Abstract

In today's digital world, Imelda University faces the challenge of managing large amounts of student data efficiently while keeping it accurate, secure, and easy to access. The manual processes they use now often lead to mistakes, delays, and inefficiencies. This case study examines the transformation of their existing student database system into a streamlined, automated one with the integration of views, triggers, and enhanced indexing techniques.

CHAPTER 1

Introduction

Introduction

This document introduces the project by discussing the study's background, the current system, and the envisioned system. It also outlines the project's aim, objectives, scope, and limitations.

Background of Study

Imelda University is a prestigious institution committed to academic excellence and innovation. The university currently maintains an academic database that stores information about its academic staff members, including personal details, educational qualifications, employment history, research publications, and courses taught. However, the system faces several inefficiencies that hinder effective data management and retrieval.

Statement of Problem

Challenges of the Existing System

The current academic staff database at Imelda University lacks efficient search capabilities and suffers from performance issues when handling large datasets. The existing system relies on basic SQL queries without the use of indexing or full-text search functionality, making data retrieval slow and inefficient. The absence of these optimizations leads to delayed responses and difficulties in extracting relevant information.

Envisioned System

The proposed system aims to optimize the university's academic staff database by implementing indexes, full-text search functionality, stored procedures, audit trails, and triggers. These enhancements will significantly improve search capabilities, increase system performance, and provide a more efficient way of retrieving relevant information.

- Indexes will be created to speed up search operations.
- Full-text search functionality will be integrated for advanced keyword-based queries.

- Triggers will be implemented to maintain data integrity and automate responses.
- An audit trail system will be developed to track changes within the database
- Stored procedures will be implemented to improve performance by reducing the overhead of parsing and compiling SQL statements.

These improvements will enhance accessibility, ensure data security, and streamline database management.

Aim and Objectives

The primary aim of this project is to optimize the university's academic database by integrating indexing, full-text search, views, stored procedures, and triggers.

Specific Objectives:

1. Identify the indexing and full-text search techniques suitable for database optimization.
2. Assess the structure and requirements of the existing academic staff database.
3. Design and implement an optimized database system with enhanced query performance.
4. Integrate an audit trail to ensure security and data integrity.

Significance of the Study

Understanding SQL optimization techniques is crucial for efficient database management. This study provides insights into enhancing database performance through indexing, full-text searches, and automation tools like triggers. By improving query speed and data retrieval accuracy, the project contributes valuable knowledge to database administrators, data analysts, and software developers working in the field.

Scope of the Study

This project focuses on optimizing the university's academic database by implementing the following:

- Full-text search and indexing techniques for faster query responses.

- Stored procedures for better data representation and access control.
- Triggers to automate database processes and enforce business rules.
- Audit trails to monitor database modifications and improve security.

The database is implemented using Microsoft SQL Server Management Studio (SSMS) with Database Engine Services. The study does not cover front-end application development, user interface design, or external integrations beyond the database system.

Existing System

Currently, the university employs a conventional database management system characterized by:

- Disparate data sources and redundant records.
- Manual data entry leading to errors and inconsistencies.
- Limited automation, requiring extensive administrative intervention.
- Lack of real-time updates and notifications.

This fragmented approach results in inefficiencies in data retrieval, analysis, and decision-making processes.

Envisioned System

The proposed student database management system aims to overcome the current system's limitations by leveraging stored procedures and triggers for enhanced data integration, integrity, and automation. The key improvements include:

- Centralized student and staff data within a unified database platform.
- Improved accessibility and search capabilities using full-text search and indexes.
- Automation of administrative tasks through stored procedures and triggers.
- Enhanced data security and compliance with privacy regulations.

With these enhancements, administrators, faculty, and staff will gain access to a more efficient, accurate, and reliable database system that meets the growing needs of Imelda University.

CHAPTER TWO

SYSTEM ARCHITECTURE AND METHODOLOGY

DATABASE DESIGN

The database design represents how data is organized and related within the system. This ensures efficient management, accessibility, and integrity of academic data at Imelda University.

TABLE STRUCTURES

The database **Imelda University** was created, and relevant tables were designed based on the entities and attributes needed. The key tables include:

Table 1: Tbl students

This table helps in managing student data efficiently within the database system and enabling easy access to student information.

Column Name	Datatype	Nullable	Constraint Type
StudentID	Integer	No	PRIMARY KEY
Name	Varchar	No	UNIQUE
Gender	Char	No	CHECK CONSTRAINT, for one of the following values (Gender (M, F))
Date of Birth	Date	No	
Email	Varchar	No	CHECK CONSTRAINT, should be in this format ('%_@__%. __%')
Contact Number	Varchar	No	
Address	Varchar	No	
Admission year	Integer	No	
Program	Varchar	No	

Table 2: Tbl Courses

The purpose of a course table is to keep track of various courses offered by the university. This table helps organize and manage course information.

Column Name	Datatype	Nullable	Constraint Type
CourseID	Integer	No	PRIMARY KEY
Session	Varchar	No	
Title	Varchar	No	
Credit Unit	integer	No	
Credit hours	integer	No	FOREIGN KEY REFERENCES (SessionID)

Table 3: Tbl Enrollment

This table stores information about which students are enrolled in in which courses, creating a relationship between the two entities.

Column Name	Datatype	Nullable	Constraint Type
EnrollmentID	Integer	No	PRIMARY KEY
StudentID	Integer	No	
CourseID	Integer	No	
SessionID	Integer	No	
Enrollment date	Date	No	CHECK CONSTRAINT, CHECK (Enrollment Date <= Get date ()
Date of birth	Date	No	
Completion date	Date	No	
Age (Calculated) datediff (year, date of birth, get date ()			CHECK constraint (datediff (year, date of birth, get date ()) >= 18)

Table 4: Tbl Sessions

In a university database, the table sessions are used to track various aspects of student sessions within the system.

Column Name	Datatype	Nullable	Constraint Type
SessionID	Integer	No	PRIMARY KEY
Semester	Varchar	No	
Start date	Date	No	
End date	Date	No	
Academic year	Varchar	No	

Table 5: Course Assignment

This table plays a crucial role in connecting courses and instructors, this table is designed to assign instructors to specific courses.

Column Name	Datatype	Nullable	Constraint Type
InstructorID	Integer	No	FOREIGN KEY (InstructorID) REFERENCES TblInstructors
CourseID	Integer	No	FOREIGN KEY (CourseID) REFERENCES Tblcourses

Table 6: Tbl Instructors

The table serve the purpose of storing essential information about the instructors at the university.

Column Name	Datatype	Nullable	Constraint Type
InstructorID	Integer	No	PRIMARY KEY
Name	Varchar	No	UNIQUE
Gender	Char	No	CHECK CONSTRAINT, for one of the following values (Gender (M, F))
Date of birth	Date	No	
Hire date	Date	No	CHECK CONSTRAINT (hire date <= get date ()
Marital status	Varchar	Yes	
Address	Varchar	No	
Email	Varchar	Yes	CHECK CONSTRAINT, should be in this format ('%_@__%. __%')
Contact number	Varchar	No	CHECK CONSTRAINT, should be in this format ('+234 __ ____ ____')

Table 7: Tbl Qualification

With a dedicated table for qualifications, the university can effectively track and manage the qualifications, ensuring they meet the necessary criteria to teach specific courses within the university,

Column Name	Datatype	Nullable	Constraint Type
QualificationID	Integer	No	
InstructorID	Integer	No	FOREIGN KEY (InstructorID) References tblInstructors
Degree	Varchar	No	
Institution	Varchar	No	
Year obtained	Date	No	
Field of study	Varchar	No	
Additional certification	Varchar	Yes	
Year of experience	Integer	yes	

Table 8: Tbl fees

Stores information related to fees that students need to pay for various services or activities provided by the university, also identify students who have paid the required fees and are eligible to take the exams.

Column Name	Datatype	Nullable	Constraint Type
FeedID	Integer	No	PRIMARY KEY
StudentID	Integer	No	FOREIGN KEY (StudentID) References tblstudent
SessionID	Integer	No	FOREIGN KEY (SessionID)References tbl Session
CourseID	Integer	No	FOREIGN KEY (CourseID)References tbl Courses
Amount	Decimal	No	
Fee type	Varchar	No	CHECK CONSTRAINT, for one of the following values (fee type (tuition, hostel, exam))
Payment Due	Date	No	
Payment status	Varchar	No	CHECK CONSTRAINT, for one of the following values (payment status (paid, unpaid))

Table 9: Tbl Exams

The purpose of this table is to store information about the exams taken by students. This table is linked with the grade table using a common identifier like the studentID, hence the system can determine the grades for each student based on their exam scores.

Column Name	Datatype	Nullable	Constraint Type
ExamID	Integer	No	PRIMARY KEY
FeeID	Integer	No	FOREIGN KEY (FeeID) References Tbl Fees
StudentID	Integer	No	FOREIGN KEY (StudentID) References Tbl Students
InstructorID	Integer	No	FOREIGNKEY (InstructorID)References TblInstructor
CourseID	Integer	No	FOREIGNKEY(CourseID)References TblCourses
SessionID	Integer	No	FOREIGNKEY(SessionID)References TblSessions
Examdate	Date	No	
ExamTime	Time	No	
Exam Room	Varchar	No	
Score	Decimal	Yes	CHECK CONSTRAINT, (Score BETWEEN 0 AND 100)

Table 10: Tbl Grades

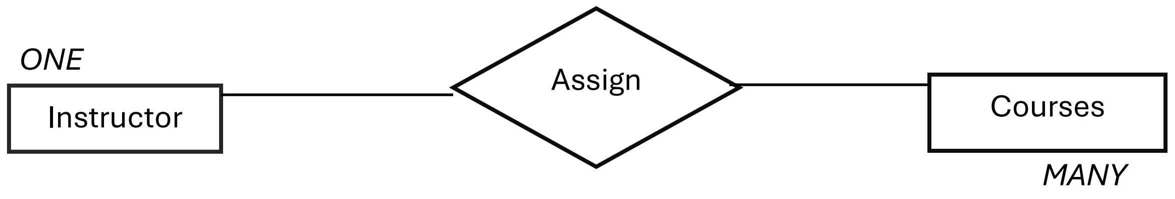
The purpose of the grade table is to determine the grades for students based on their exam scores, when a student takes an exam, their score is compared against predefined criteria in the grade table, depending on the scores achieved, the system automatically assigns the corresponding grade and grade point to the student.

Column Name	Datatype	Nullable	Constraint Type
GradeID	Integer	No	RIMARYKEY
StudentID	Integer	No	FOREIGNKEY (StudentID) References Tblstudents
CourseID	Integer	No	FOREIGNKEY (CourseID)References TblCourses
Score	Integer	Yes	
Grade	Char	Yes	
Grade Point	Decimal	Yes	

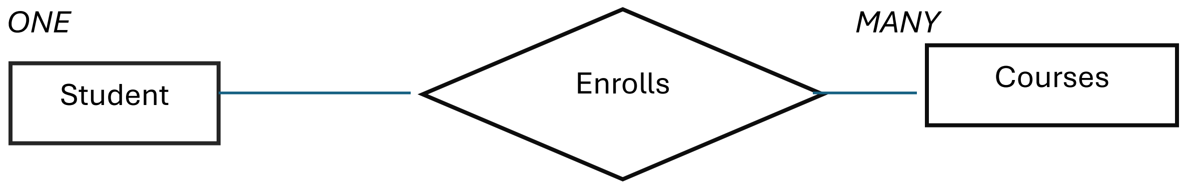
RELATIONSHIPS

At Imelda University, we explore various connections between entities. The diagrams help us pinpoint the types of relationships that exist between them. The Major relationships in our database includes:

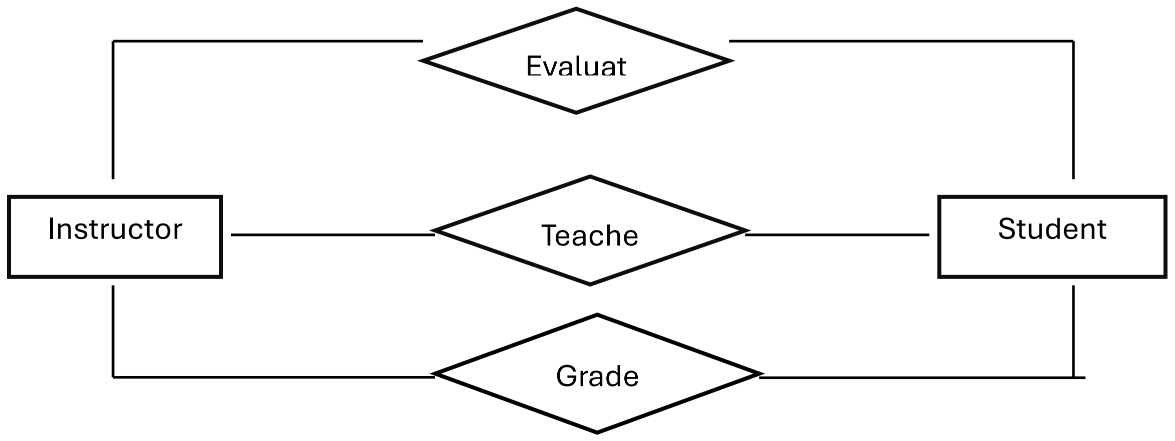
1. The instructor and courses relationship is a one -to- many.



2. The relationship between the student and courses is a one-to-many, the relationship described is based on the relationship in the first semester of the Academic Year. At Imelda University During the first semester, each student is assigned just one course based on their program, ensuring that each course can be taken by multiple students but each student takes only one course



3. Multiple relationships also exist between the same entities, this relationship showcase the different connections. An instructor can evaluate students through exams, teach students and assign grades to their students. Also, the student and instructor relationship is a one -to- many, A student can have multiple instructors as they may be taught different courses from different instructors.



CONSTRAINTS

Constraints ensure data integrity by enforcing rules.

1. **Primary Key Constraint**: Uniquely identifies each record in a table.
2. **Foreign Key Constraint**: Establishes relationships between tables, maintaining referential integrity.
3. **Unique Constraint**: Ensures no duplicate values in specified columns.
4. **Check Constraint**: Enforces specific conditions (e.g., valid email format, age >= 18).

Example SQL Constraint Implementation:

```
CREATE TABLE tblStudents (
  StudentID INTEGER IDENTITY (1,1) PRIMARY KEY,
  Name VARCHAR(100) UNIQUE NOT NULL,
  Gender CHAR(1) CHECK (Gender IN ('M', 'F')) NOT NULL,
  Email VARCHAR(100) CHECK (Email LIKE '%_@__%. __%') NOT NULL,
  ContactNumber VARCHAR(20) CHECK (ContactNumber LIKE '+234 __ ____ ____')
);
```

USER INTERFACE

The database is managed using **SQL Server Management Studio (SSMS 2014)**, which offers:

- **Query Editor**: For writing and executing SQL queries.
- **Table Designer**: For defining table structures and relationships.
- **Query Execution Plans**: For optimizing query performance.
- **Security Features**: For controlling access and maintaining data confidentiality.

CHAPTER 3

DATA OPTIMIZATION OVERVIEW

This overview will detail how each optimization strategy was utilized in our project to streamline processes and boost overall database performance. By leveraging triggers and cursors, indexes, stored procedures, full text search and audit trail mechanism, we aimed to enhance the efficiency and effectiveness of our database operations.

TRIGGERS

By implementing DML trigger, it is fired when data in the underlying table is affected by DML statements, such as *INSERT, UPDATE OR DELETE*. These triggers help in maintaining consistent, reliable and correct data in tables. They enable the process of reflecting changes made in a table to other related tables.

TRIGGER LOGIC:

- I utilized a cursor in the trigger, the logic behind our trigger with a cursor is to iterate through each row in the table, checking if it meets certain conditions and fetching the data accordingly.
- The trigger designed as a *FOR INSERT and FOR UPDATE* Trigger, automatically populates the target table once new values are inserted in the underlying table.
- This automation simplifies the process of updating the intended table based on specific criteria.
- The primary purpose of this trigger is to streamline data population and ensure that the target table stays up to date without manual intervention.
- The detailed query statement is provided in the SQL file.
- I utilized the trigger on Tbl Fees and Tbl Exams and in Audit trail.

STORED PROCEDURE

A Stored procedure in a database is a set of instructions or tasks that can be saved and reused, it helps in organizing and executing complex database operations efficiently.

Stored procedure can enhance performance, security and maintainability of database systems. we developed a set of well-defined stored procedures that can handle a common task of updating student scores.

STORED PROCEDURE LOGIC

- I created a stored procedure that aids the trigger, this stored procedure enables the user to update the score in the exams table and the trigger on the exams table fetches the score as a criterion to populate the grade table:
- I created a stored procedure to also fetch information about students called the power search student, more information about this will be in unit testing.

INDEXES

Indexes are like bookmarks in a book that help you quickly find specific information. They are used to speed up retrieval of data from database tables by creating a sorted reference to the data. Indexes can improve query performance by allowing the database engine to locate and retrieve data more efficiently

INDEXES LOGIC:

Speed up the execution of the following tasks:

- Search by name (Student or Instructor):

  (CREATE INDEX idx_student_name ON tblStudents (Name);

  (CREATE INDEX idx_instructor_name ON tblInstructors (Name);
- Course Title:

  (CREATE INDEX idx_course_title ON tblCourses (Title);
- Address (Student or Instructor

  (CREATE INDEX idx_student_address ON tblStudents ([Address]);

  (CREATE INDEX idx_instructor_address ON tblInstructors ([Address]);

CONSIDERATION

Indexes are beneficial for speeding up *SELECT* queries but can impact the performance of INSERT, UPDATE AND DELETE operations as the indexes need to be updated along with the data. Therefore, it's essential to create indexes wisely based on queries commonly used to create a balance between query performance and data modification efficiency.

FULL TEXT SEARCH

Full text search is like having a powerful search engine within your datatabase.it allows you to search for specific words or phrases within text columns efficiently. The purpose of full text search is to enable users to perform complex searches on large amounts of text data with speed and accuracy.

FULL TEXT SEARCH LOGIC:

Simplify the following tasks using FULL TEXT:

- Extracting Instructor members information based on specific keyword related to their courses.
- Extracting Instructors members information based on specific keywords related to their academic qualifications.

To retrieve the required details by using full text search, we configured full text search on the database first by:

- creating a full text catalog
- creating a unique index
- creating a full text index
- populate the full text index (full text search)

By creating full text indexes on text columns, the database can efficiently execute full text search queries.

- *create full text catalog CAT1 as default*

  create unique index Ix_Itr on tblInstructors (InstructorID)
  create full text index on tblInstructors
  (Name, Gender, Marital Status, [Address], Email, Contact Number)
  KEY INDEX Ix_Itr on CAT1

- *create full text catalog CAT2*

  create unique index Ix_cos on tblCourses (CourseID)
  create full text index on tblCourses
  (Title)
  KEY INDEX Ix_cos on CAT2

- *create full text catalog CAT3*

  create unique index Ix_qlf on tblQualifications (QualificationID)
  create full text index on tblQualifications
  (Degree, Institution, FieldOfStudy, Additional Certification)
  KEY INDEX Ix_qlf on CAT3.

AUDIT TRAIL

An audit trail is like a detailed record of all the changes made to data in a database. It tracks modifications, additions, and deletions of data, along with information like who made the changes and when. Audit trail is crucial for maintaining data integrity, tracking user actions, and ensuring compliance with regulations.

AUDIT TRAIL LOGIC

To implement Audit trail in Imelda University database I:

- Created a separate table in SQL to store audit information.
- *(CREATE TABLE tblStudentsAudit AuditID INTEGER IDENTITY (1,1) CONSTRAINT PK_AuditID PRIMARY KEY (AuditID), Audit Data NVARCHAR (200)*
- Record details like user, timestamp, operation type, and modified data.
- Use triggers to capture these details automatically on table changes.

  (CREATE TRIGGER udtr_StudentsAuditON tblStudents AFTER INSERT, UPDATE, DELETE   AS BEGIN)
- Ensure a comprehensive audit trail for tracking changes and maintaining data integrity.
- The detailed query statement is provided in the SQL file.

UNIT TESTING

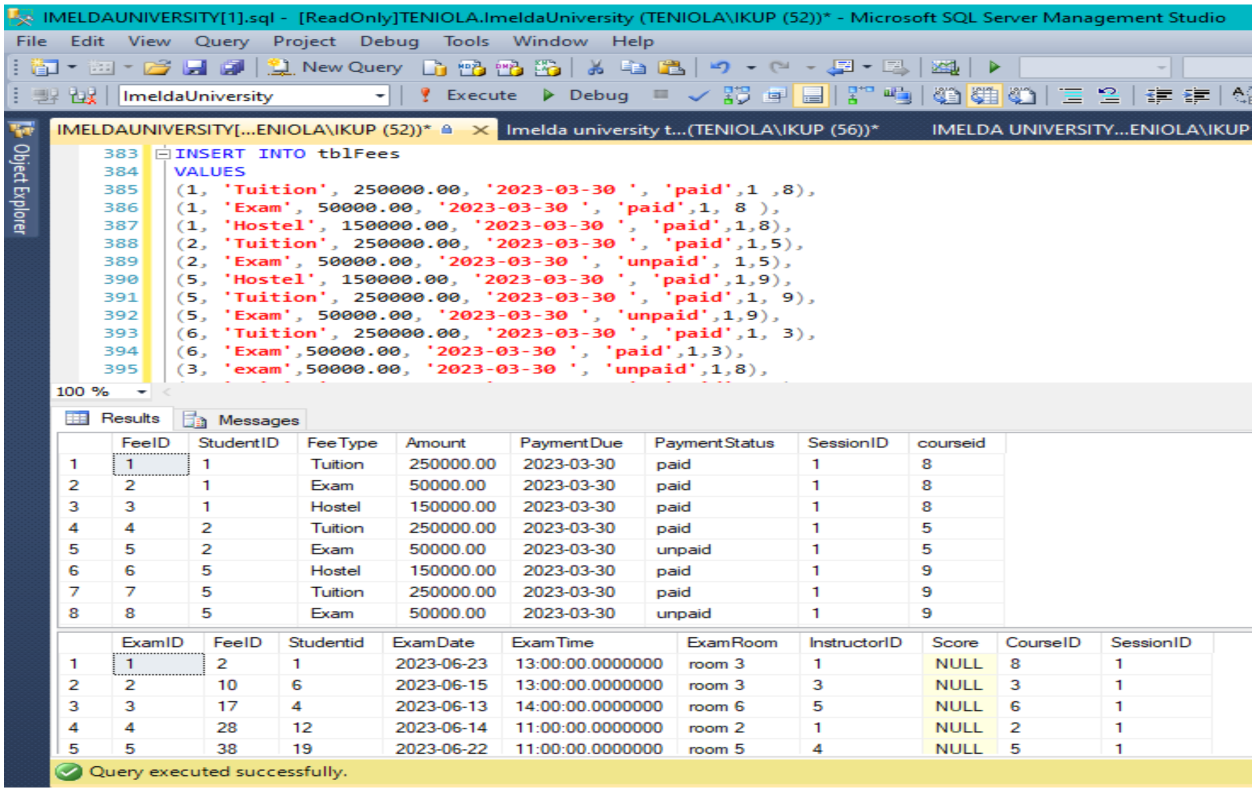
Test Case 1. Trigger on Fees Table

- Input: Insert values into the fees table
- Expected output: Verify that the exam table is populated based on the trigger condition.

Steps:

- Create the *FOR-INSERT* trigger that populates the exam table based on the table fees condition,
- Insert test values into the fees table.
- When the payment type is “exam” and the corresponding payment status is “paid”, fetch the record of that student and populate the exam table columns declared excluding the score column.
- The Score column is excluded because scores are inserted after a student has qualified for the exam and sat for the exam.

The trigger performs the function based on the condition in the fee table by populating these columns in the exam table (fee id, student id, instructor id, course id, session, exam date, exam time and exam room), all these columns are implicitly provided when values are now inserted in the fees table.



IMELDAUNIVERSITY[1].sql - [ReadOnly]TENIOLA.ImeldaUniversity (TENIOLA\IKUP (52))* - Microsoft SQL Server Management Studio

```sql
INSERT INTO tblFees
VALUES
(1, 'Tuition', 250000.00, '2023-03-30 ', 'paid',1 ,8),
(1, 'Exam', 50000.00, '2023-03-30 ', 'paid',1, 8 ),
(1, 'Hostel', 150000.00, '2023-03-30 ', 'paid',1,8),
(2, 'Tuition', 250000.00, '2023-03-30 ', 'paid',1,5),
(2, 'Exam', 50000.00, '2023-03-30 ', 'unpaid', 1,5),
(5, 'Hostel', 150000.00, '2023-03-30 ', 'paid',1,9),
(5, 'Tuition', 250000.00, '2023-03-30 ', 'paid',1, 9),
(5, 'Exam', 50000.00, '2023-03-30 ', 'unpaid',1,9),
(6, 'Tuition', 250000.00, '2023-03-30 ', 'paid',1, 3),
(6, 'Exam',50000.00, '2023-03-30 ', 'paid',1,3),
(3, 'exam',50000.00, '2023-03-30 ', 'unpaid',1,8),
```

```
   FeeID  StudentID  FeeType  Amount     PaymentDue  PaymentStatus  SessionID  courseid
1  1      1          Tuition  250000.00  2023-03-30  paid           1          8
2  2      1          Exam     50000.00   2023-03-30  paid           1          8
3  3      1          Hostel   150000.00  2023-03-30  paid           1          8
4  4      2          Tuition  250000.00  2023-03-30  paid           1          5
5  5      2          Exam     50000.00   2023-03-30  unpaid         1          5
6  6      5          Hostel   150000.00  2023-03-30  paid           1          9
7  7      5          Tuition  250000.00  2023-03-30  paid           1          9
8  8      5          Exam     50000.00   2023-03-30  unpaid         1          9
```

```
   ExamID  FeeID  Studentid  ExamDate    ExamTime          ExamRoom  InstructorID  Score  CourseID  SessionID
1  1       2      1          2023-06-23  13:00:00.0000000  room 3    1             NULL   8         1
2  2       10     6          2023-06-15  13:00:00.0000000  room 3    3             NULL   3         1
3  3       17     4          2023-06-13  14:00:00.0000000  room 6    5             NULL   6         1
4  4       28     12         2023-06-14  11:00:00.0000000  room 2    1             NULL   2         1
5  5       38     19         2023-06-22  11:00:00.0000000  room 5    4             NULL   5         1
```

Query executed successfully.

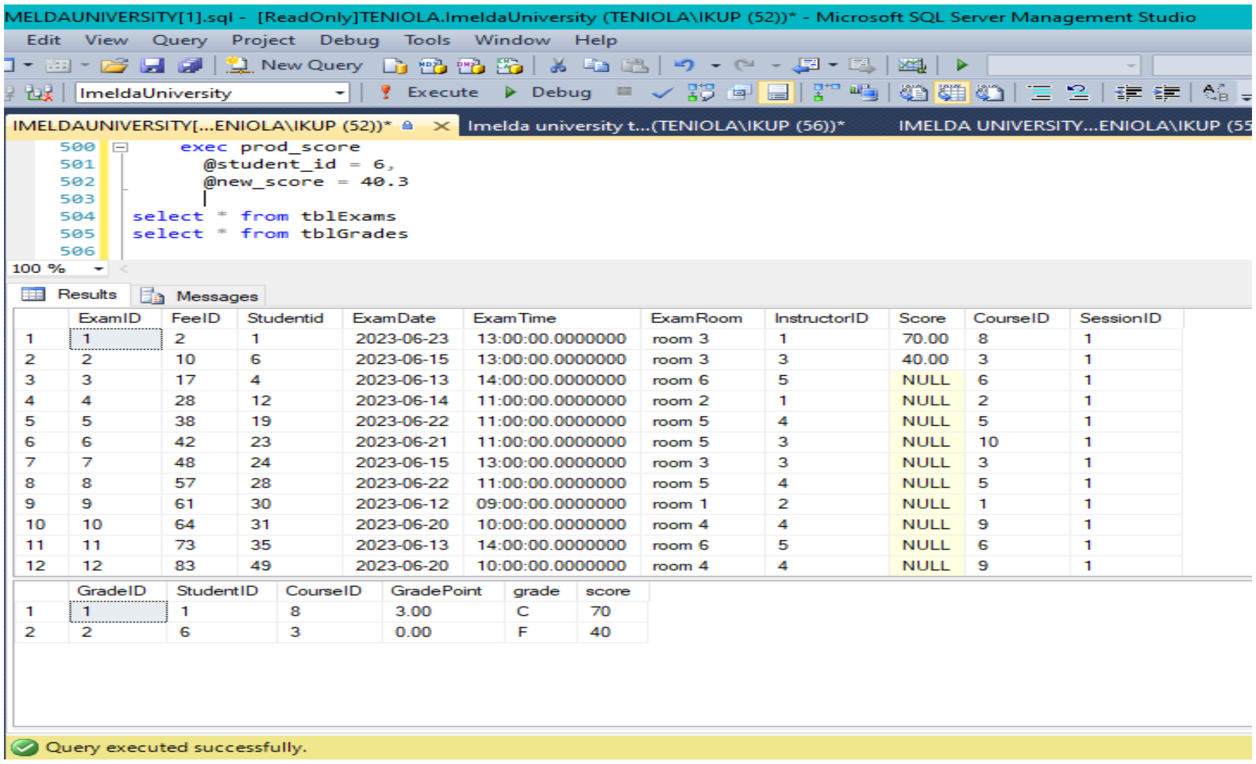
FIGURE 1 :SHOWING THE INSERT VALUES AND FOR INSERT TRIGGER

<u>Test Case 2: Updating Exam Scores</u>

- Input: Insert score into the exam table after the student has taken the exam.
- Expected Output: Verify that the grade table is populated based on the score inserted in the exam table using the trigger.

<u>Steps:</u>

- Create a stored procedure to update the student's score in the exam table.
- Create a *FOR UPDATE* trigger that populates the grade table based on the score conditions.
- The score determines the columns in the grade table (Grade id, student id, course id, score, grade, grade point).
- The stored procedure only needs the user to insert the student id and score and automatically the grade table is populated.
- showing the syntax to call the stored procedure, results showing the exam table with the updated score, grade table is populated as seen for id 1 and 6 where the score has been updated.
- showing inserted values, results showing fee table and exam table, verify that the student id that meets the condition are fetched from fee table and inserted into exam table. i.e. (ID 1 and 6).


MELDAUNIVERSITY[1].sql - [ReadOnly]TENIOLA.ImeldaUniversity (TENIOLA\IKUP (52))* - Microsoft SQL Server Management Studio
Edit View Query Project Debug Tools Window Help
New Query
ImeldaUniversity
Execute
Debug
IMELDAUNIVERSITY[...ENIOLA\IKUP (52))*
Imelda university t...(TENIOLA\IKUP (56))*
IMELDA UNIVERSITY...ENIOLA\IKUP (55
100 %
Results
Messages

```
    exec prod_score
      @student_id = 6,
      @new_score = 40.3

select * from tblExams
select * from tblGrades
```

	ExamID	FeeID	Studentid	ExamDate	ExamTime	ExamRoom	InstructorID	Score	CourseID	SessionID
1	1	2	1	2023-06-23	13:00:00.0000000	room 3	1	70.00	8	1
2	2	10	6	2023-06-15	13:00:00.0000000	room 3	3	40.00	3	1
3	3	17	4	2023-06-13	14:00:00.0000000	room 6	5	NULL	6	1
4	4	28	12	2023-06-14	11:00:00.0000000	room 2	1	NULL	2	1
5	5	38	19	2023-06-22	11:00:00.0000000	room 5	4	NULL	5	1
6	6	42	23	2023-06-21	11:00:00.0000000	room 5	3	NULL	10	1
7	7	48	24	2023-06-15	13:00:00.0000000	room 3	3	NULL	3	1
8	8	57	28	2023-06-22	11:00:00.0000000	room 5	4	NULL	5	1
9	9	61	30	2023-06-12	09:00:00.0000000	room 1	2	NULL	1	1
10	10	64	31	2023-06-20	10:00:00.0000000	room 4	4	NULL	9	1
11	11	73	35	2023-06-13	14:00:00.0000000	room 6	5	NULL	6	1
12	12	83	49	2023-06-20	10:00:00.0000000	room 4	4	NULL	9	1

	GradeID	StudentID	CourseID	GradePoint	grade	score
1	1	1	8	3.00	C	70
2	2	6	3	0.00	F	40

Query executed successfully.

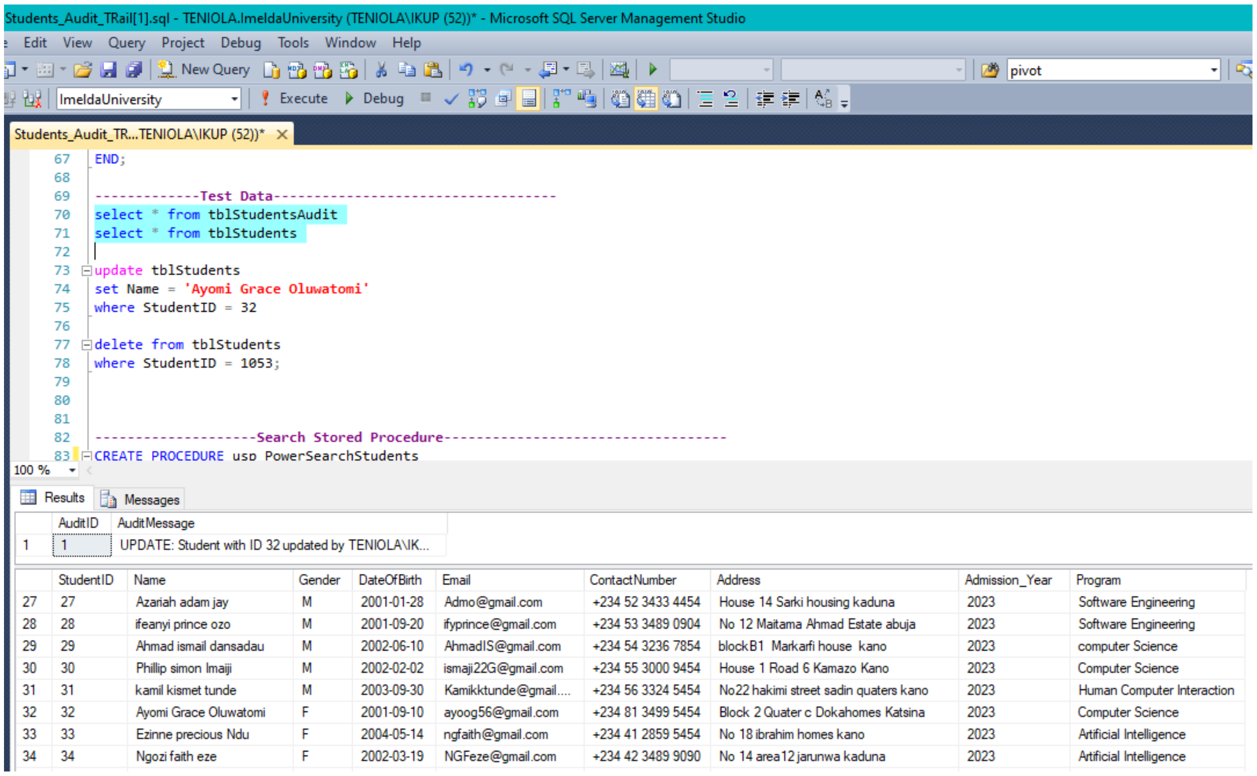
FIGURE 2: SHOWING THE UPDATE STORED PROCEDURE AND UPDATE TRIGGER

Test case 3: Audit Trail Verification

- Input: update values into the student table
- Expected output: With *AUDIT TRAIL,* verify that the student table has been modified and the change has been captured and recorded.

Steps:

- Create an *Audit Trail,* a table to record the changes made to data in the database. we created the audit trail to track changes made to the student table.
- When an update has been made to the student table, verify by checking the student and audit table
- The figure below will display the audit trail table and student table to record the changes and confirm the modification in the student table.


Students_Audit_TRail[1].sql - TENIOLA.ImeldaUniversity (TENIOLA\IKUP (52))* - Microsoft SQL Server Management Studio
Edit View Query Project Debug Tools Window Help
New Query
ImeldaUniversity
Execute
Debug
pivot
Students_Audit_TR...TENIOLA\IKUP (52))*
100 %
Results
Messages

```sql
END;

------------Test Data---------------------------------
select * from tblStudentsAudit
select * from tblStudents

update tblStudents
set Name = 'Ayomi Grace Oluwatomi'
where StudentID = 32

delete from tblStudents
where StudentID = 1053;



--------------------Search Stored Procedure-----------------------------------
CREATE PROCEDURE usp PowerSearchStudents
```

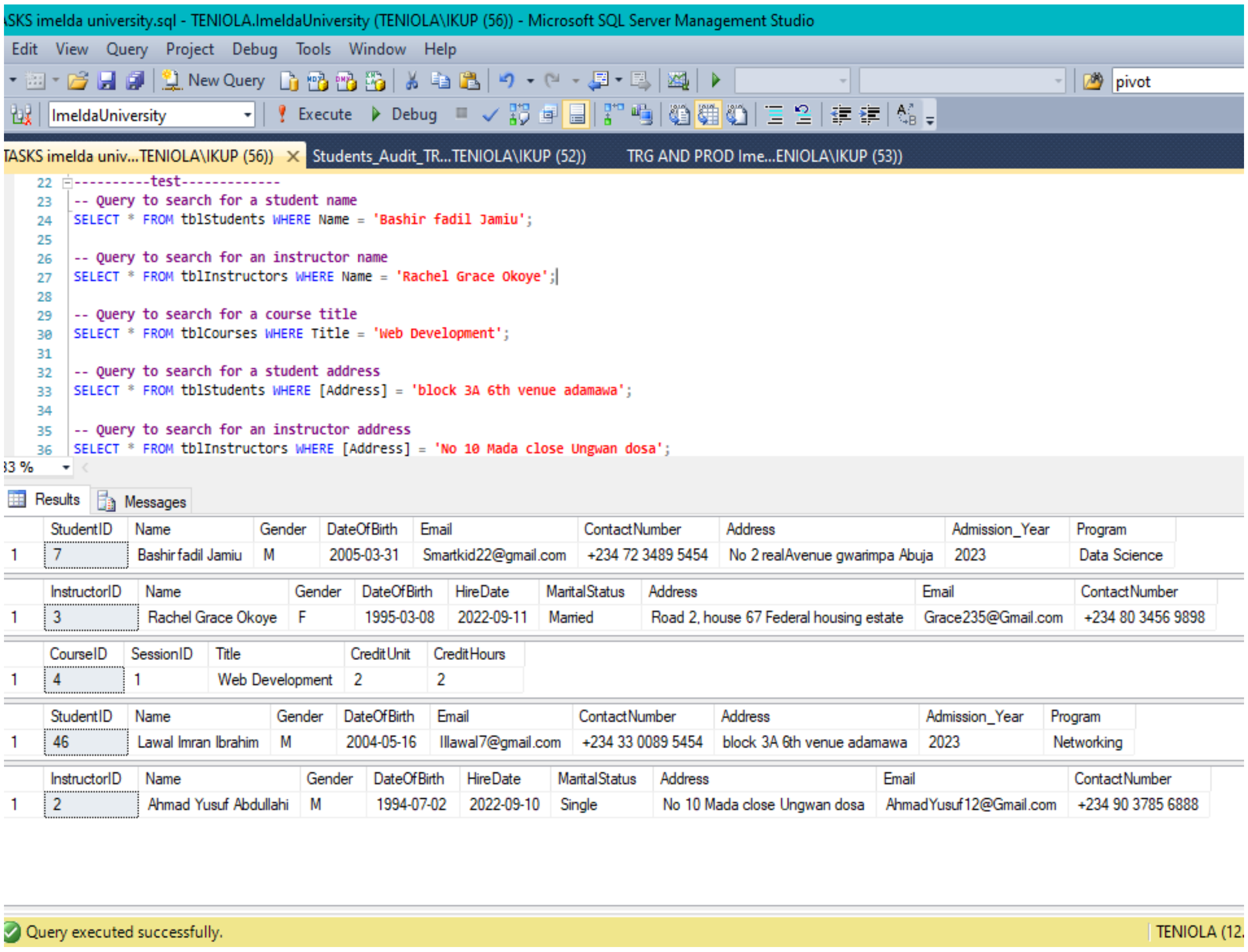
	AuditID	AuditMessage
1	1	UPDATE: Student with ID 32 updated by TENIOLA\IK...

	StudentID	Name	Gender	DateOfBirth	Email	ContactNumber	Address	Admission_Year	Program
27	27	Azariah adam jay	M	2001-01-28	Admo@gmail.com	+234 52 3433 4454	House 14 Sarki housing kaduna	2023	Software Engineering
28	28	ifeanyi prince ozo	M	2001-09-20	ifyprince@gmail.com	+234 53 3489 0904	No 12 Maitama Ahmad Estate abuja	2023	Software Engineering
29	29	Ahmad ismail dansadau	M	2002-06-10	AhmadIS@gmail.com	+234 54 3236 7854	blockB1 Markafi house kano	2023	computer Science
30	30	Phillip simon Imaiji	M	2002-02-02	ismaji22G@gmail.com	+234 55 3000 9454	House 1 Road 6 Kamazo Kano	2023	Computer Science
31	31	kamil kismet tunde	M	2003-09-30	Kamikktunde@gmail....	+234 56 3324 5454	No22 hakimi street sadin quaters kano	2023	Human Computer Interaction
32	32	Ayomi Grace Oluwatomi	F	2001-09-10	ayoog56@gmail.com	+234 81 3499 5454	Block 2 Quater c Dokahomes Katsina	2023	Computer Science
33	33	Ezinne precious Ndu	F	2004-05-14	ngfaith@gmail.com	+234 41 2859 5454	No 18 ibrahim homes kano	2023	Artificial Intelligence
34	34	Ngozi faith eze	F	2002-03-19	NGFeze@gmail.com	+234 42 3489 9090	No 14 area12 jarunwa kaduna	2023	Artificial Intelligence

FIGURE 3: SHOWING AUDIT TRAIL TRACKING

Test Case 4: Index Performance Test

- Test on Indexes created.



ASKS imelda university.sql - TENIOLA.ImeldaUniversity (TENIOLA\IKUP (56)) - Microsoft SQL Server Management Studio
Edit View Query Project Debug Tools Window Help
New Query
ImeldaUniversity
Execute
Debug
TASKS imelda univ...TENIOLA\IKUP (56))
Students_Audit_TR...TENIOLA\IKUP (52))
TRG AND PROD Ime...ENIOLA\IKUP (53))
Results
Messages
TENIOLA (12.

```sql
----------test-------------
-- Query to search for a student name
SELECT * FROM tblStudents WHERE Name = 'Bashir fadil Jamiu';

-- Query to search for an instructor name
SELECT * FROM tblInstructors WHERE Name = 'Rachel Grace Okoye';

-- Query to search for a course title
SELECT * FROM tblCourses WHERE Title = 'Web Development';

-- Query to search for a student address
SELECT * FROM tblStudents WHERE [Address] = 'block 3A 6th venue adamawa';

-- Query to search for an instructor address
SELECT * FROM tblInstructors WHERE [Address] = 'No 10 Mada close Ungwan dosa';
```

	StudentID	Name	Gender	DateOfBirth	Email	ContactNumber	Address	Admission_Year	Program
1	7	Bashir fadil Jamiu	M	2005-03-31	Smartkid22@gmail.com	+234 72 3489 5454	No 2 realAvenue gwarimpa Abuja	2023	Data Science

	InstructorID	Name	Gender	DateOfBirth	HireDate	MaritalStatus	Address	Email	ContactNumber
1	3	Rachel Grace Okoye	F	1995-03-08	2022-09-11	Married	Road 2, house 67 Federal housing estate	Grace235@Gmail.com	+234 80 3456 9898

	CourseID	SessionID	Title	CreditUnit	CreditHours
1	4	1	Web Development	2	2

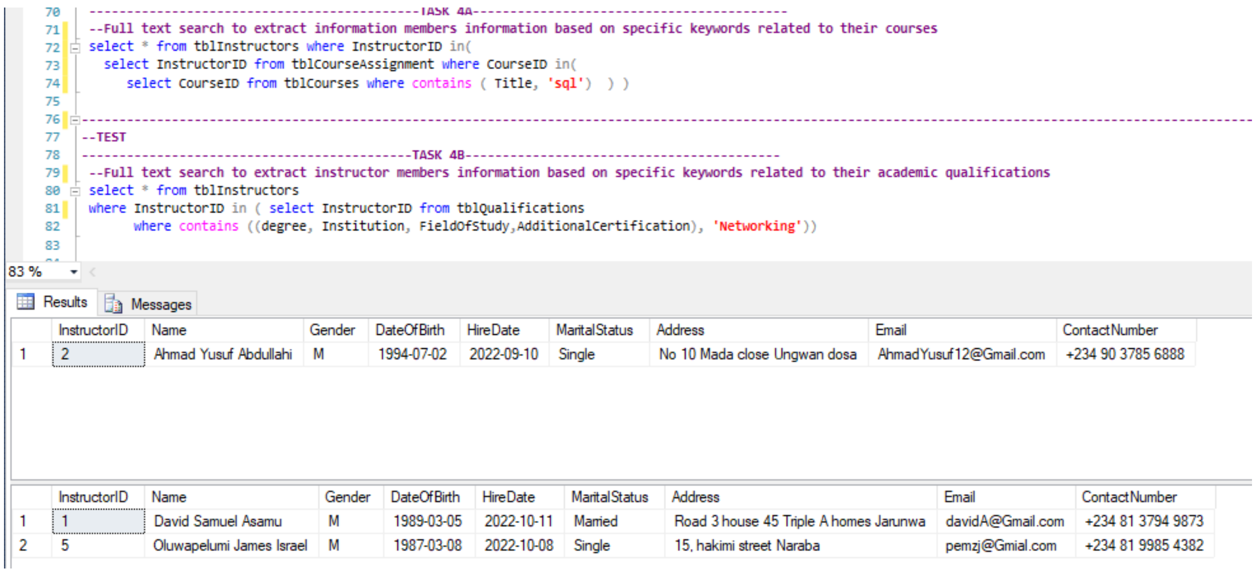
	StudentID	Name	Gender	DateOfBirth	Email	ContactNumber	Address	Admission_Year	Program
1	46	Lawal Imran Ibrahim	M	2004-05-16	Illawal7@gmail.com	+234 33 0089 5454	block 3A 6th venue adamawa	2023	Networking

	InstructorID	Name	Gender	DateOfBirth	HireDate	MaritalStatus	Address	Email	ContactNumber
1	2	Ahmad Yusuf Abdullahi	M	1994-07-02	2022-09-10	Single	No 10 Mada close Ungwan dosa	AhmadYusuf12@Gmail.com	+234 90 3785 6888

Query executed successfully.

FIGURE 4: SHOWING TEST ON INDEXES CREATED

Test Case 5: FULL TEXT SEARCH FUNCTIONALITY



```sql
--------------------------------------------TASK 4A-----------------------------------------
--Full text search to extract information members information based on specific keywords related to their courses
select * from tblInstructors where InstructorID in(
  select InstructorID from tblCourseAssignment where CourseID in(
     select CourseID from tblCourses where contains ( Title, 'sql')  ) )

---------------------------------------------------------------------------------------------------------------------------------------------------------------
--TEST
---------------------------------------------TASK 4B-----------------------------------------
--Full text search to extract instructor members information based on specific keywords related to their academic qualifications
select * from tblInstructors
where InstructorID in ( select InstructorID from tblQualifications
      where contains ((degree, Institution, FieldOfStudy,AdditionalCertification), 'Networking'))
```

83 %

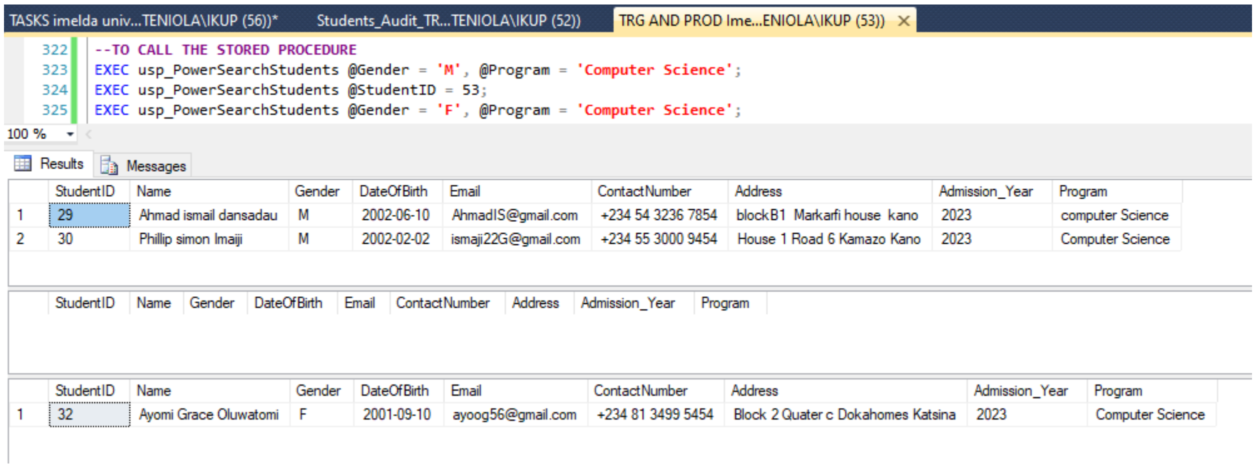
Results | Messages

	InstructorID	Name	Gender	DateOfBirth	HireDate	MaritalStatus	Address	Email	ContactNumber
1	2	Ahmad Yusuf Abdullahi	M	1994-07-02	2022-09-10	Single	No 10 Mada close Ungwan dosa	AhmadYusuf12@Gmail.com	+234 90 3785 6888

	InstructorID	Name	Gender	DateOfBirth	HireDate	MaritalStatus	Address	Email	ContactNumber
1	1	David Samuel Asamu	M	1989-03-05	2022-10-11	Married	Road 3 house 45 Triple A homes Jarunwa	davidA@Gmail.com	+234 81 3794 9873
2	5	Oluwapelumi James Israel	M	1987-03-08	2022-10-08	Single	15, hakimi street Naraba	pemzj@Gmial.com	+234 81 9985 4382

FIGURE 5:TEST ON FULL TEXT SEARCH CRETAED

Test Case 6: STORED PROCEDURE EXECUTION



TASKS imelda univ...TENIOLA\IKUP (56))* | Students_Audit_TR...TENIOLA\IKUP (52)) | TRG AND PROD Ime...ENIOLA\IKUP (53))

```sql
--TO CALL THE STORED PROCEDURE
EXEC usp_PowerSearchStudents @Gender = 'M', @Program = 'Computer Science';
EXEC usp_PowerSearchStudents @StudentID = 53;
EXEC usp_PowerSearchStudents @Gender = 'F', @Program = 'Computer Science';
```

100 %

Results | Messages

	StudentID	Name	Gender	DateOfBirth	Email	ContactNumber	Address	Admission_Year	Program
1	29	Ahmad ismail dansadau	M	2002-06-10	AhmadIS@gmail.com	+234 54 3236 7854	block B1 Markarfi house kano	2023	computer Science
2	30	Phillip simon Imaiji	M	2002-02-02	ismaji22G@gmail.com	+234 55 3000 9454	House 1 Road 6 Kamazo Kano	2023	Computer Science

	StudentID	Name	Gender	DateOfBirth	Email	ContactNumber	Address	Admission_Year	Program

	StudentID	Name	Gender	DateOfBirth	Email	ContactNumber	Address	Admission_Year	Program
1	32	Ayomi Grace Oluwatomi	F	2001-09-10	ayoog56@gmail.com	+234 81 3499 5454	Block 2 Quater c Dokahomes Katsina	2023	Computer Science

FIGURE 6:TEST ON STORED PROCEDURE

CHAPTER FIVE

CHALLENGES AND SOLUTIONS

Challenges Encountered

During the implementation of the database system, we encountered several challenges, including:

- **Change Data Capture (CDC) Issues:** My server did not respond to CDC queries due to system limitations.
- **Identity Column Disruptions:** Frequent deletions caused the identity column to break its sequential numbering.
- **Trigger Functionality:** The trigger failed to fetch data row by row as required.
- **Delayed Table Population:** The results table was not being updated promptly.

Solutions Implemented

To address these issues, we applied the following solutions:

- **CDC Alternative:** I implemented an audit trail to track database changes effectively.
- **Identity Column Fix:** A system procedure was used to reset the identity column, ensuring correct sequential numbering.
- **Trigger Optimization:** A cursor was introduced within the trigger to process each row individually, ensuring accurate data retrieval.
- **Performance Enhancements:** Optimizations were applied to improve the efficiency of data population.

CHAPTER SIX

CONCLUSION AND RECOMMENDATIONS

Conclusion

Developing a robust database system for a university was both challenging and insightful. SQL Server Management Studio (SSMS) enabled us to design a structured solution that streamlined operations and enhanced student information management. By automating key processes, we improved efficiency, minimized redundancy, and strengthened data integrity through optimization techniques.

Recommendations

I recommend using SQL for its powerful data-handling capabilities, which ensure efficient university operations. Mastering SQL can help users automate tasks, maintain data accuracy, and optimize system performance, making it a valuable skill for database management.