**CS F212 | Database Systems**

**Project 7**

**Student Assignment Management System**

****

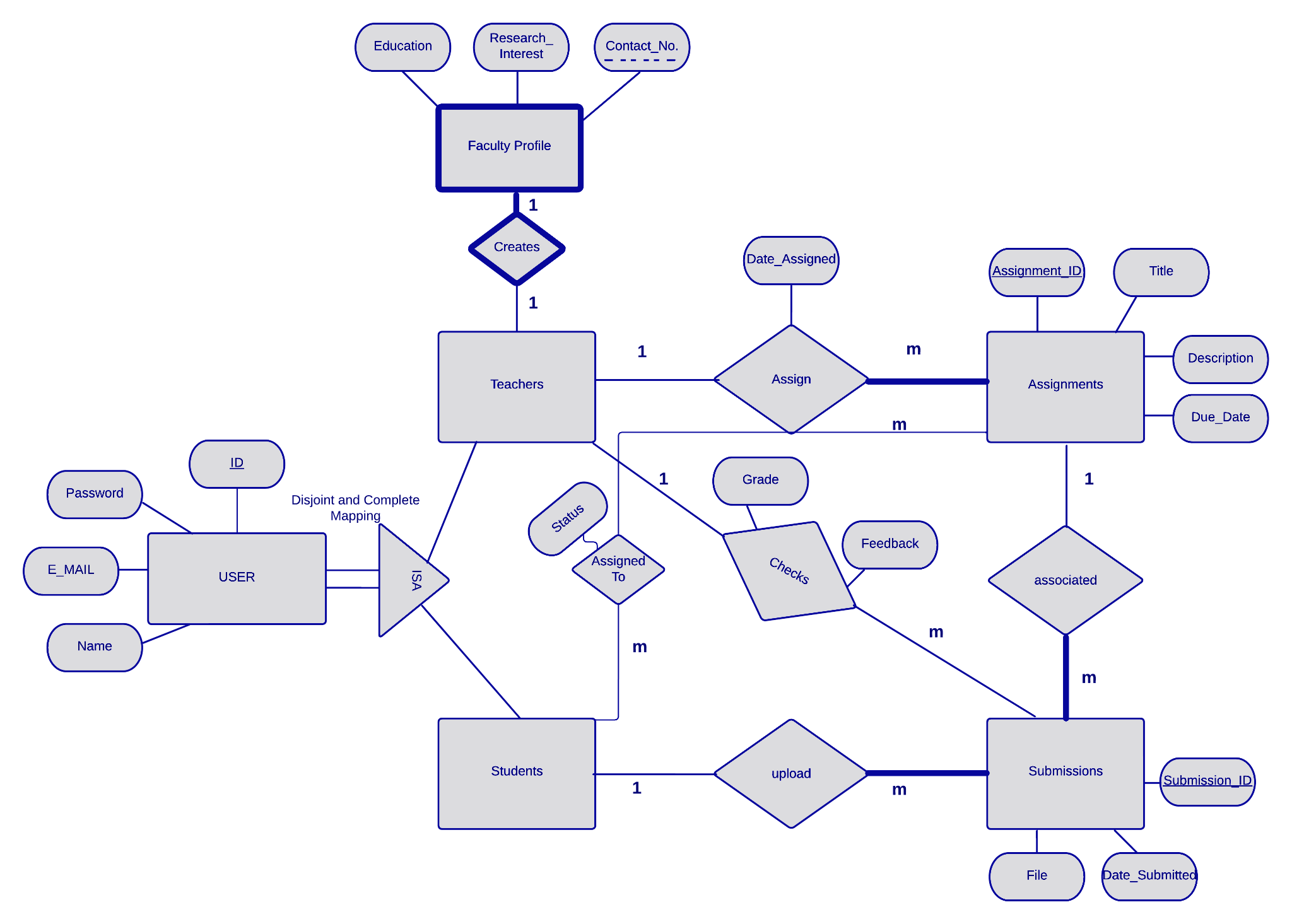
**Divyan Poddar | 2020B5A70604P**

**Bharat Jain | 2020B1A70622P**

**Contents**

1. EER Diagram 3
2. EER-EXPLANATION 3
3. CONVERSION OF ER TO RELATIONAL MODEL 4
4. RELATIONAL SCHEMA AFTER REMOVING REDUNDANCIES 4
5. NORMALIZING EACH RELATION UPTO 3NF 5
   * Teachers Relation
   * Students Relation
   * Assignments Relation
   * Submissions Relation
   * AssignedTo Relation
6. SQL QUERIES TO CREATE RELATIONS 7
7. PROCEDURES (SQL QUERIES) 9
   * REGISTER\_USER
   * VALIDATE\_USER
   * RESET\_PASSWORD
   * ADD\_ASSIGNMENT
   * ADD\_STUDENT\_TO\_ASSIGNMENT
   * ADD\_SUBMISSION
   * GRADE\_SUBMISSION
   * ASSIGNMENT\_COMPLETION\_RATE
   * UPDATE\_TEACHER\_PROFILE
   * ASSIGNMENT\_STATISTIC

**EER-DIAGRAM**

****

**EER-EXPLANATION**

* Each teacher is an user
* Each student is an user
* Each Teacher can create exactly one faculty profile.
* Each faculty profile is created by exactly one teacher.
* Each teacher can assign many assignments.
* Each assignment has exactly one teacher assigning it.
* Each assignment has many submissions associated with it.
* Each submission is associated with exactly one assignment.
* Each student uploads multiple submissions.
* Each submission is uploaded by exactly one student.
* Each student can have many assignments assigned to him/her.
* Each assignment can have many students assigned to it.

**CONVERSION OF ER TO RELATIONAL MODEL**

Entity Relationship model can be reduced to a relational schema by inferring the redundancies and merging tables that can be represented together.

* Specializations **Teachers** and **Students** together form a disjoint and complete mapping of User, hence no need to make a User table explicitly.

The two specializations of User **Teachers** and **Students** will have their own tables having all the fields of User as well as their own.

* **Assign** relation is one-to-many hence can be merged with the many side Assignments.

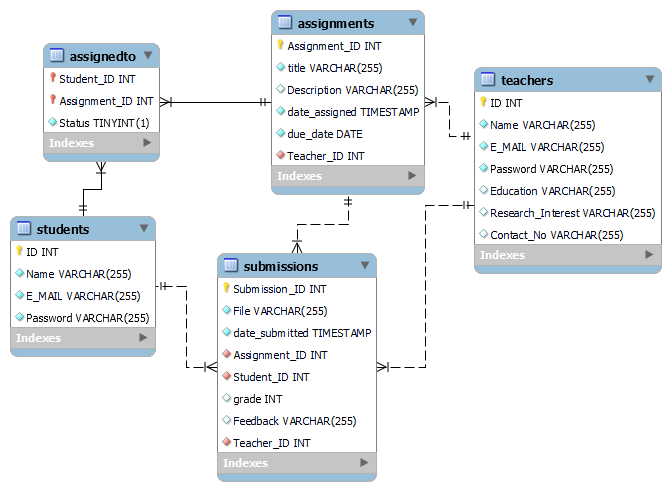
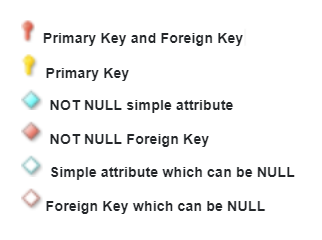
**Assignments** relation so formed has foreign key

* + Teacher\_ID which references the ID field in the Teachers relation.
* **Associated** is a one-to-many relationship hence it can be merged with the many side **Submissions**.
* **Upload** is a one-to-many relationship hence it can be merged with the many side **Submissions**.
* **Grade** is a one-to-many relationship hence it can be merged with the many side **Submissions**.

**Submissions** so formed has foreign keys

* + Assignment\_ID references Assignment\_ID from Assignments
  + Teacher\_ID references ID from Teachers
  + Student\_ID reference ID from Students
* **AssignedTo** is a many-to-many relationship hence we have to make a separate table for it. AssignedTo has 2 foreign keys:
  + Student\_ID references ID from Students
  + Assignment\_ID reference Assignment\_ID from Assignments.
* **Creates** is a one-to-one relationship hence Creates,Teachers,and Faculty-Profile relations are merged into a single relation Teachers.

**RELATIONAL SCHEMA AFTER REMOVING REDUNDANCIES**

****

**NORMALIZING EACH RELATION UPTO 3NF**

Condition for 3NF:

* The relation is in **Second Normal Form (2NF)**.
* There are no **transitive dependencies** between non-prime attributes and the candidate key.

Direct Check for 3NF: LHS of functional dependency Candidate Key/Super Key **OR** RHS Prime Attribute

**Teachers Relation**

1. Functional dependencies:
   1. ID -> Name, E\_MAIL, Password, Education, Research\_Interest, Contact\_No
   2. E\_MAIL -> ID, Name, Password, Education, Research\_Interest, Contact\_No
2. Closure of each attribute:
   1. ID+: ID, Name, E\_MAIL, Password, Education
   2. Name+: Name
   3. E\_MAIL+: ID, Name, E\_MAIL, Password
   4. Password+: Password
   5. Education+: Education
   6. Research\_Interest+: Research\_Interes
   7. Contact\_No.+: Contact\_No.
3. Candidate Keys: ID, E\_MAIL
4. Prime attributes: ID, E\_MAIL
5. Non-prime attributes: Name, Password, Education, Research\_Interest, Contact\_No.

Since all the functional dependencies have either the LHS as a candidate key/superkey or the RHS as a prime attribute the **Teachers** relation is already in 3NF.

**Students Relation**

1. Functional dependencies:
   1. ID -> Name, E\_MAIL, Password
   2. E\_MAIL -> ID, Name, Password
2. Closure of each attribute:
   1. ID+: ID, Name, E\_MAIL, Password
   2. Name+: Name
   3. E\_MAIL+: ID, Name, E\_MAIL, Password
   4. Password+: Password
3. Candidate Keys: ID, E\_MAIL
4. Prime attributes: ID, E\_MAIL
5. Non-prime attributes: Name, Password

Since all the functional dependencies have either the LHS as a candidate key/superkey or the RHS as a prime attribute the **Students** relation is already in 3NF.

**Assignments Relation**

1. Functional dependencies:
   1. Assignment\_ID -> Title, Description, Date\_Assigned, Due\_Date, Teacher\_ID
2. Closure of each attribute:
   1. Assignment\_ID+: Assignment\_ID, Title, Description, Date\_Assigned, Due\_Date, Teacher\_ID
   2. Title+: Title
   3. Description+: Description
   4. Date\_Assigned+: Date\_Assigned
   5. Due\_Date+: Due\_Date
   6. Teacher\_ID+: Teacher\_ID
3. Candidate Keys: Assignment\_ID
4. Prime attributes: Assignment\_ID
5. Non-prime attributes: Title, Description, Date\_Assigned,Due\_Date, Teacher\_ID

Since all the functional dependencies have either the LHS as a candidate key/superkey or the RHS as a prime attribute the **Assignments** relation is already in 3NF.

**Submissions Relation**

1. Functional dependencies:
   1. Submission\_ID -> File, Date\_Submitted, Assignment\_ID, Student\_ID, Grade, Feedback, Teacher\_ID
2. Closure of each attribute:
   1. Submission\_ID+: Submission\_ID, File, Date-Submitted, Assignment\_ID, Student\_ID, Grade, Feedback, Teacher\_ID
   2. File+: File
   3. Date\_Submitted+: Date\_Submitted
   4. Assignment\_ID+: Assignment\_ID
   5. Student\_ID+: Student\_ID
   6. Grade+: Grade
   7. Feedback+: Feedback
   8. Teacher\_ID+: Teacher\_ID
3. Candidate Keys: Submission\_ID
4. Prime attributes: Submission\_ID
5. Non-prime attributes: Title, Description, Date\_Assigned,Due\_Date, Teacher\_ID

Since all the functional dependencies have either the LHS as a candidate key/superkey or the RHS as a prime attribute the **Submission** relation is already in 3NF.

**AssignedTo Relation**

1. Functional dependencies:
   1. Student\_ID, Assignment\_ID -> Status
2. Closure of each attribute:
   1. Student\_ID+: Student\_ID
   2. Assignment\_ID+: Assignment\_ID
   3. Status+: Status
3. Candidate Keys: (Student\_ID, Assignment\_ID)
4. Prime attributes: Student\_ID, Assignment\_ID
5. Non-prime attributes: Status

Since all the functional dependencies have either the LHS as a candidate key/superkey or the RHS as a prime attribute the **AssignedTo** relation is already in 3NF.

**Since all the tables are already in 3NF form, there is no need for further normalization.**

**SQL QUERIES TO CREATE RELATIONS**

**CREATE TABLE `Teachers` (**

**`ID` int NOT NULL,**

**`Name` varchar(255) NOT NULL,**

**`E\_MAIL` varchar(255) NOT NULL,**

**`Password` varchar(255) NOT NULL,**

**`Education` varchar(255) DEFAULT NULL,**

**`Research\_Interest` varchar(255) DEFAULT NULL,**

**`Contact\_No` varchar(255) DEFAULT NULL,**

**PRIMARY KEY (`ID`),**

**UNIQUE KEY `unique\_email` (`E\_MAIL`)**

**)**

**CREATE TABLE `Students` (**

**`ID` int NOT NULL,**

**`Name` varchar(255) NOT NULL,**

**`E\_MAIL` varchar(255) NOT NULL,**

**`Password` varchar(255) NOT NULL,**

**PRIMARY KEY (`ID`),**

**UNIQUE KEY `E\_MAIL` (`E\_MAIL`)**

**)**

**CREATE TABLE `Assignments` (**

**`Assignment\_ID` int NOT NULL AUTO\_INCREMENT,**

**`title` varchar(255) NOT NULL,**

**`Description` varchar(255) DEFAULT NULL,**

**`date\_assigned` timestamp NOT NULL,**

**`due\_date` date NOT NULL,**

**`Teacher\_ID` int NOT NULL,**

**PRIMARY KEY (`Assignment\_ID`),**

**KEY `Teacher\_ID` (`Teacher\_ID`),**

**CONSTRAINT `assignments\_ibfk\_1` FOREIGN KEY (`Teacher\_ID`) REFERENCES `teachers` (`ID`)**

**)**

**CREATE TABLE `Submissions` (**

**`Submission\_ID` int NOT NULL AUTO\_INCREMENT,**

**`File` varchar(255) NOT NULL,**

**`date\_submitted` timestamp NOT NULL,**

**`Assignment\_ID` int NOT NULL,**

**`Student\_ID` int NOT NULL,**

**`grade` int DEFAULT NULL,**

**`Feedback` varchar(255) DEFAULT NULL,**

**`Teacher\_ID` int NOT NULL,**

**PRIMARY KEY (`Submission\_ID`),**

**KEY `Assignment\_ID` (`Assignment\_ID`),**

**KEY `Student\_ID` (`Student\_ID`),**

**KEY `Teacher\_ID` (`Teacher\_ID`),**

**CONSTRAINT `submissions\_ibfk\_1` FOREIGN KEY (`Assignment\_ID`) REFERENCES `assignments` (`Assignment\_ID`),**

**CONSTRAINT `submissions\_ibfk\_2` FOREIGN KEY (`Student\_ID`) REFERENCES `students` (`ID`),**

**CONSTRAINT `submissions\_ibfk\_3` FOREIGN KEY (`Teacher\_ID`) REFERENCES `teachers` (`ID`)**

**)**

**CREATE TABLE `AssigneTo` (**

**`Student\_ID` int NOT NULL,**

**`Assignment\_ID` int NOT NULL,**

**`Status` tinyint(1) NOT NULL,**

**PRIMARY KEY (`Student\_ID`,`Assignment\_ID`),**

**KEY `Assignment\_ID` (`Assignment\_ID`),**

**CONSTRAINT `assignedto\_ibfk\_1` FOREIGN KEY (`Student\_ID`) REFERENCES `students` (`ID`),**

**CONSTRAINT `assignedto\_ibfk\_2` FOREIGN KEY (`Assignment\_ID`) REFERENCES `assignments` (`Assignment\_ID`)**

**)**

**PROCEDURES (SQL QUERIES)**

**1. REGISTER\_USER**

**-- This procedure registers a new user as either a Teacher or a Student**

**-- based on the provided role. It takes in the email, password and role as**

**-- parameters and inserts a new record into either the Teacher or Student table**

**-- accordingly. The ID is set to one more than the maximum ID in the respective table.**

**DELIMITER //**

**CREATE PROCEDURE register\_user (IN name VARCHAR(255), IN email VARCHAR(255), IN password VARCHAR(255), IN role VARCHAR(255))**

**BEGIN**

**DECLARE new\_id INT;**

**-- Start a new transaction**

**START TRANSACTION;**

**IF role = 'Teacher' THEN**

**SET new\_id = (SELECT MAX(ID) + 1 FROM Teachers);**

**INSERT INTO Teachers (ID, Name, E\_MAIL, Password)**

**VALUES (new\_id, name, email, password);**

**SELECT CONCAT('Teacher with name ', name, ' and email ', email, ' has been registered.') AS message;**

**ELSEIF role = 'Student' THEN**

**SET new\_id = (SELECT MAX(ID) + 1 FROM Students);**

**INSERT INTO Students (ID, Name, E\_MAIL, Password)**

**VALUES (new\_id, name, email, password);**

**SELECT CONCAT('Student with name ', name, ' and email ', email, ' has been registered.') AS message;**

**END IF;**

**-- Commit the transaction**

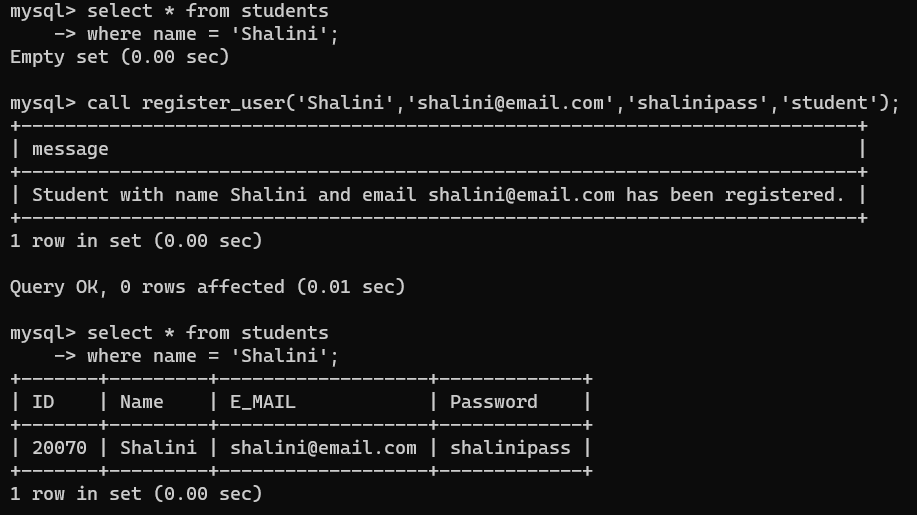
**COMMIT;**

**END //**

**DELIMITER ;**

**-- Example query to call the procedure**

**-- CALL register\_user('Email\_ID’, 'password', 'Role’');**

****

**2. VALIDATE\_USER**

**-- This procedure validates a user's email and password by checking if they exist in either the Students or Teachers table**

**-- and returns a message indicating whether the login was successful or not.**

**DELIMITER $$**

**CREATE PROCEDURE `validate\_user`(**

**IN email VARCHAR(255),**

**IN password1 VARCHAR(255)**

**)**

**BEGIN**

**DECLARE user\_type VARCHAR(255);**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Check if the email exists in either the Students or Teachers table**

**SELECT UserType INTO user\_type FROM (**

**SELECT 'Student' AS UserType, E\_MAIL, Password**

**FROM Students**

**UNION ALL**

**SELECT 'Teacher' AS UserType, E\_MAIL, Password**

**FROM Teachers**

**) AS UserTable**

**WHERE E\_MAIL = email;**

**-- If the email does not exist in either table, return an invalid user message**

**IF user\_type IS NULL THEN**

**SELECT 'Invalid user';**

**-- If the email exists in either table, check if the password is correct**

**ELSEIF EXISTS (**

**SELECT \***

**FROM (**

**SELECT ID, 'Student' AS UserType, E\_MAIL, Password**

**FROM Students**

**WHERE E\_MAIL = email AND Password = password1**

**UNION ALL**

**SELECT ID, 'Teacher' AS UserType, E\_MAIL, Password**

**FROM Teachers**

**WHERE E\_MAIL = email AND Password = password1**

**) AS ValidUserTable**

**) THEN**

**-- If the password is correct, return a successful login message with the user type**

**SELECT CONCAT(user\_type, ' login successful') as ‘SUCCESS’;**

**ELSE**

**-- If the password is incorrect, return an incorrect password message with the user type**

**SELECT CONCAT('Incorrect password for ', user\_type) as ‘ERROR’;**

**END IF;**

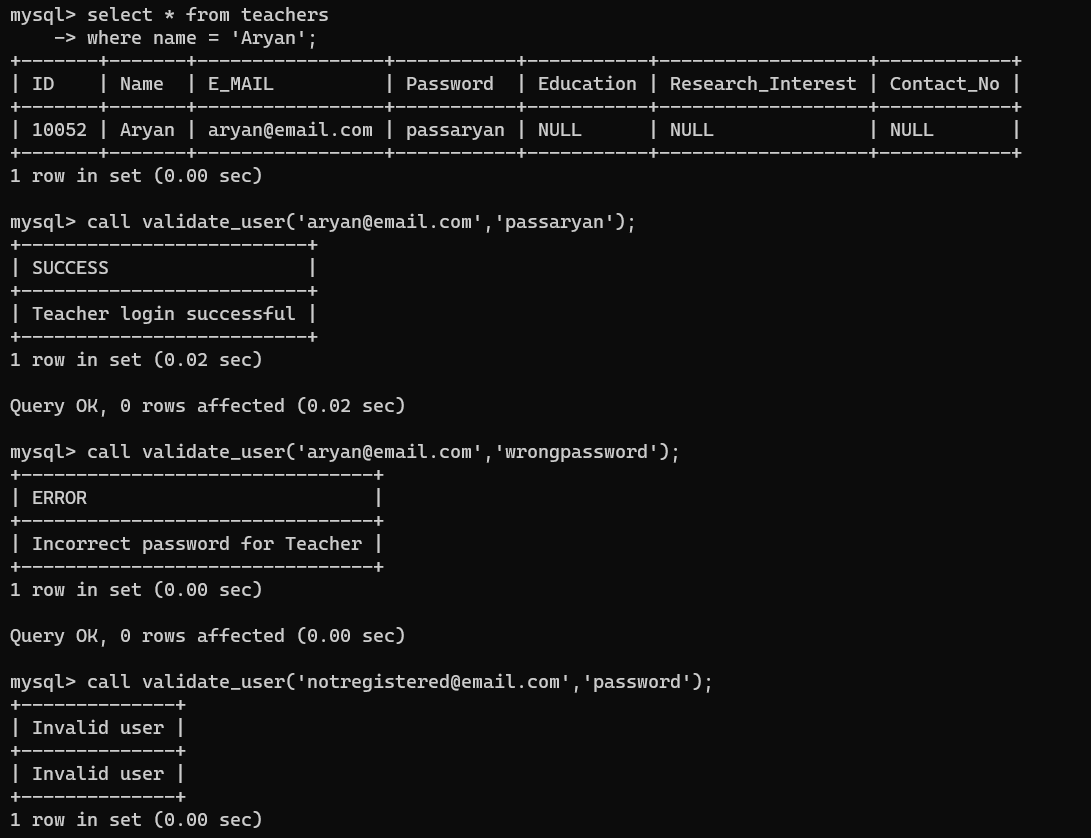
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL validate\_user('email@example.com', 'password');**

****

**3. RESET\_PASSWORD**

**-- This procedure resets a user's password by updating the password in either the Students or Teachers table.**

**DELIMITER $$**

**CREATE PROCEDURE reset\_password(IN email VARCHAR(255), IN password VARCHAR(255))**

**BEGIN**

**DECLARE user\_type VARCHAR(255);**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Update the password in the Students table for the given email**

**UPDATE Students**

**SET Password = password**

**WHERE E\_MAIL = email;**

**-- Update the password in the Teachers table for the given email**

**UPDATE Teachers**

**SET Password = password**

**WHERE E\_MAIL = email;**

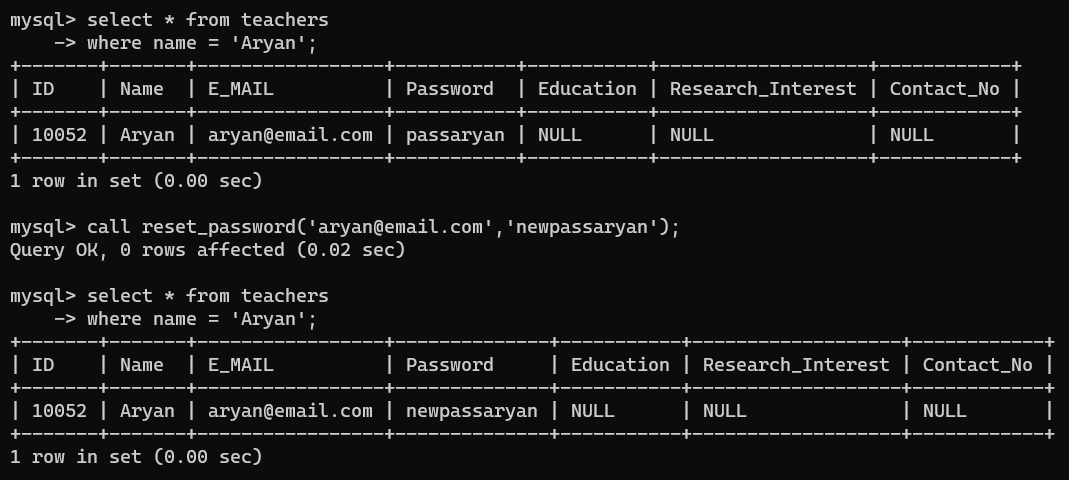
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL reset\_password('email@example.com', 'newpassword');**

****

**4. ADD\_ASSIGNMENT**

**-- This procedure adds an assignment to the Assignments table if the current user is a teacher.**

**DELIMITER $$**

**CREATE PROCEDURE add\_assignment (**

**IN title VARCHAR(255),**

**IN description TEXT,**

**IN due\_date DATE,**

**IN teacher\_email VARCHAR(255)**

**)**

**BEGIN**

**DECLARE user\_type VARCHAR(255);**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Check if the current user is a teacher**

**IF EXISTS (**

**SELECT \***

**FROM Teachers**

**WHERE E\_MAIL = teacher\_email**

**) THEN**

**-- If the current user is a teacher, add the assignment**

**INSERT INTO Assignments (Title, Description, Due\_Date, Teacher\_ID, Date\_Assigned)**

**VALUES (title, description, due\_date,**

**(SELECT ID FROM Teachers WHERE E\_MAIL = teacher\_email),**

**CURRENT\_DATE);**

**SELECT 'Assignment added successfully' AS Result;**

**ELSE**

**-- If the current user is not a teacher, return an error message**

**SELECT 'Error: User is not authorized to add assignments' AS Result;**

**END IF;**

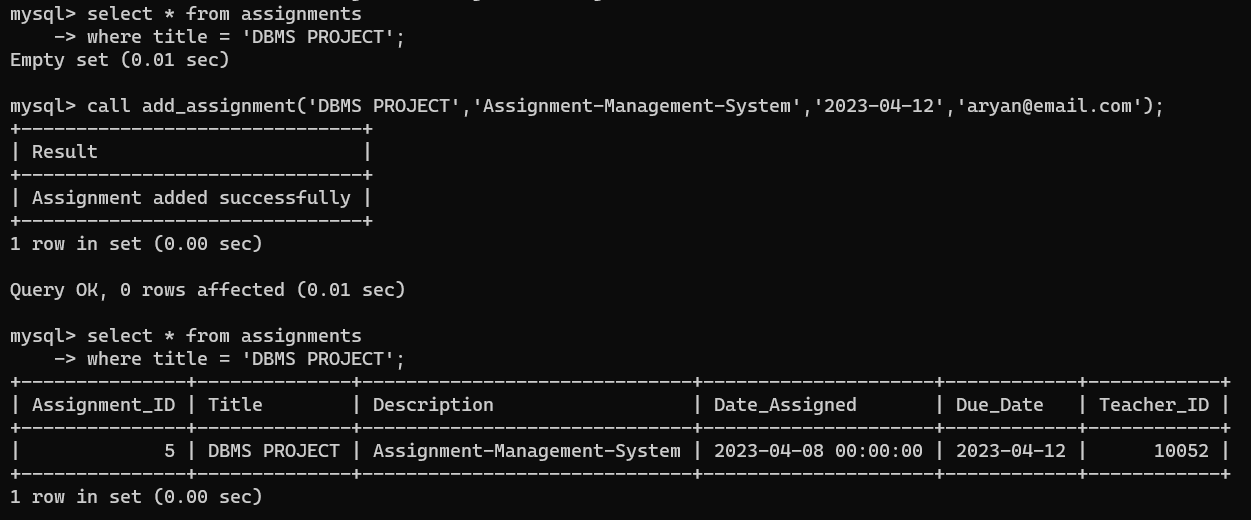
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL add\_assignment('Assignment Title', 'Assignment Description', 'YYYY-MM-DD', '**[**teacher@example.com**](mailto:teacher@example.com)**');**

****

**5. ADD\_STUDENT\_TO\_ASSIGNMENT**

**-- This code adds a student to an assignment in the AssignedTo table if the current user is a teacher who made the assignment.**

**DELIMITER $$**

**CREATE PROCEDURE add\_student\_to\_assignment (**

**IN assignment\_id INT,**

**IN student\_email VARCHAR(255),**

**IN teacher\_email VARCHAR(255)**

**)**

**BEGIN**

**DECLARE user\_type VARCHAR(255);**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Check if the teacher email exists and is the one who made the assignment**

**IF EXISTS (**

**SELECT \***

**FROM Teachers t**

**JOIN Assignments a ON t.ID = a.Teacher\_ID**

**WHERE t.E\_MAIL = teacher\_email AND a.Assignment\_ID = assignment\_id**

**) THEN**

**-- Check if the student email exists**

**IF EXISTS (**

**SELECT \***

**FROM Students**

**WHERE E\_mail = student\_email**

**) THEN**

**-- Add the student to the assignment with a status of false**

**INSERT INTO AssignedTo (Student\_ID, Assignment\_ID, Status)**

**VALUES (**

**(SELECT ID FROM Students WHERE E\_mail = student\_email),**

**assignment\_id,**

**false**

**);**

**SELECT 'Student added to assignment successfully' AS Result;**

**ELSE**

**-- If the student email does not exist, return an error message**

**SELECT 'Error: Student email does not exist' AS Result;**

**END IF;**

**ELSE**

**-- If the teacher email does not exist or is not the one who made the assignment, return an error message**

**SELECT 'Error: Teacher email does not exist or is not the one who made the assignment' AS Result;**

**END IF;**

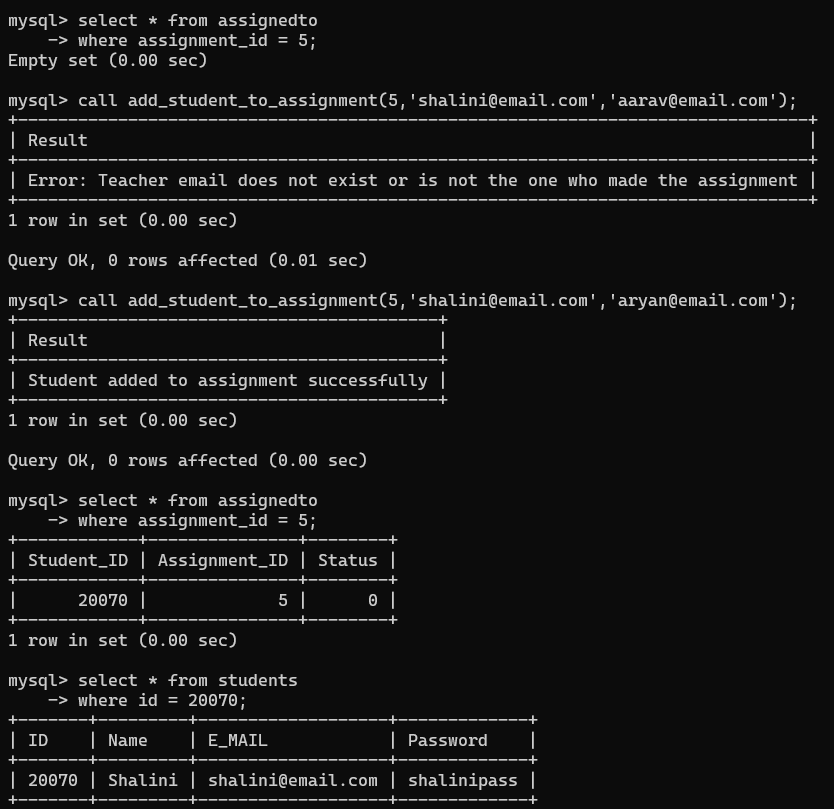
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL add\_student\_to\_assignment(assignment\_id, 'student@example.com', '**[**teacher@example.com**](mailto:teacher@example.com)**');**

****

**6. ADD\_SUBMISSION**

**-- This code adds a submission to the submissions table if the student is assigned the assignment and has not already submitted it.**

**DELIMITER $$**

**CREATE PROCEDURE add\_submission(**

**IN p\_student\_id INT,**

**IN p\_assignment\_id INT,**

**IN p\_file VARCHAR(255)**

**)**

**BEGIN**

**DECLARE v\_due\_date DATE;**

**DECLARE v\_status BOOLEAN;**

**DECLARE v\_submission\_count INT;**

**DECLARE v\_teacher\_id INT;**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Check if the student is assigned the assignment**

**SELECT COUNT(\*) INTO v\_submission\_count**

**FROM AssignedTo**

**WHERE Student\_ID = p\_student\_id AND Assignment\_ID = p\_assignment\_id;**

**IF v\_submission\_count = 0 THEN**

**SELECT 'Student is not assigned this assignment' AS Error;**

**ELSE**

**-- Check if the due date has passed**

**SELECT Due\_Date, Teacher\_ID INTO v\_due\_date, v\_teacher\_id**

**FROM Assignments**

**WHERE Assignment\_ID = p\_assignment\_id;**

**IF CURRENT\_DATE > v\_due\_date THEN**

**SELECT 'Due date has passed' AS Error;**

**ELSE**

**-- Check if the student has already submitted the assignment**

**SELECT COUNT(\*) INTO v\_submission\_count**

**FROM submissions**

**WHERE Student\_ID = p\_student\_id AND Assignment\_ID = p\_assignment\_id;**

**IF v\_submission\_count > 0 THEN**

**SELECT 'Student has already submitted this assignment' AS Error;**

**ELSE**

**-- Add the submission**

**INSERT INTO submissions (File, Date\_Submitted, Assignment\_ID, Student\_ID, Teacher\_ID)**

**VALUES (p\_file, CURRENT\_DATE, p\_assignment\_id, p\_student\_id, v\_teacher\_id);**

**-- Update the status in the AssignedTo table**

**UPDATE AssignedTo**

**SET Status = TRUE**

**WHERE Student\_ID = p\_student\_id AND Assignment\_ID = p\_assignment\_id;**

**SELECT 'Submission added successfully' AS Success;**

**END IF;**

**END IF;**

**END IF;**

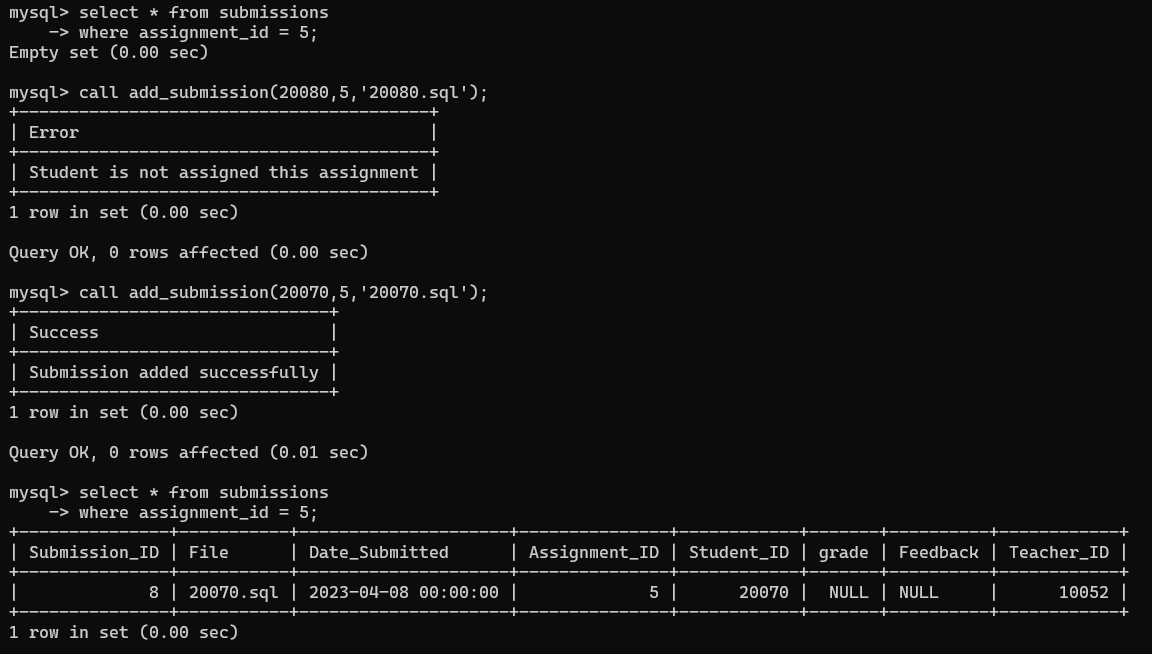
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL add\_submission(student\_id, assignment\_id, 'file.txt');**

****

**7. GRADE\_SUBMISSION**

**-- This procedure updates the grade and feedback for a submission if the teacher grading is the one who created the assignment.**

**DELIMITER $$**

**CREATE PROCEDURE grade\_submission(**

**IN p\_teacher\_id INT,**

**IN p\_submission\_id INT,**

**IN p\_grade INT,**

**IN p\_feedback TEXT**

**)**

**BEGIN**

**DECLARE v\_assignment\_id INT;**

**DECLARE v\_assignment\_count INT;**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Get the assignment ID for the submission**

**SELECT Assignment\_ID INTO v\_assignment\_id**

**FROM submissions**

**WHERE Submission\_ID = p\_submission\_id;**

**-- Check if the teacher created the assignment**

**SELECT COUNT(\*) INTO v\_assignment\_count**

**FROM Assignments**

**WHERE Assignment\_ID = v\_assignment\_id AND Teacher\_ID = p\_teacher\_id;**

**IF v\_assignment\_count = 0 THEN**

**SELECT 'Teacher did not create this assignment' AS Error;**

**ELSE**

**-- Update the grade and feedback for the submission**

**UPDATE submissions**

**SET Grade = p\_grade, Feedback = p\_feedback**

**WHERE Submission\_ID = p\_submission\_id;**

**SELECT 'Grade and feedback updated successfully' AS Success;**

**END IF;**

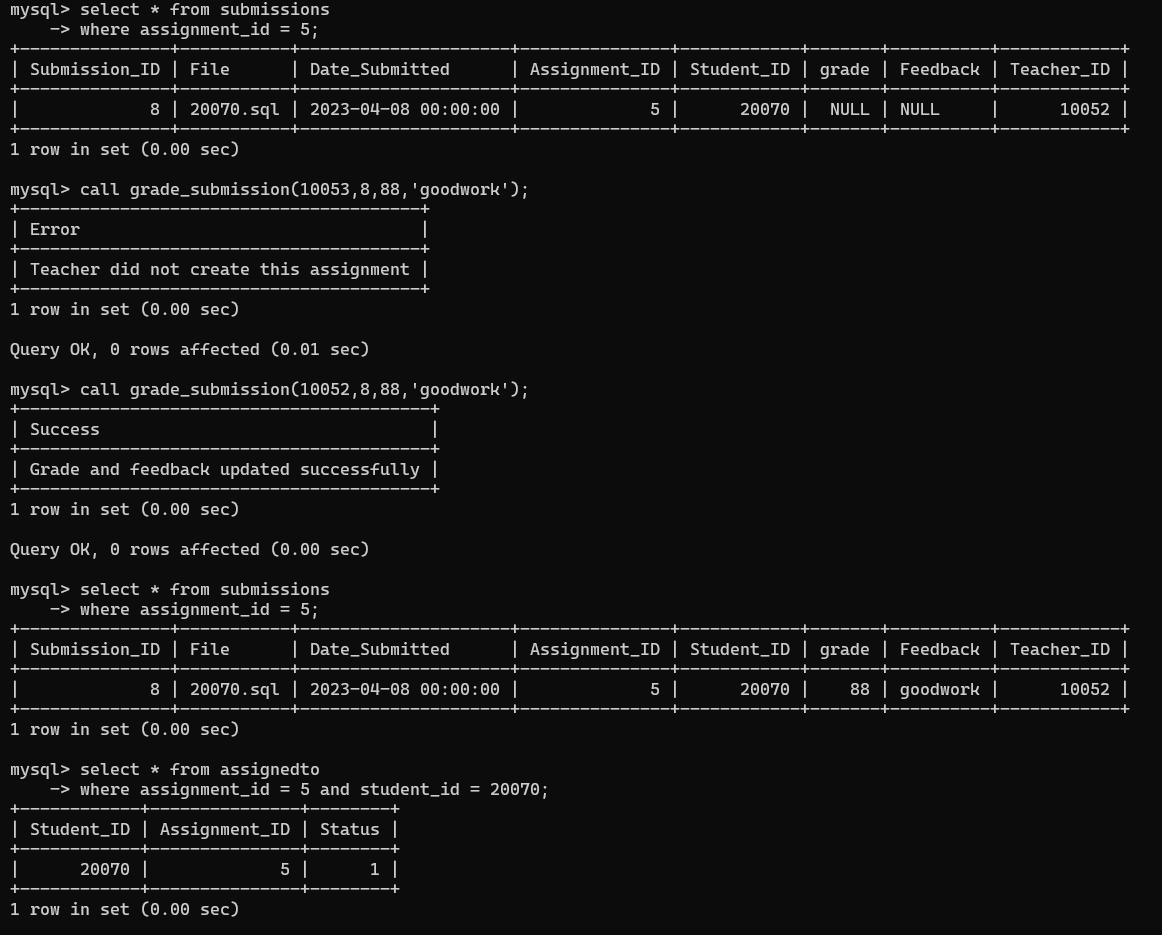
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL grade\_submission(teacher\_id, submission\_id, grade, 'feedback');**

****

**8. ASSIGNMENT\_COMPLETION\_RATE**

**-- This code calculates the completion rate for an assignment.**

**DELIMITER $$**

**CREATE PROCEDURE assignment\_completion\_rate(**

**IN p\_assignment\_id INT**

**)**

**BEGIN**

**DECLARE v\_total\_students INT;**

**DECLARE v\_completed\_students INT;**

**DECLARE v\_assignment\_count INT;**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Check if the assignment exists**

**SELECT COUNT(\*) INTO v\_assignment\_count**

**FROM Assignments**

**WHERE Assignment\_ID = p\_assignment\_id;**

**IF v\_assignment\_count = 0 THEN**

**SELECT 'Assignment does not exist' AS Error;**

**ELSE**

**-- Get the total number of students assigned the assignment**

**SELECT COUNT(\*) INTO v\_total\_students**

**FROM AssignedTo**

**WHERE Assignment\_ID = p\_assignment\_id;**

**-- Get the number of students who have completed the assignment**

**SELECT COUNT(\*) INTO v\_completed\_students**

**FROM AssignedTo**

**WHERE Assignment\_ID = p\_assignment\_id AND Status = TRUE;**

**-- Calculate and return the completion rate**

**SELECT (v\_completed\_students / v\_total\_students) \* 100 AS CompletionRate;**

**END IF;**

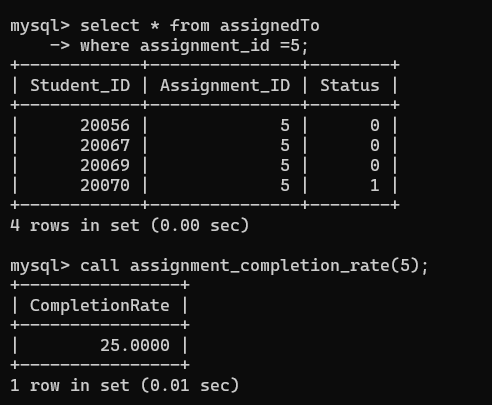
**-- Commit the transaction**

**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL assignment\_completion\_rate(assignment\_id);**

****

**9. UPDATE\_TEACHER\_PROFILE**

**-- This code updates the profile of a teacher.**

**DELIMITER $$**

**CREATE PROCEDURE update\_teacher\_profile (**

**IN TeacherEmail VARCHAR(100),**

**IN Education VARCHAR(100),**

**IN ResearchInterest VARCHAR(100),**

**IN ContactNo VARCHAR(20)**

**)**

**BEGIN**

**-- Start a new transaction**

**START TRANSACTION;**

**-- Update the teacher's profile**

**UPDATE Teachers**

**SET Education = Education,**

**Research\_Interest = ResearchInterest,**

**Contact\_No = ContactNo**

**WHERE E\_MAIL = TeacherEmail;**

**-- Return a success message**

**SELECT 'Profile updated successfully' AS Success;**

**-- Commit the transaction**

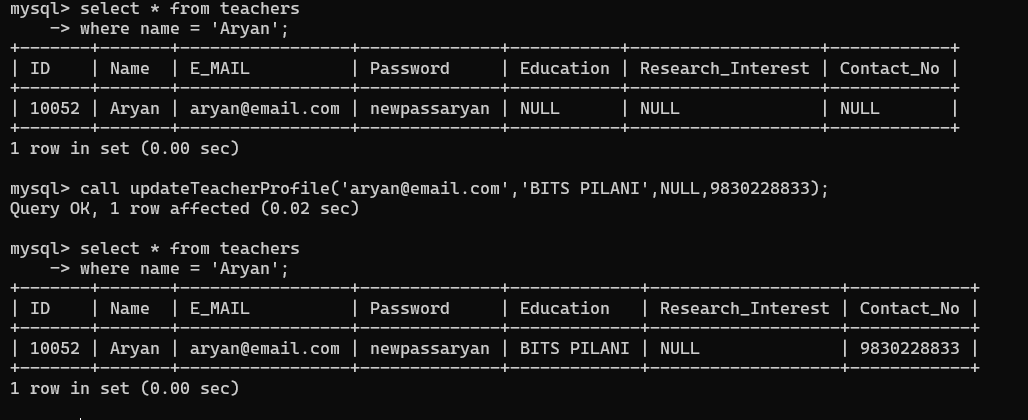
**COMMIT;**

**END$$**

**-- Example query to call the procedure:**

**-- CALL UpdateTeacherProfile('teacher@example.com', 'Education', 'Research Interest', 'ContactNo');**

**-- NOTE: Teacher has flexibility to update some of the fields by putting the rest as NULL.**

****

**10. ASSIGNMENT\_STATISTICS**

**-- This procedure calculates the average, lowest and highest grades for an assignment.**

**DELIMITER //**

**CREATE PROCEDURE assignment\_statistics(**

**IN p\_assignment\_id INT**

**)**

**BEGIN**

**-- Calculate and return the average, lowest and highest grades for the assignment**

**SELECT AVG(Grade) AS Average,**

**MIN(Grade) AS Lowest,**

**MAX(Grade) AS Highest**

**FROM submissions**

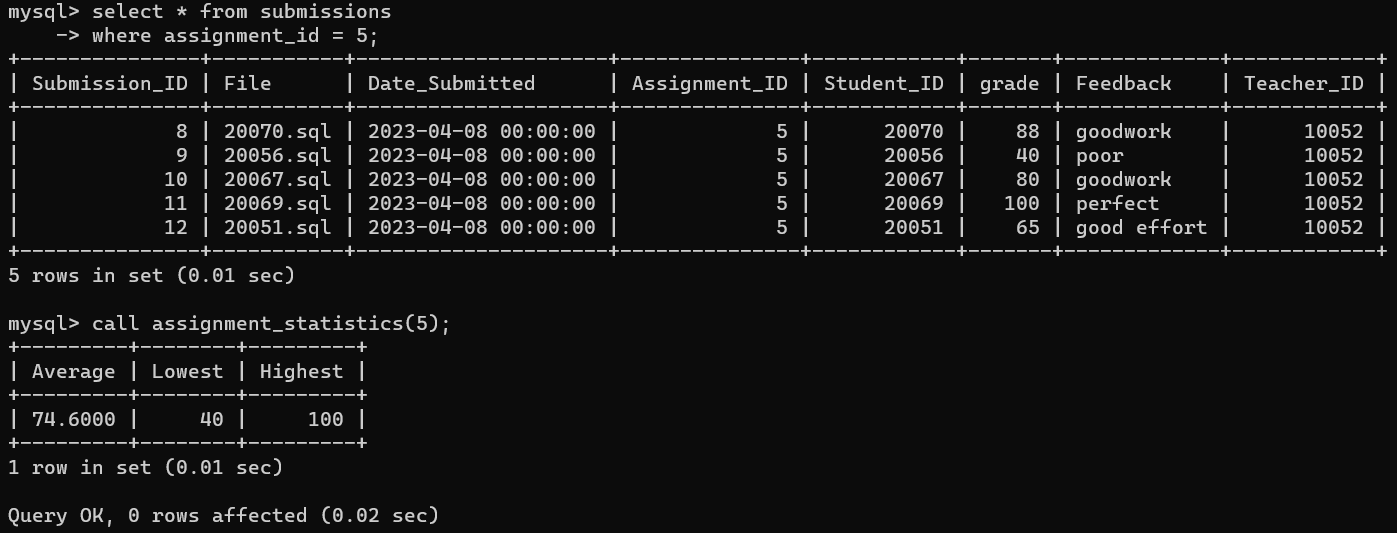
**WHERE Assignment\_ID = p\_assignment\_id;**

**END //**

**DELIMITER ;**

**-- Example query to call the procedure:**

**-- CALL assignment\_statistics(assignment\_id);**

****