddleName	varchar(50)	YES		NULL	
LastName	varchar(50)	NO		NULL	
Gender	enum('Male','Female','Other')	NO		NULL	
DateOfBirth	date	NO		NULL	
HouseNo	varchar(50)	YES		NULL	
RoadName	varchar(255)	YES		NULL	
Area	varchar(255)	YES		NULL	
City	varchar(50)	YES		NULL	
Country	varchar(50)	YES		NULL	
PostCode	varchar(10)	YES		NULL	
Email	varchar(100)	YES		NULL	
MobileNo	varchar(15)	YES		NULL	

Figure 12: Student Table Description

```
INSERT INTO Student (StudentID, FirstName, MiddleName, LastName, Gender, DateOfBirth, HouseNo, RoadName, Area, City, Country, PostCode, Email, MobileNo) VALUES
(1, 'Liam', 'John', 'Wilson', 'Male', '2001-04-23', '12', 'High Street', 'Central', 'Wellington', 'New Zealand', '6011', 'liamwilson@email.com', '021-123-4567'),
(2, 'Olivia', 'Jane', 'Smith', 'Female', '2002-05-19', '34', 'King Street', 'East', 'Auckland', 'New Zealand', '1010', 'olivia.smith@email.com', '022-234-5678'),
(3, 'Noah', 'James', 'Brown', 'Male', '2000-03-30', '56', 'Queen Street', 'West', 'Christchurch', 'New Zealand', '8011', 'noah.brown@email.com', '023-345-6789'),
(4, 'Emma', 'Charlotte', 'Taylor', 'Female', '2003-07-22', '78', 'Victoria Street', 'South', 'Dunedin', 'New Zealand', '9016', 'emma.taylor@email.com', '024-456-7890'),
(5, 'Sophia', 'Marie', 'Jones', 'Female', '1999-12-13', '90', 'Elizabeth Street', 'North', 'Hamilton', 'New Zealand', '3204', 'sophia.jones@email.com', '025-567-8901'),
(6, 'Mason', 'Alexander', 'Lee', 'Male', '2002-11-05', '100', 'Lincoln Road', 'Central', 'Tauranga', 'New Zealand', '3110', 'mason.lee@email.com', '026-678-9012'),
(7, 'Isabella', 'Ann', 'Wilson', 'Female', '2001-09-15', '110', 'Lake Road', 'East', 'Palmerston North', 'New Zealand', '4410', 'isabella.wilson@email.com', '027-789-0123'),
(8, 'Lucas', 'George', 'Davis', 'Male', '2002-02-20', '120', 'Park Avenue', 'West', 'Nelson', 'New Zealand', '7010', 'lucas.davis@email.com', '028-890-1234'),
(9, 'Mia', 'Elizabeth', 'White', 'Female', '2003-01-17', '130', 'Church Street', 'South', 'Invercargill', 'New Zealand', '9810', 'mia.white@email.com', '029-901-2345'),
(10, 'Ethan', 'Robert', 'Martin', 'Male', '1998-08-25', '140', 'Station Road', 'North', 'New Plymouth', 'New Zealand', '4310', 'ethan.martin@email.com', '030-012-3456');
```

Figure 13: Insert Scripts for Student Table

StudentID	FirstName	MiddleName	LastName	Gender	DateOfBirth	HouseNo	RoadName	Area	City	Country	PostCode	Email	MobileNo
1	Liam	John	Wilson	Male	2001-04-23	12	High Street	Central	Wellington	New Zealand	6011	liamwilson@email.com	021-123-4567
2	Olivia	Jane	Smith	Female	2002-05-19	34	King Street	East	Auckland	New Zealand	1010	olivia.smith@email.com	022-234-5678
3	Noah	James	Brown	Male	2000-03-30	56	Queen Street	West	Christchurch	New Zealand	8011	noah.brown@email.com	023-345-6789
4	Emma	Charlotte	Taylor	Female	2003-07-22	78	Victoria Street	South	Dunedin	New Zealand	9016	emma.taylor@email.com	024-456-7890
5	Sophia	Marie	Jones	Female	1999-12-13	90	Elizabeth Street	North	Hamilton	New Zealand	3204	sophia.jones@email.com	025-567-8901
6	Mason	Alexander	Lee	Male	2002-11-05	100	Lincoln Road	Central	Tauranga	New Zealand	3110	mason.lee@email.com	026-678-9012
7	Isabella	Ann	Wilson	Female	2001-09-15	110	Lake Road	East	Palmerston North	New Zealand	4410	isabella.wilson@email.com	027-789-0123
8	Lucas	George	Davis	Male	2002-02-20	120	Park Avenue	West	Nelson	New Zealand	7010	lucas.davis@email.com	028-890-1234
9	Mia	Elizabeth	White	Female	2003-01-17	130	Church Street	South	Invercargill	New Zealand	9810	mia.white@email.com	029-901-2345
10	Ethan	Robert	Martin	Male	1998-08-25	140	Station Road	North	New Plymouth	New Zealand	4310	ethan.martin@email.com	030-012-3456

Figure 14: Select query of Student Table

3.1.3 Course

```
CREATE TABLE Course (
  CourseID INT PRIMARY KEY,
  Code VARCHAR(20) NOT NULL,
  Name VARCHAR(255) NOT NULL,
  Description TEXT,
  Credit INT NOT NULL,
  Level VARCHAR(10) NOT NULL,
  Hours INT NOT NULL,
  DepartmentID INT,
  FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)
    ON UPDATE CASCADE
    ON DELETE SET NULL
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 15: Structure of Course Table

Field	Type	Null	Key	Default	Extra
CourseID	int	NO	PRI	NULL	
Code	varchar(20)	NO		NULL	
Name	varchar(255)	NO		NULL	
Description	text	YES		NULL	
Credit	int	NO		NULL	
Level	varchar(10)	NO		NULL	
Hours	int	NO		NULL	
DepartmentID	int	YES	MUL	NULL	

Figure 16: Course Table Description

```
INSERT INTO Course (CourseID, Code, Name, Description, Credit, Level, Hours, DepartmentID) VALUES
(1, 'GP106', 'Introduction to Programming', 'Learn the basics of programming.', 3, 100, 45, 1),
(2, 'CS205', 'Data Structures', 'Advanced data organization techniques.', 3, 200, 30, 2),
(3, 'CS303', 'Operating Systems', 'In-depth study of modern operating systems.', 4, 300, 60, 2),
(4, 'ME101', 'Intro to Mechanics', 'Foundational mechanics for engineers.', 3, 100, 45, 3),
(5, 'EE202', 'Circuit Analysis', 'Analyzing complex electrical circuits.', 4, 200, 50, 4),
(6, 'HU103', 'World History', 'A survey of world historical events and figures.', 3, 100, 40, 5),
(7, 'CS404', 'Machine Learning', 'Techniques and applications of machine learning.', 4, 400, 60, 2);
```

Figure 17: Insert Scripts for Course Table

CourseID	Code	Name	Description	Credit	Level	Hours	DepartmentID
1	GP106	Introduction to Programming	Learn the basics of programming.	3	100	45	1
2	CS205	Data Structures	Advanced data organization techniques.	3	200	30	2
3	CS303	Operating Systems	In-depth study of modern operating systems.	4	300	60	2
4	ME101	Intro to Mechanics	Foundational mechanics for engineers.	3	100	45	3
5	EE202	Circuit Analysis	Analyzing complex electrical circuits.	4	200	50	4
6	HU103	World History	A survey of world historical events and figures.	3	100	40	5
7	CS404	Machine Learning	Techniques and applications of machine learning.	4	400	60	2

Figure 18: Select query of Course Table

3.1.4 Semester

```
CREATE TABLE Semester (
  SemesterID INT PRIMARY KEY,
  SemesterName VARCHAR(100) NOT NULL,
  Year INT NOT NULL,
  StartDate DATE NOT NULL,
  EndDate DATE NOT NULL
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 19: Structure of Semester Table

Field	Type	Null	Key	Default	Extra
SemesterID	int	NO	PRI	NULL	
SemesterName	varchar(100)	NO		NULL	
Year	int	NO		NULL	
StartDate	date	NO		NULL	
EndDate	date	NO		NULL	

Figure 20: Semester Table Description

```sql
INSERT INTO Semester (SemesterID, SemesterName, Year, StartDate, EndDate) VALUES
(1, '2023-Jan', 2023, '2023-01-01', '2023-03-31'),
(2, '2023-April', 2023, '2023-04-01', '2023-06-30'),
(3, '2023-July', 2023, '2023-07-01', '2023-09-30'),
(4, '2023-Oct', 2023, '2023-10-01', '2023-12-31'),
(5, '2024-Jan', 2024, '2024-01-01', '2024-03-31'),
(6, '2024-April', 2024, '2024-04-01', '2024-06-30'),
(7, '2024-July', 2024, '2024-07-01', '2024-09-30'),
(8, '2024-Oct', 2024, '2024-10-01', '2024-12-31');
```

Figure 21: Insert Scripts for Semester Table

SemesterID	SemesterName	Year	StartDate	EndDate
1	2023-Jan	2023	2023-01-01	2023-03-31
2	2023-April	2023	2023-04-01	2023-06-30
3	2023-July	2023	2023-07-01	2023-09-30
4	2023-Oct	2023	2023-10-01	2023-12-31
5	2024-Jan	2024	2024-01-01	2024-03-31
6	2024-April	2024	2024-04-01	2024-06-30
7	2024-July	2024	2024-07-01	2024-09-30
8	2024-Oct	2024	2024-10-01	2024-12-31

Figure 22: Select query of Semester Table

3.1.5 Enrollment

```sql
CREATE TABLE Enrollment (
  EnrollmentID INT PRIMARY KEY,
  StudentID INT NOT NULL,
  CourseID INT NOT NULL,
  StartDate DATE NOT NULL,
  Grade VARCHAR(2),
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 23: Structure of Enrollment Table

Field	Type	Null	Key	Default	Extra
EnrollmentID	int	NO	PRI	NULL	
StudentID	int	NO	MUL	NULL	
CourseID	int	NO	MUL	NULL	
StartDate	date	NO		NULL	
Grade	varchar(2)	YES		NULL	

Figure 24: Enrollment Table Description

```
INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID, StartDate, Grade) VALUES
-- Student 1 enrolls in 2 courses, extending into 2024
(1, 1, 1, '2023-01-01', 'A'),
(2, 1, 3, '2023-04-01', 'A'),
(3, 1, 2, '2024-01-01', 'A'),
-- Student 2 enrolls in 2 different courses
(4, 2, 1, '2023-04-01', 'B'),
(5, 2, 2, '2024-01-01', 'B'),
-- Student 3 enrolls in the same course across multiple semesters
(6, 3, 1, '2023-01-01', 'C'),
(7, 3, 1, '2023-04-01', 'B'),
(8, 3, 2, '2024-04-01', 'B'),
-- Student 4 enrolls in multiple courses
(9, 4, 2, '2023-04-01', 'A'),
(10, 4, 4, '2023-10-01', 'A'),
-- Student 5 enrolls in multiple courses
(11, 5, 2, '2023-04-01', 'B'),
(12, 5, 5, '2024-04-01', 'B'),
-- Additional enrollments for students 6 and 7
(13, 6, 3, '2023-01-01', 'C'),
(14, 7, 4, '2023-04-01', 'D');
```

Figure 25: Insert Scripts for Enrollment Table

EnrollmentID	StudentID	CourseID	StartDate	Grade
1	1	1	2023-01-01	A
2	1	3	2023-04-01	A
3	1	2	2024-01-01	A
4	2	1	2023-04-01	B
5	2	2	2024-01-01	B
6	3	1	2023-01-01	C
7	3	1	2023-04-01	B
8	3	2	2024-04-01	B
9	4	2	2023-04-01	A
10	4	4	2023-10-01	A
11	5	2	2023-04-01	B
12	5	5	2024-04-01	B
13	6	3	2023-01-01	C
14	7	4	2023-04-01	D

Figure 26: Select query of Enrollment Table

3.1.6 StudentDepartment

```
CREATE TABLE StudentDepartment (
  StudentID INT,
  DepartmentID INT,
  MajorStartDate DATE NOT NULL,
  IsPrimary BOOLEAN NOT NULL,
  PRIMARY KEY (StudentID, DepartmentID),
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 27: Structure of StudentDepartment Table

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	
DepartmentID	int	NO	PRI	NULL	
MajorStartDate	date	NO		NULL	
IsPrimary	tinyint(1)	NO		NULL	

Figure 28: StudentDepartment Table Description

```
INSERT INTO StudentDepartment (StudentID, DepartmentID, MajorStartDate, IsPrimary) VALUES
(1, 1, '2023-01-01', TRUE),
(2, 1, '2023-01-01', TRUE),
(3, 1, '2023-01-01', TRUE),
(1, 2, '2023-01-01', FALSE);  -- Example of a student with a secondary major
```

Figure 29: Insert Scripts for StudentDepartment Table

StudentID	DepartmentID	MajorStartDate	IsPrimary
1	1	2023-01-01	1
1	2	2023-01-01	0
2	1	2023-01-01	1
3	1	2023-01-01	1

Figure 30: Select query of StudentDepartment Table

3.1.7 CourseSemesterEnrollment

```
CREATE TABLE CourseSemesterEnrollment (
  EnrollmentID INT,
  SemesterID INT,
  SemesterGrade VARCHAR(2),
  PRIMARY KEY (EnrollmentID, SemesterID),
  FOREIGN KEY (EnrollmentID) REFERENCES Enrollment(EnrollmentID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (SemesterID) REFERENCES Semester(SemesterID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
)ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

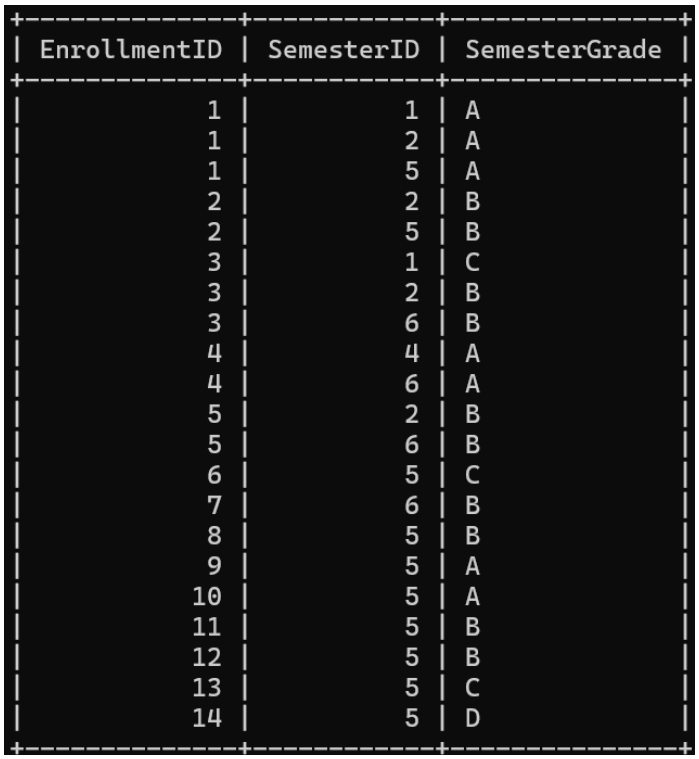
Figure 31: Structure of CourseSemesterEnrollment Table

Field	Type	Null	Key	Default	Extra
EnrollmentID	int	NO	PRI	NULL	
SemesterID	int	NO	PRI	NULL	
SemesterGrade	varchar(2)	YES		NULL	

Figure 32: CourseSemesterEnrollment Table Description

```
INSERT INTO CourseSemesterEnrollment (EnrollmentID, SemesterID, SemesterGrade) VALUES
(1, 1, 'A'),  -- Student 1 in Semester 2023-Jan
(1, 2, 'A'),  -- Continuation in Semester 2023-April
(1, 5, 'A'),  -- Continuing into Semester 2024-Jan
(2, 2, 'B'),  -- Student 2 in Semester 2023-April
(2, 5, 'B'),  -- Continuing into Semester 2024-Jan
(3, 1, 'C'),  -- Student 3 in Semester 2023-Jan
(3, 2, 'B'),  -- Continuation in Semester 2023-April
(3, 6, 'B'),  -- Continuing into Semester 2024-April
(4, 6, 'A'),  -- Student 4 in Semester 2024-April
(4, 4, 'A'),  -- Continuation in Semester 2023-Oct
(5, 2, 'B'),  -- Student 5 in Semester 2023-April
(5, 6, 'B'),  -- Continuing into Semester 2024-April
(6, 5, 'C'),  -- Continuing into Semester 2024-Jan
(7, 6, 'B'),  -- Continuing into Semester 2024-April
(8, 5, 'B'),  -- Continuing into Semester 2024-Jan
(9, 5, 'A'),  -- Continuing into Semester 2024-Jan
(10, 5, 'A'),  -- Continuing into Semester 2024-Jan
(11, 5, 'B'),  -- Continuing into Semester 2024-Jan
(12, 5, 'B'),  -- Continuing into Semester 2024-Jan
(13, 5, 'C'),  -- Continuing into Semester 2024-Jan
(14, 5, 'D');  -- Continuing into Semester 2024-Jan
```

Figure 33: Insert Scripts for CourseSemensterEnrollment Table



EnrollmentID	SemesterID	SemesterGrade
1	1	A
1	2	A
1	5	A
2	2	B
2	5	B
3	1	C
3	2	B
3	6	B
4	4	A
4	6	A
5	2	B
5	6	B
6	5	C
7	6	B
8	5	B
9	5	A
10	5	A
11	5	B
12	5	B
13	5	C
14	5	D

Figure 34: Select query of CourseSemesterEnrollment Table

3.2 MongoDB Implementation

MongoDB scripts are contained within the “mongo_scripts.txt” file, which can be executed sequentially through the MongoDB shell. These scripts are designed to establish the “FinalDB” database, followed by the creation of Collections and their respective Documents. The collections created include Departments, Students, Courses, and Enrollments. Due to space constraints in this document, full insert queries are not displayed here. Instead, selected portions of insert queries and partial results, including total document counts, are provided for reference. For complete query details, please refer to the “mongo_scripts.txt” file.

3.2.1 Departments

```
FinalDTS123> db.createCollection("Departments");
{ ok: 1 }
FinalDTS123> db.Departments.createIndex({"dname" : 1});
dname_1
FinalDTS123>

FinalDTS123>

FinalDTS123> db.Departments.insertMany([
...     { "schema": 1, "dname": "Computer Engineering"},
...     { "schema": 1, "dname": "Computer Science"},
...     { "schema": 1, "dname": "Mechanical Engineering"},
...     { "schema": 1, "dname": "Electrical Engineering"},
...     { "schema": 1, "dname": "Humanities"}
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("66377c4e1cc8e6149559ec7e"),
    '1': ObjectId("66377c4e1cc8e6149559ec7f"),
    '2': ObjectId("66377c4e1cc8e6149559ec80"),
    '3': ObjectId("66377c4e1cc8e6149559ec81"),
    '4': ObjectId("66377c4e1cc8e6149559ec82")
  }
}
FinalDTS123>

FinalDTS123> db.Departments.countDocuments();
5
FinalDTS123>
```

Figure 35: Departments

```
FinalDTS123> var depComEng = db.Departments.findOne({"dname" : "Computer Engineering"},{_id: 1});

FinalDTS123> var depComSci = db.Departments.findOne({"dname" : "Computer Science"},{_id: 1});

FinalDTS123> var depMacEng = db.Departments.findOne({"dname" : "Mechanical Engineering"},{_id: 1});

FinalDTS123> var depEleEng = db.Departments.findOne({"dname" : "Electrical Engineering"},{_id: 1});

FinalDTS123> var depHum    = db.Departments.findOne({"dname" : "Humanities"},{_id: 1});
```

Figure 36: Department Document's Object_ID extract

Variables have been defined to capture the ObjectIDs associated with specific documents, facilitating future scripts that will link to department documents. Typically, not all ObjectIDs are stored in variables; rather, references are made only to objects that are actively being manipulated. However, for the purpose of this instance, where the goal is to comprehensively capture data into documents, this approach of capturing ObjectIDs in variables has been adopted to streamline subsequent data linking and manipulation tasks.

3.2.2 Students

```
FinalDTS123> db.createCollection("Students");
{ ok: 1 }
FinalDTS123> db.Students.createIndex({"first_name" : 1});
first_name_1
FinalDTS123> db.Students.createIndex({"last_name" : 1});
last_name_1
FinalDTS123> db.Students.createIndex({"contact_details.email" : 1});
contact_details.email_1
FinalDTS123>
```

Figure 37: Student Collection & Index Creation

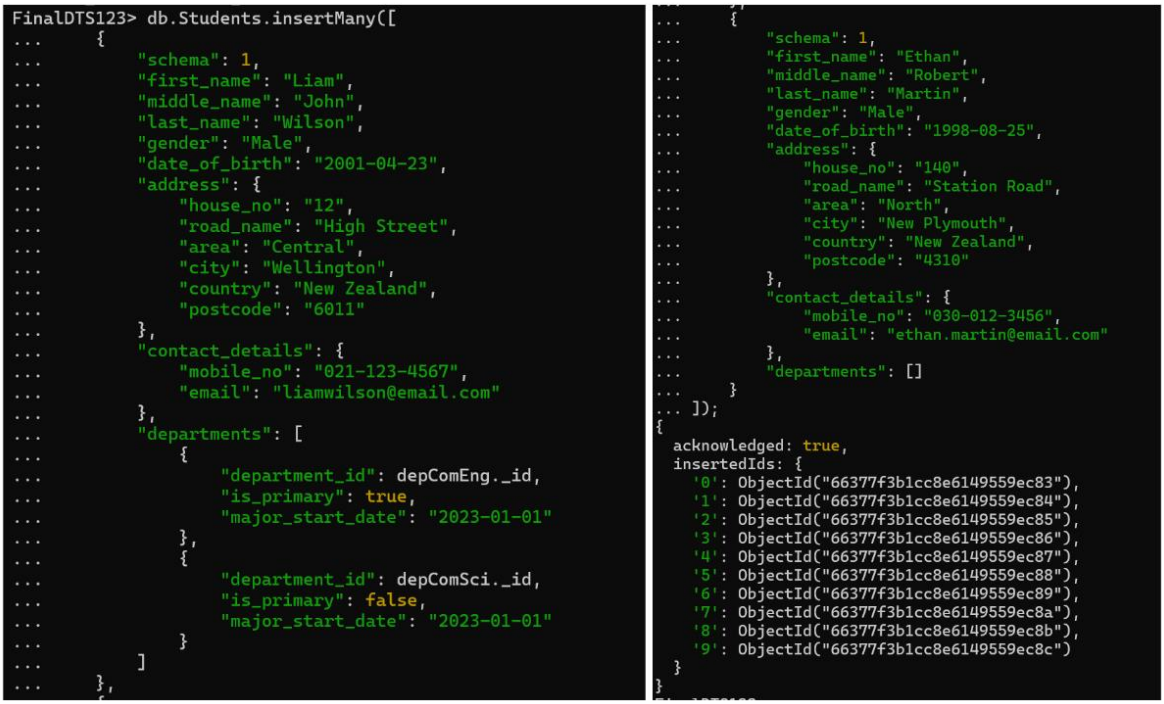


Figure 38: Sample Student Document Insert

Refer to the script file for complete code. The depComEng is the variable declared to capture document ID.

Figure 39: Student documents count & variables

3.2.3 Courses

```
FinalDTS123> db.createCollection("Courses");
{ ok: 1 }
FinalDTS123> db.Courses.createIndex({"code" : 1});
code_1
FinalDTS123> db.Courses.createIndex({"name" : 1});
name_1
FinalDTS123>
```

Figure 40: Courses Collection

```
db.Courses.insertMany([
...     {
...         "code": "GP106",
...         "name": "Introduction to Programming",
...         "description": "Learn the basics of programming.",
...         "credit": 3,
...         "level": 100,
...         "hours": 45,
...         "department_id": depComEng._id
...     },
```

```
...     {
...         "code": "HU103",
...         "name": "World History",
...         "description": "A survey of world historical events and figures.",
...         "credit": 3,
...         "level": 100,
...         "hours": 40,
...         "department_id": depHum._id
...     }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("663782db1cc8e6149559ec8d"),
    '1': ObjectId("663782db1cc8e6149559ec8e"),
    '2': ObjectId("663782db1cc8e6149559ec8f"),
    '3': ObjectId("663782db1cc8e6149559ec90"),
    '4': ObjectId("663782db1cc8e6149559ec91"),
    '5': ObjectId("663782db1cc8e6149559ec92"),
    '6': ObjectId("663782db1cc8e6149559ec93")
  }
}
```

Figure 41: Insert Documents for Courses

```
FinalDTS123> var corGP106 = db.Courses.findOne({"code": "GP106"}, {_id: 1});

FinalDTS123> var corCS205 = db.Courses.findOne({"code": "CS205"}, {_id: 1});

FinalDTS123> var corCS303 = db.Courses.findOne({"code": "CS303"}, {_id: 1});

FinalDTS123> var corCS404 = db.Courses.findOne({"code": "CS404"}, {_id: 1});

FinalDTS123> var corME101 = db.Courses.findOne({"code": "ME101"}, {_id: 1});

FinalDTS123> var corEE202 = db.Courses.findOne({"code": "EE202"}, {_id: 1});

FinalDTS123> var corHU103 = db.Courses.findOne({"code": "HU103"}, {_id: 1});

FinalDTS123> db.Courses.countDocuments();
7
```

Figure 42: Course document Count & Variables

3.2.4 Enrollments

```
FinalDTS123> db.createCollection("Enrollments");
{ ok: 1 }
FinalDTS123> db.Enrollments.createIndex({"semester.semester_name" : 1});
semester.semester_name_1
FinalDTS123>
```

Figure 43: Enrollment Collection & Index Creation

```
FinalDTS123> db.Enrollments.insertMany([
...     {
...         "schema": 1,
...         "student_id": stdLiam._id,
...         "course_id": corGP106._id,
...         "semester": [
...             {
...                 "semester_name": "2023-Jan",
...                 "year": 2023,
...                 "start_date": ISODate("2023-01-01"),
...                 "end_date": ISODate("2023-03-31"),
...                 "semester_grade": "A"
...             },
...             {
...                 "semester_name": "2023-April",
...                 "year": 2023,
...                 "start_date": ISODate("2023-04-01"),
...                 "end_date": ISODate("2023-06-30"),
...                 "semester_grade": "A"
...             },
...             {
...                 "semester_name": "2024-Jan",
...                 "year": 2024,
...                 "start_date": ISODate("2024-01-01"),
...                 "end_date": ISODate("2024-03-31"),
...                 "semester_grade": "A"
...             }
...         ],
...         "grade": "A"
...     },
...                 "end_date": ISODate("2023-06-30"),
...                 "semester_grade": "B"
...             },
...             {
...                 "semester_name": "2024-April",
...                 "year": 2024,
...                 "start_date": ISODate("2024-04-01"),
...                 "end_date": ISODate("2024-06-30"),
...                 "semester_grade": "B"
...             }
...         ],
...         "grade": "B"
...     }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("663828236e47134e8000878a"),
    '1': ObjectId("663828236e47134e8000878b"),
    '2': ObjectId("663828236e47134e8000878c"),
    '3': ObjectId("663828236e47134e8000878d"),
    '4': ObjectId("663828236e47134e8000878e"),
    '5': ObjectId("663828236e47134e8000878f"),
    '6': ObjectId("663828236e47134e80008790"),
    '7': ObjectId("663828236e47134e80008791"),
    '8': ObjectId("663828236e47134e80008792"),
    '9': ObjectId("663828236e47134e80008793"),
    '10': ObjectId("663828236e47134e80008794"),
    '11': ObjectId("663828236e47134e80008795"),
    '12': ObjectId("663828236e47134e80008796")
  }
}
```

Figure 44: Enrollment Document Insert

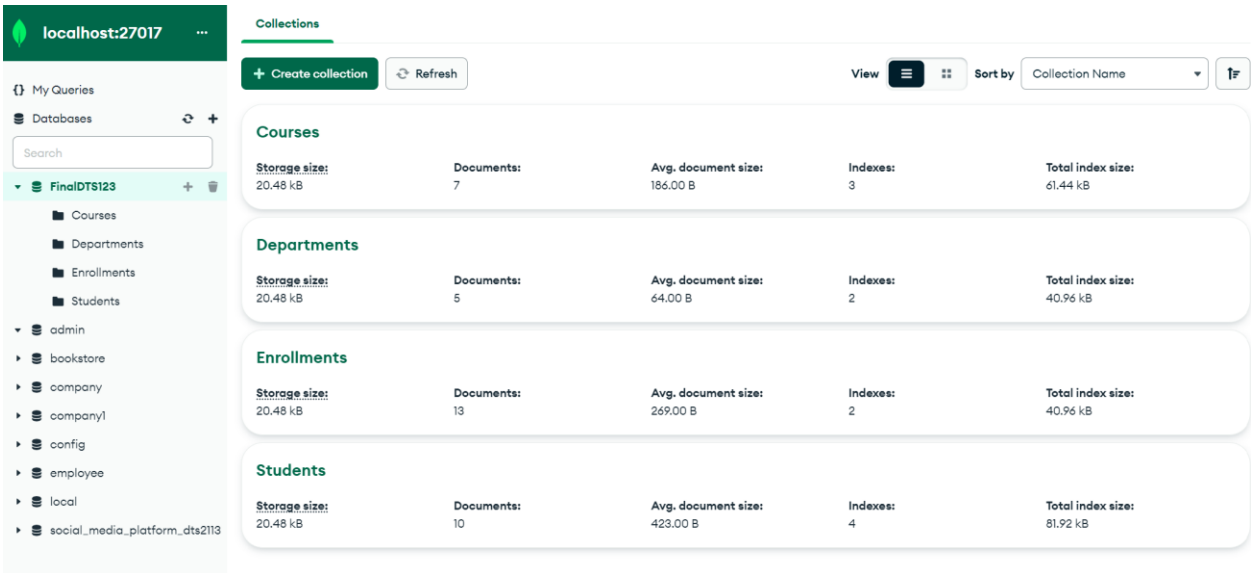


Figure 45: Mongo Compass View After Capturing

4 Task 3: Querying the Database

All SQL for MySQL and scripts for MongoDB are attached to script files.

4.1 Query 1:

Retrieve all students majoring in “Computer Engineering”.

4.1.1 MySQL

```
SELECT
    s.StudentID,
    s.FirstName,
    s.MiddleName,
    s.LastName,
    s.Gender,
    s.DateOfBirth,
    s.HouseNo,
    s.RoadName,
    s.Area,
    s.City,
    s.Country,
    s.PostCode,
    s.Email,
    s.MobileNo
FROM
    Student s
JOIN
    StudentDepartment sd ON s.StudentID = sd.StudentID
JOIN
    Department d ON sd.DepartmentID = d.DepartmentID
WHERE
    d.DepartmentName = 'Computer Engineering';
```

Figure 46: Query 1 - MySQL Script

StudentID	FirstName	MiddleName	LastName	Gender	DateOfBirth	HouseNo	RoadName	Area	City	Country	PostCode	Email	MobileNo
1	Liam	John	Wilson	Male	2001-04-23	12	High Street	Central	Wellington	New Zealand	6011	liamwilson@email.com	021-123-4567
2	Olivia	Jane	Smith	Female	2002-05-19	34	King Street	East	Auckland	New Zealand	1010	olivia.smith@email.com	022-234-5678
3	Noah	James	Brown	Male	2000-03-30	56	Queen Street	West	Christchurch	New Zealand	8011	noah.brown@email.com	023-345-6789

3 rows in set (0.00 sec)

Figure 47: Query 1 - MySQL Script Output

4.1.2 MongoDB

This is 2 steps process. First, find the department's object Id and student records based on the department object Id.

Find Department ID

```
FinalDTS123> var depComEng = db.Departments.findOne({"dname": "Computer Engineering"}, {_id: 1});
```

Figure 48: Query 1 - MongoDB Find Department ObjectId

Find and arrange the data extracted.

```
FinalDTS123> db.Students.aggregate([
...     {
...         $match: {
...             "departments.department_id": depComEng._id
...         }
...     },
...     {
...         $project: {
...             _id: 0,
...             first_name: 1,
...             middle_name: 1,
...             last_name: 1,
...             gender: 1,
...             date_of_birth: 1,
...             address: 1,
...             contact_details: 1,
...             departments: {
...                 $filter: {
...                     input: "$departments",
...                     as: "department",
...                     cond: { $eq: ["$$department.department_id", depComEng._id] }
...                 }
...             }
...         }
...     }
... ]).pretty();
```

Figure 49: Query 1 - MongoDB Get Students based on Department ObjectID

Figure 50: Query 1 - MongoDB Output

Results are matching with MySQL data extract.

4.2 Query 2:

List all courses along with the number of students enrolled in each.

4.2.1 MySQL

```
SELECT
    c.Code AS 'Course Code',
    c.Name AS 'Course Name',
    COUNT(distinct e.StudentID) AS 'Number of Students'
FROM
    Course c
LEFT JOIN
    Enrollment e ON c.CourseID = e.CourseID
GROUP BY
    c.CourseID, c.Code, c.Name;
```

Figure 51: Query 2 - MySQL Script

In the above query, the Student table is not used or joined because we can find the student unique reference at the Enrollment level, which is good enough for counting. Joining Student will further impact the performance but does not help with results.

```
+-------------+-----------------------------+--------------------+
| Course Code | Course Name                 | Number of Students |
+-------------+-----------------------------+--------------------+
| GP106       | Introduction to Programming |                  3 |
| CS205       | Data Structures             |                  5 |
| CS303       | Operating Systems           |                  2 |
| ME101       | Intro to Mechanics          |                  2 |
| EE202       | Circuit Analysis            |                  1 |
| HU103       | World History               |                  0 |
| CS404       | Machine Learning            |                  0 |
+-------------+-----------------------------+--------------------+
7 rows in set (0.22 sec)
```

Figure 52: Query 2 - MySQL Script Output

4.2.2 MongoDB

```
FinalDTS123> db.Courses.aggregate([
...     {
...         $lookup: {
...             from: "Enrollments",  // The collection to join with
...             localField: "_id",    // Field from the Courses collection
...             foreignField: "course_id",  // Field from the Enrollments collection
...             as: "enrollments"     // The array to which the joined documents will be added
...         }
...     },
...     {
...         $project: {
...             _id: 0,
...             code: 1,
...             name: 1,
...             numberOfStudents: { $size: "$enrollments" }  // Counts the number of entries in the enrollments array
...         }
...     }
... ]).pretty();
```

Figure 53: Query 2 - MongoDB Query Script

```
[
  {
    code: 'GP106',
    name: 'Introduction to Programming',
    numberOfStudents: 3
  },
  { code: 'CS205', name: 'Data Structures', numberOfStudents: 5 },
  { code: 'CS303', name: 'Operating Systems', numberOfStudents: 2 },
  { code: 'CS404', name: 'Machine Learning', numberOfStudents: 0 },
  { code: 'ME101', name: 'Intro to Mechanics', numberOfStudents: 2 },
  { code: 'EE202', name: 'Circuit Analysis', numberOfStudents: 1 },
  { code: 'HU103', name: 'World History', numberOfStudents: 0 }
]
FinalDTS123>
```

Figure 54: Query 2 - MongoDB Script Output

Return results are matched with MySQL results.

4.3 Query 3:

Find the average grade of students in the ‘GP106’ course for the ‘2024-April’ semester.

4.3.1 MySQL

```
SELECT
    c.Code,
    s.SemesterName,
    AVG(
        CASE e.Grade
            WHEN 'F' THEN 20
            WHEN 'D' THEN 40
            WHEN 'C' THEN 50
            WHEN 'B' THEN 60
            WHEN 'A' THEN 70
            ELSE 0
        END
    ) AS 'Average Grade'
FROM
    Course c
JOIN
    Enrollment e ON c.CourseID = e.CourseID
JOIN
    CourseSemesterEnrollment cse ON e.EnrollmentID = cse.EnrollmentID
JOIN
    Semester s ON cse.SemesterID = s.SemesterID
WHERE
    c.Code = 'GP106'
    AND s.SemesterName = '2024-April';
```

Figure 55: Query 3 - MySQL Script 1

As per the database design, overall grades are stored in the Enrollment table; therefore, grades in the Enrollment table are used for calculation. It was assumed that grade F = 20, D = 40, C = 50, B = 60 and A = 70 for this exercise.

```
+-------+--------------+---------------+
| Code  | SemesterName | Average Grade |
+-------+--------------+---------------+
| GP106 | 2024-April   |       60.0000 |
+-------+--------------+---------------+
1 row in set (0.00 sec)
```

Figure 56: Query 3 - MySQL Script Output 1

Next, SQL will turn the average number into a letter.

```sql
SELECT
    c.Code AS 'Course Code',
    s.SemesterName AS 'Semester Name',
    AVG(
        CASE e.Grade
            WHEN 'F' THEN 20
            WHEN 'D' THEN 40
            WHEN 'C' THEN 50
            WHEN 'B' THEN 60
            WHEN 'A' THEN 70
            ELSE 0
        END
    ) AS 'Average Numeric Grade',
    CASE
        WHEN AVG(
            CASE e.Grade
                WHEN 'F' THEN 20
                WHEN 'D' THEN 40
                WHEN 'C' THEN 50
                WHEN 'B' THEN 60
                WHEN 'A' THEN 70
                ELSE 0
            END
        ) BETWEEN 0 AND 29 THEN 'F'
        WHEN AVG(
            CASE e.Grade
        WHEN AVG(
            CASE e.Grade
        WHEN AVG(
            CASE e.Grade
        WHEN AVG(
            CASE e.Grade
        ELSE 'N/A'  -- Handles cases where an average cannot be calculated
    END AS 'Average Letter Grade'
FROM
    Course c
JOIN
    Enrollment e ON c.CourseID = e.CourseID
JOIN
    CourseSemesterEnrollment cse ON e.EnrollmentID = cse.EnrollmentID
JOIN
    Semester s ON cse.SemesterID = s.SemesterID
WHERE
    c.Code = 'GP106'
    AND s.SemesterName = '2024-April'
GROUP BY
    c.Code, s.SemesterName;
```

Figure 57: Query 3 - MySQL Script 2

```
+-------------+---------------+-----------------------+----------------------+
| Course Code | Semester Name | Average Numeric Grade | Average Letter Grade |
+-------------+---------------+-----------------------+----------------------+
| GP106       | 2024-April    |               60.0000 | B                    |
+-------------+---------------+-----------------------+----------------------+
1 row in set (0.14 sec)
```

Figure 58: Query 3 - MySQL Script Output 2

4.3.2 MongoDB

```
FinalDTS123> db.Enrollments.aggregate([
...     {
...         $lookup: {
...             from: "Courses",
...             localField: "course_id",
...             foreignField: "_id",
...             as: "course_details"
...         }
...     },
...     {
...         $unwind: "$course_details"
...     },
...     {
...         $match: {
...             "course_details.code": "GP106",
...             "semester.semester_name": "2024-April"
...         }
...     },
...     {
...         $project: {
...             grade: 1,
...             numericGrade: {
...                 $switch: {
...                     branches: [
...                         { case: { $eq: ["$grade", "A"] }, then: 70 },
...                         { case: { $eq: ["$grade", "B"] }, then: 60 },
...                         { case: { $eq: ["$grade", "C"] }, then: 50 },
...                         { case: { $eq: ["$grade", "D"] }, then: 40 },
...                         { case: { $eq: ["$grade", "F"] }, then: 20 }
...                     ],
...                     default: 0
...                 }
...             }
...         }
...     },
...     {
...         $group: {
...             _id: null,
...             averageGrade: { $avg: "$numericGrade" }
...         }
...     }
... ]).pretty();
```

Figure 59: Query 3 - MongoDB Scripts

```
[ { _id: null, averageGrade: 60 } ]
FinalDTS123>
```

Figure 60: Query 3 - MongoDB Script Output

Results match with MySQL results.

4.4 Query 4:

Retrieve all students who have an ‘A’ grade in any course.

4.4.1 MySQL

```
SELECT
    s.StudentID,
    s.FirstName,
    s.MiddleName,
    s.LastName,
    s.Gender,
    s.DateOfBirth,
    s.Email,
    s.MobileNo,
    c.Code AS 'Course Code',
    c.Name AS 'Course Name',
    e.Grade
FROM
    Student s
JOIN
    Enrollment e ON s.StudentID = e.StudentID
JOIN
    Course c ON e.CourseID = c.CourseID
WHERE
    e.Grade = 'A';
```

Figure 61: Query 4 - MySQL Script

StudentID	FirstName	MiddleName	LastName	Gender	DateOfBirth	Email	MobileNo	Course Code	Course Name	Grade
1	Liam	John	Wilson	Male	2001-04-23	liamwilson@email.com	021-123-4567	GP106	Introduction to Programming	A
1	Liam	John	Wilson	Male	2001-04-23	liamwilson@email.com	021-123-4567	CS303	Operating Systems	A
1	Liam	John	Wilson	Male	2001-04-23	liamwilson@email.com	021-123-4567	CS205	Data Structures	A
4	Emma	Charlotte	Taylor	Female	2003-07-22	emma.taylor@email.com	024-456-7890	CS205	Data Structures	A
4	Emma	Charlotte	Taylor	Female	2003-07-22	emma.taylor@email.com	024-456-7890	ME101	Intro to Mechanics	A

5 rows in set (0.00 sec)

Figure 62: Query 4 - MySQL Script Output

4.4.2 MongoDB

```
FinalDTS123> db.Enrollments.aggregate([
...     {
...         $match: {
...             grade: "A"  // Filter to include only enrollments where the grade is 'A'
...         }
...     },
...     {
...         $lookup: {
...             from: "Students",
...             localField: "student_id",
...             foreignField: "_id",
...             as: "student_info"
...         }
...     },
...     {
...         $lookup: {
...             from: "Courses",
...             localField: "course_id",
...             foreignField: "_id",
...             as: "course_info"
...         }
...     },
...     {
...         $unwind: "$student_info"  // Unwind the student_info array to simplify data structure
...     },
...     {
...         $unwind: "$course_info"  // Unwind the course_info array to simplify data structure
...     },
...     {
...         $project: {
...             _id: 0,  // Exclude the _id field for cleaner output
...             firstName: "$student_info.first_name",
...             middleName: "$student_info.middle_name",
...             lastName: "$student_info.last_name",
...             gender: "$student_info.gender",
...             email: "$student_info.contact_details.email",
...             courseCode: "$course_info.code",
...             courseName: "$course_info.name",
...             grade: "$grade"
...         }
...     }
... ]).pretty();
```

Figure 63: Query 4 - MongoDB Script

Figure 64: Query 4 - MongoDB Script Output

Results are matching with MySQL output.

4.5 Query 5:

Find all students who are enrolled in more than 3 courses.

4.5.1 MySQL

```sql
SELECT
    s.StudentID,
    s.FirstName,
    s.MiddleName,
    s.LastName,
    s.Gender,
    s.DateOfBirth,
    s.Email,
    s.MobileNo,
    COUNT(DISTINCT e.CourseID) AS 'Number of Courses'
FROM
    Student s
JOIN
    Enrollment e ON s.StudentID = e.StudentID
JOIN
    Course c ON e.CourseID = c.CourseID
GROUP BY
    s.StudentID, s.FirstName, s.MiddleName, s.LastName, s.Gender, s.DateOfBirth, s.Email, s.MobileNo
HAVING
    COUNT(DISTINCT e.CourseID) >3;
```

Figure 65: Query 5 - MySQL Script 1

The sample data I have created does not have any student with more than 3 courses under him. The following results are evidence for it.

```
mysql> SELECT
    ->     s.StudentID,
    ->     s.FirstName,
    ->     s.MiddleName,
    ->     s.LastName,
    ->     s.Gender,
    ->     s.DateOfBirth,
    ->     s.Email,
    ->     s.MobileNo,
    ->     COUNT(DISTINCT e.CourseID) AS 'Number of Courses'
    -> FROM
    ->     Student s
    -> JOIN
    ->     Enrollment e ON s.StudentID = e.StudentID
    -> JOIN
    ->     Course c ON e.CourseID = c.CourseID
    -> GROUP BY
    ->     s.StudentID, s.FirstName, s.MiddleName, s.LastName, s.Gender, s.DateOfBirth, s.Email, s.MobileNo
    -> HAVING
    ->     COUNT(DISTINCT e.CourseID) >3;
Empty set (0.05 sec)
```

Figure 66: Query 5 - MySQL Script Output 1

However, if I search for students with more than or equal to 3 courses.

```
SELECT
    s.StudentID,
    s.FirstName,
    s.MiddleName,
    s.LastName,
    s.Gender,
    s.DateOfBirth,
    s.Email,
    s.MobileNo,
    COUNT(DISTINCT e.CourseID) AS 'Number of Courses'
FROM
    Student s
JOIN
    Enrollment e ON s.StudentID = e.StudentID
JOIN
    Course c ON e.CourseID = c.CourseID
GROUP BY
    s.StudentID, s.FirstName, s.MiddleName, s.LastName, s.Gender, s.DateOfBirth, s.Email, s.MobileNo
HAVING
    COUNT(DISTINCT e.CourseID) >=3;
```

Figure 67: Query 5 - MySQL Script 2

```
+-----------+-----------+------------+----------+--------+-------------+---------------------+-------------+--------------------+
| StudentID | FirstName | MiddleName | LastName | Gender | DateOfBirth | Email               | MobileNo    | Number of Courses |
+-----------+-----------+------------+----------+--------+-------------+---------------------+-------------+--------------------+
|         1 | Liam      | John       | Wilson   | Male   | 2001-04-23  | liamwilson@email.com | 021-123-4567 |                  3 |
+-----------+-----------+------------+----------+--------+-------------+---------------------+-------------+--------------------+
1 row in set (0.00 sec)
```

Figure 68: Query 5 - MySQL Script Output 2

4.5.2 MongoDB

As explained in the MySQL section, since I don't have data for more than 3 courses, I will run the script for more than or equal conditions.

```
FinalDTS123> db.Students.aggregate([
...     {
...         $lookup: {
...             from: "Enrollments",
...             localField: "_id",
...             foreignField: "student_id",
...             as: "enrollments"
...         }
...     },
...     {
...         $project: {
...             firstName: "$first_name",
...             lastName: "$last_name",
...             email: "$contact_details.email",
...             totalCourses: { $size: "$enrollments" }
...         }
...     },
...     {
...         $match: {
...             totalCourses: { $gte: 3 }
...         }
...     }
... ]).pretty();
```

Figure 69: Query 5 - MongoDB Script

```
[
  {
    _id: ObjectId("6638275f6e47134e80008779"),
    firstName: 'Liam',
    lastName: 'Wilson',
    email: 'liamwilson@email.com',
    totalCourses: 3
  }
]
FinalDTS123>
```

Figure 70: Query 5 - MongoDB Script Output

Match with MySQL results.

4.6 Query 6:

Update grades for all students in the ‘GP106’ course for the ‘2024-April’ semester to ‘B’

Note: This will be performed in 3 steps. First, we will visualise the records based on update conditions. Then, I will update the records and finally revisit those records.

4.6.1 MySQL

```sql
SELECT
    s.FirstName,
    s.MiddleName,
    s.LastName,
    s.Gender,
    s.DateOfBirth,
    s.Email,
    c.Code AS 'Course Code',
    c.Name AS 'Course Name',
    sem.SemesterName AS 'Semester',
    e.Grade
FROM
    Student s
JOIN
    Enrollment e ON s.StudentID = e.StudentID
JOIN
    Course c ON e.CourseID = c.CourseID
JOIN
    CourseSemesterEnrollment cse ON e.EnrollmentID = cse.EnrollmentID
JOIN
    Semester sem ON cse.SemesterID = sem.SemesterID
WHERE
    c.Code = 'GP106'
    AND sem.SemesterName = '2024-April';
```

Figure 71: Query 6 - MySQL Select Script

```
+-----------+------------+----------+--------+-------------+--------------------+-------------+-----------------------------+-------------+-------+
| FirstName | MiddleName | LastName | Gender | DateOfBirth | Email              | Course Code | Course Name                 | Semester    | Grade |
+-----------+------------+----------+--------+-------------+--------------------+-------------+-----------------------------+-------------+-------+
| Olivia    | Jane       | Smith    | Female | 2002-05-19  | olivia.smith@email.com | GP106       | Introduction to Programming | 2024-April | B     |
| Noah      | James      | Brown    | Male   | 2000-03-30  | noah.brown@email.com   | GP106       | Introduction to Programming | 2024-April | B     |
+-----------+------------+----------+--------+-------------+--------------------+-------------+-----------------------------+-------------+-------+
2 rows in set (0.11 sec)
```

Figure 72: Query 6 - MySQL Select Script Output

In my case, students already have B grades for the condition given. Therefore, updating it to B grade again will not help us visualise the update query's effect. Therefore, I will change it to update to A Grade.

```sql
UPDATE Enrollment e
JOIN Course c ON e.CourseID = c.CourseID
JOIN CourseSemesterEnrollment cse ON e.EnrollmentID = cse.EnrollmentID
JOIN Semester sem ON cse.SemesterID = sem.SemesterID
SET e.Grade = 'A'
WHERE c.Code = 'GP106'
  AND sem.SemesterName = '2024-April';
```

Figure 73: Query 6 - MySQL Update Script

```
mysql> UPDATE Enrollment e
    -> JOIN Course c ON e.CourseID = c.CourseID
    -> JOIN CourseSemesterEnrollment cse ON e.EnrollmentID = cse.EnrollmentID
    -> JOIN Semester sem ON cse.SemesterID = sem.SemesterID
    -> SET e.Grade = 'A'
    -> WHERE c.Code = 'GP106'
    ->   AND sem.SemesterName = '2024-April';
Query OK, 2 rows affected (0.78 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

Figure 74: Query 6 - MySQL Update Script Output

The record set was visualised after the update.

```
+-----------+------------+----------+--------+-------------+----------------------+-------------+---------------------------+------------+-------+
| FirstName | MiddleName | LastName | Gender | DateOfBirth | Email                | Course Code | Course Name               | Semester   | Grade |
+-----------+------------+----------+--------+-------------+----------------------+-------------+---------------------------+------------+-------+
| Olivia    | Jane       | Smith    | Female | 2002-05-19  | olivia.smith@email.com | GP106       | Introduction to Programming | 2024-April | A     |
| Noah      | James      | Brown    | Male   | 2000-03-30  | noah.brown@email.com   | GP106       | Introduction to Programming | 2024-April | A     |
+-----------+------------+----------+--------+-------------+----------------------+-------------+---------------------------+------------+-------+
2 rows in set (0.09 sec)
```

Figure 75: Query 6 - MySQL Post-Update Script Output

4.6.2 MongoDB

Will follow a similar approach as MySQL.

```
FinalDTS123> db.Enrollments.aggregate([
...     {
...         $match: {
...             "semester.semester_name": "2024-April"  // Filter for the specific semester at the start
...         }
...     },
...     {
...         $lookup: {
...             from: "Courses",
...             localField: "course_id",
...             foreignField: "_id",
...             as: "course_info"
...         }
...     },
...     {
...         $unwind: "$course_info"
...     },
...     {
...         $match: {
...             "course_info.code": "GP106"  // Filter for the specific course
...         }
...     },
...     {
...         $lookup: {
...             from: "Students",
...             localField: "student_id",
...             foreignField: "_id",
...             as: "student_info"
...         }
...     },
...     {
...         $unwind: "$student_info"
...     },
...     {
...         $project: {
...             _id: 0,
...             studentId: "$student_info._id",
...             firstName: "$student_info.first_name",
...             lastName: "$student_info.last_name",
...             email: "$student_info.contact_details.email",
...             courseCode: "$course_info.code",
...             courseName: "$course_info.name",
...             semester: "$semester.semester_name",  // Ensure the semester data comes from the enrollment document
...             grade: "$grade"
...         }
...     }
... ]).pretty();
```

Figure 76: Query 6 - MongoDB Select Script

```
[
  {
    studentId: ObjectId("6638275f6e47134e8000877b"),
    firstName: 'Noah',
    lastName: 'Brown',
    email: 'noah.brown@email.com',
    courseCode: 'GP106',
    courseName: 'Introduction to Programming',
    semester: [ '2024-Jan', '2024-April' ],
    grade: 'B'
  },
  {
    studentId: ObjectId("6638275f6e47134e8000877a"),
    firstName: 'Olivia',
    lastName: 'Smith',
    email: 'olivia.smith@email.com',
    courseCode: 'GP106',
    courseName: 'Introduction to Programming',
    semester: [ '2023-Oct', '2024-April' ],
    grade: 'B'
  }
]
```

Figure 77: Query 6 - MongoDB Select Script Output

Results match with MySQL results.

```
FinalDTS123> var corGP106 = db.Courses.findOne({ code: "GP106" });

FinalDTS123>

FinalDTS123> if (corGP106) {
...     // Update the grade to 'A' for all enrollments in 'GP106' during '2024-April'
...     db.Enrollments.updateMany(
...         {
...             "semester.semester_name": "2024-April",
...             "course_id": corGP106._id
...         },
...         {
...             $set: {
...                 "grade": "A"
...             }
...         }
...     );
... } else {
...     print("No course found with the code GP106");
... }
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
FinalDTS123>
```

Figure 78: Query 6 - MongoDB Update scripts

Verify the update by running the previous query.

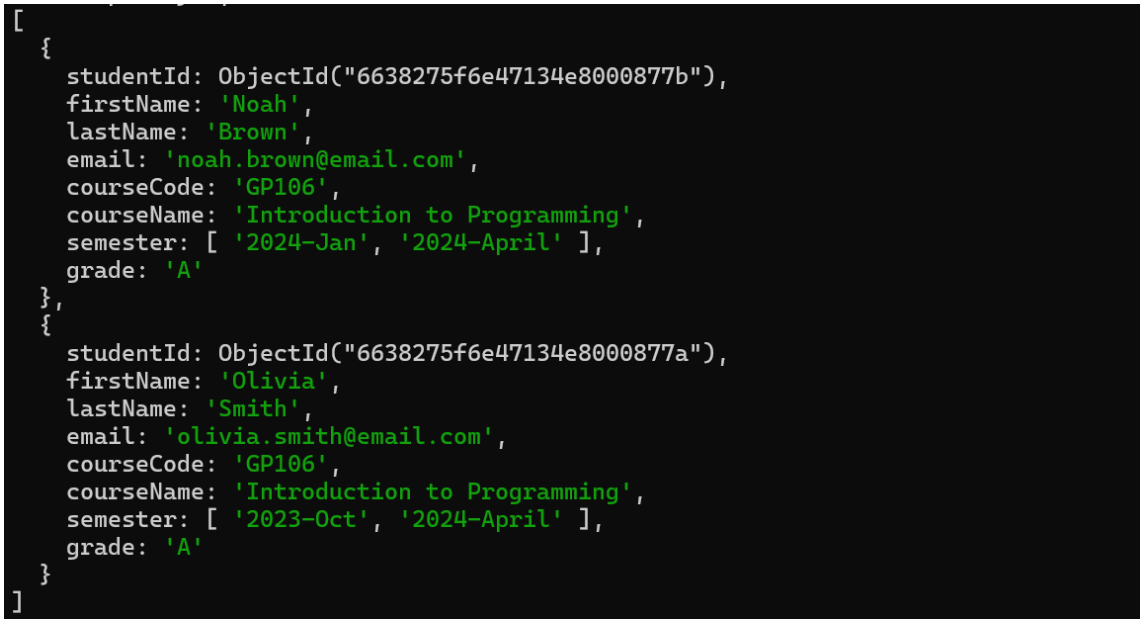


Figure 79: Query 6 - MongoDB Output after the update

As expected, the grade is updated from B to A.

5 Task 4: Comparative Analysis

When designing a university student management system, choosing MySQL and MongoDB significantly influences the system's architecture, performance, and maintainability. Each database system has its distinct advantages and challenges, particularly in terms of schema design, query performance, ease of use, scalability, and suitability for handling different data relationships and volumes.

Schema Design and Flexibility

MySQL, as a relational database management system (RDBMS), requires a well-thought-out schema defined before any data can be stored. This approach demands a comprehensive understanding of the data relationships and future requirements, as changes to the database structure can be complex and disruptive once the system is operational. The schema in MySQL must be designed to cover current and anticipated future scenarios, which might lead to over-engineering or constraints that later require costly restructuring.

In contrast, MongoDB is schema-less, providing substantial data representation flexibility. This means that the design can initially be limited to current understandings without extensive foresight into future needs, aligning well with agile development processes. Changes to the document structure can be made progressively as requirements evolve, significantly reducing the initial development overhead and allowing for quicker adaptations as use cases change.

Query Performance

MySQL optimises query performance through well-defined indexes and can execute complex queries involving multiple tables effectively, given that the schema is normalised and indexes are properly set up. However, complex joins across many tables can degrade performance, particularly as data volume grows. For applications with a heavy read requirement, this can become a bottleneck.

MongoDB, meanwhile, excels in query performance when documents are designed to embed related information that would otherwise require joins in a relational database. By embedding documents, MongoDB reduces the need to perform expensive join operations, enabling faster data retrieval. However, this comes at the cost of data redundancy and potentially larger document sizes.

Ease of Use and Scalability

MySQL requires more specialised knowledge of SQL and database design principles, which can have a steeper learning curve. Managing a relational database involves careful planning of tables, relationships, and indexes to ensure optimal performance and maintainability.

MongoDB uses a JSON-like format for data storage, making it more intuitive for developers familiar with JavaScript and modern web APIs. This can reduce the learning curve and speed up development. Scalability in MongoDB is also more straightforward due to its horizontal scaling capabilities, allowing it to handle large volumes of data more efficiently by distributing the data across multiple servers.

Handling Complex Relationships vs. Large Volumes of Data

MySQL is inherently suited for applications with complex data relationships due to its relational nature. It enforces data integrity and transactional consistency with features like foreign keys,

joins, and atomic transactions, making it ideal for systems where relationships are intricate and data integrity is paramount.

MongoDB is better suited for scenarios with large volumes of data that do not fit neatly into a schema. Its document-oriented approach allows for flexible and dynamic data modelling, which is beneficial in scenarios where data structure can vary or where speed and scalability are more critical than complex data integrity.

Summary

Both MySQL and MongoDB offer robust solutions for managing data, but their suitability varies based on the specific requirements of the university student management system. MySQL provides a structured environment ideal for complex queries and relationship integrity, making it suitable for systems where relationships and data consistency are crucial. MongoDB offers high performance, scalability, and flexibility, advantages for applications needing to manage large volumes of rapidly changing data without fixed schemas.

Ultimately, the choice between MySQL and MongoDB should be guided by the university system's specific data access patterns, scalability requirements, and future growth expectations. In many cases, a polyglot persistence approach—using both MySQL and MongoDB where they fit best—could be the optimal solution, leveraging the strengths of each according to different needs within the same application.