Advance SQL Procedures

Centralized College Database

Riddhi Bhatt & Urvi Vaidya

Index

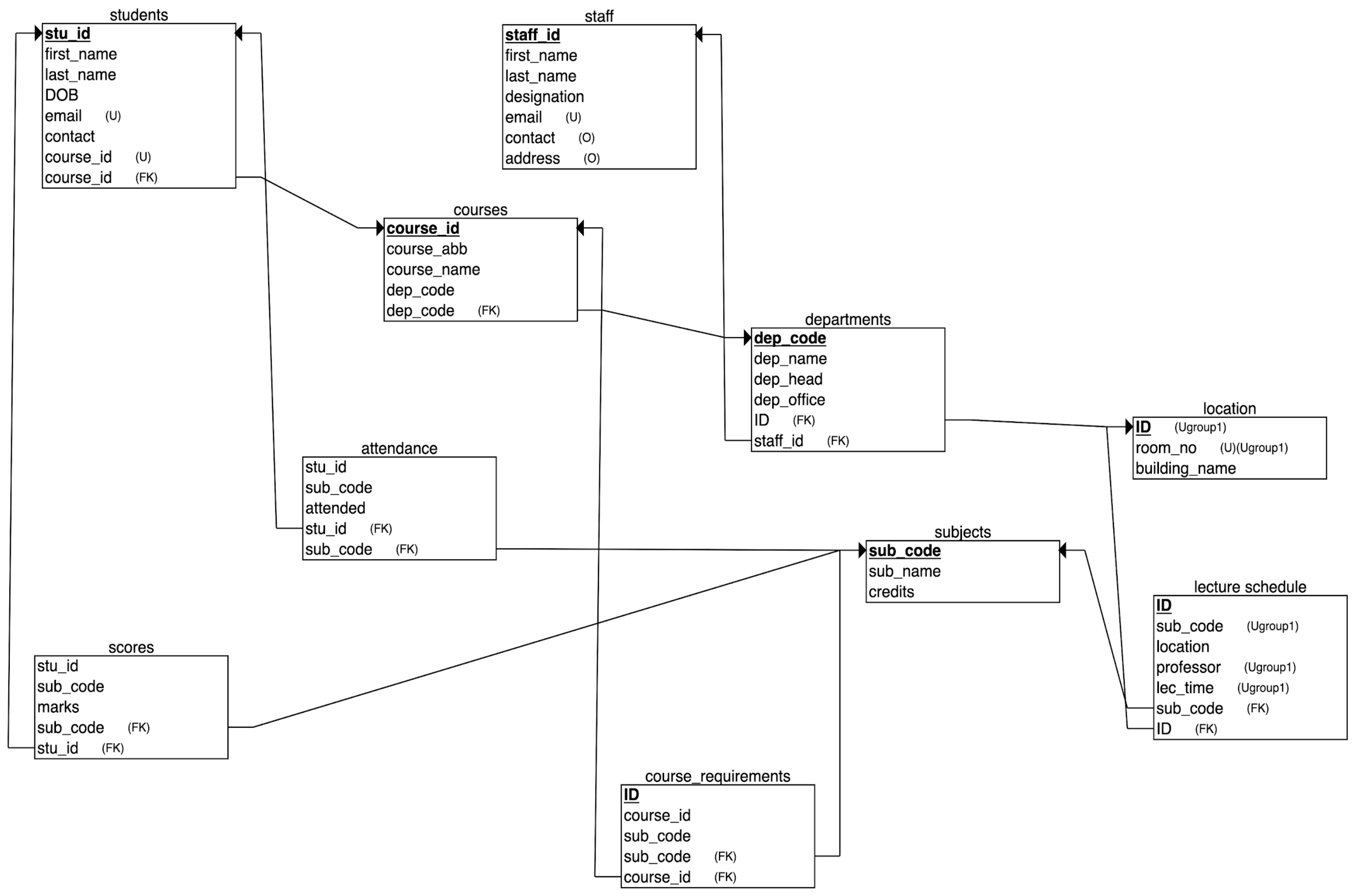
Sr.No	Particulars	Page No
1.	Project Summary	3
2.	ER Diagram	4
3.	Procedures	5
4.	Learnings - Riddhi Bhatt	
5.	Learnings - Urvi Vaidya	
6.	Bibliography	

Project Summary

This project aims at creating stored procedures to enable the user to update, edit or log queries quickly. Procedures act like functions and allow the user to execute the same query for multiple inputs without having to type out the query every single time. We continue to work on the centralized college database, which is a repository of data for a college, which we created as a part of our assignment for semester one. The said database will form the basis of our present project upon which we will build our procedures. The procedures which we will showcase which include the use or triggers, case statements, loops, exception handling, if-else statements, etc. are just a small example of what we can do. For example, if we want to get details of a particular group of students who belong to a specific course, we can create a procedure for pulling the data, so that we need only call the function with the course name as our input. Similarly if we are looking for the contact information of a specific group of students we can do that by using just the course code as our input. There are many such uses of procedures that make the users life easier. Updating student or course details can also be achieved by using stored procedures. We split the requirements between us and came up with our procedures and then critiqued each other's work and helped refine the final procedures. Riddhi took up; loops, if-else statements and triggers whereas Urvi took up ; case statements, exception handling and cursors. We will now look at the database schema and then the procedures themselves.

ER Diagram

We have reproduced the structure of our database so that it is easier to refer to and understand the various parameters and variables in our procedures. This is the most important piece of the puzzle that helps the user tie in all the information.



Procedures

1) Procedure 1 - Exception Handling

```
-- PROCEDURE 1 - EXCEPTION HANDLING - SHOWING ERROR IF ENTERING DUPLICATE VALUES

DELIMITER $$

CREATE PROCEDURE AddLocation(
IN inroom INT,
IN inbuilding VARCHAR(255)
)
BEGIN
     -- exit if the duplicate key occurs
     DECLARE EXIT HANDLER FOR 1062
     BEGIN
        SELECT CONCAT('Error, duplicate key (',inroom,',',inbuilding,') occurred') AS message;
     END;

     -- execute the procedure if there is no error
     INSERT INTO location (room_no, building_name)
     VALUES (inroom, inbuilding);

     -- return the updated table with the new entry
     SELECT * FROM location ORDER BY building_name;
END$$

DELIMITER ;
```

This procedure allows the user to add new locations to the database, in the event a new building is constructed or new rooms are added to existing buildings. In this procedure if the user tries to enter an existing room number in the same building the procedure will terminate and the error message will be shown to the user, Alternatively, if a new entry is added that is not repetitive then the same will be updated in the database and the procedure will display the entire location table with the new entry. This procedure takes the Room No (INT) and Building Name (Varchar) as input.

Usage Syntax : CALL AddLocation(RoomNo, Building Name)

Try it yourself:

CALL AddLocation(4, 'Tata Memorial Building')

CALL AddLocation(7, 'Birla Building')

2) Procedure 2 - Simple Procedure for displaying course requirements.

```
-- PROCEDURE 2 - SIMPLE PROCEDURE TO SHOW SUBJECTS OF COURSE REQUIREMENTS

DELIMITER $$

CREATE PROCEDURE GetCourseRequirements(
IN incourse_abb VARCHAR(255))

BEGIN

-- getting course requirements
SELECT c.course_abb, c.course_name, sub.sub_name,
sub.credits FROM courses as c  JOIN course_requirements as cr ON
cr.course_id = c.course_id JOIN subjects as sub ON cr.sub_code = sub.sub_code WHERE course_abb = incourse_abb;

END$$

DELIMITER ;
```

This is a procedure for displaying subjects required for a particular course. It takes as input the course abbreviation and displays the required subjects along with their credits.

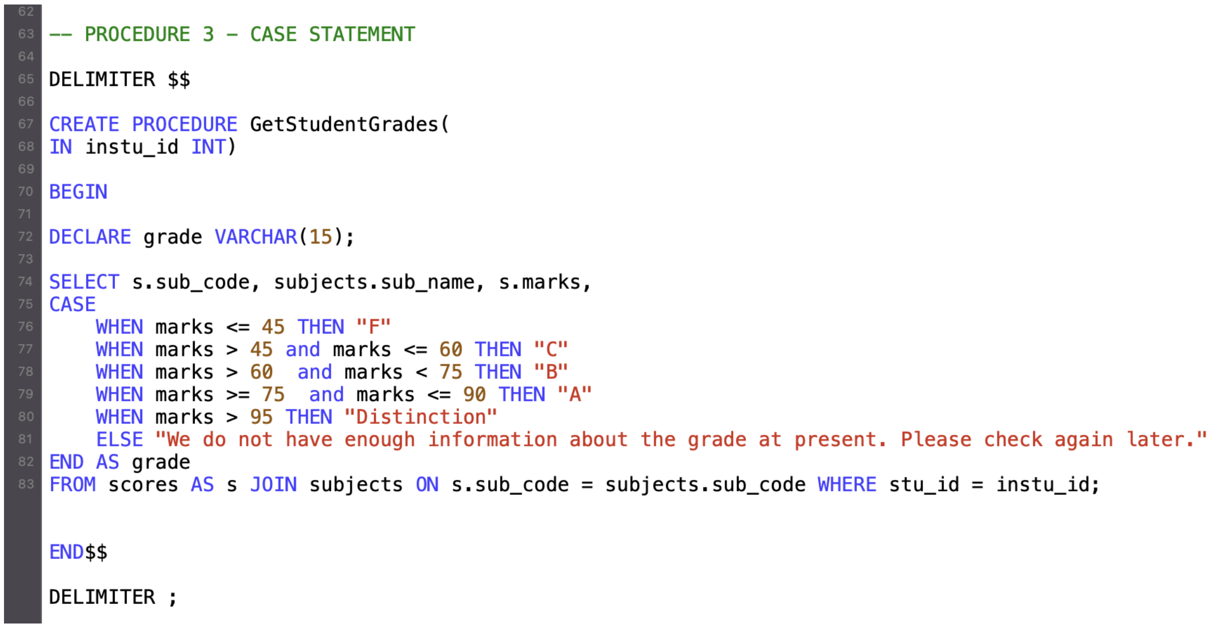
Usage Syntax : CALL GetCourseRequirements(CourseAbbreviation)

Try it yourself:

CALL GetCourseRequirements(‘MBBS’)

CALL GetCourseRequirements(‘LLB’)

3) Procedure 3 - Case Statement



```
-- PROCEDURE 3 - CASE STATEMENT

DELIMITER $$

CREATE PROCEDURE GetStudentGrades(
IN instu_id INT)

BEGIN

DECLARE grade VARCHAR(15);

SELECT s.sub_code, subjects.sub_name, s.marks,
CASE
    WHEN marks <= 45 THEN "F"
    WHEN marks > 45 and marks <= 60 THEN "C"
    WHEN marks > 60  and marks < 75 THEN "B"
    WHEN marks >= 75  and marks <= 90 THEN "A"
    WHEN marks > 95 THEN "Distinction"
    ELSE "We do not have enough information about the grade at present. Please check again later."
END AS grade
FROM scores AS s JOIN subjects ON s.sub_code = subjects.sub_code WHERE stu_id = instu_id;


END$$

DELIMITER ;
```

This procedure displays the students' grade in each subject. The grades are not inherently stored in the database, therefore the procedure uses case statements and the students' marks to evaluate their grade. The procedure takes as input the student id and displays the students marks and grade in all the subjects.

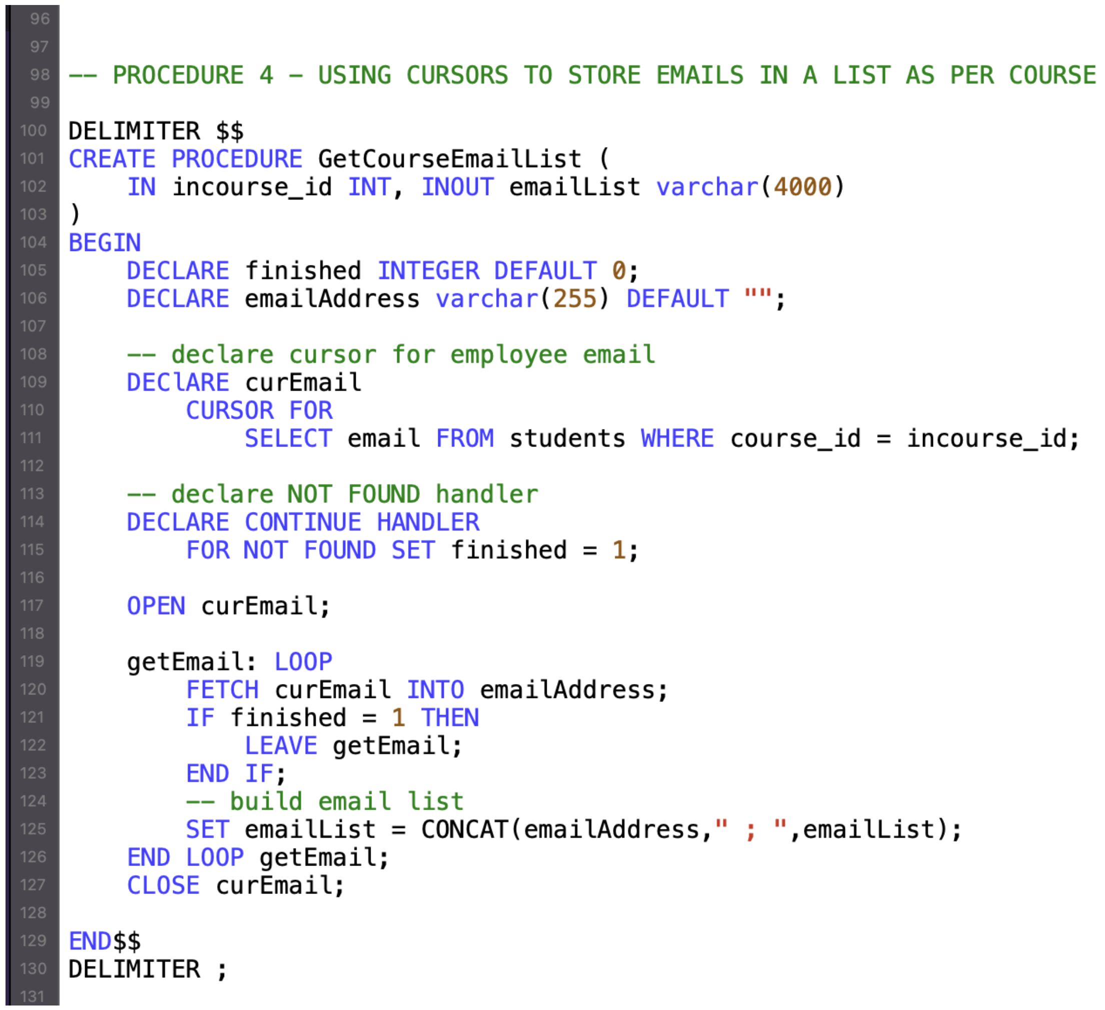
Usage Syntax : CALL GetStudentGrades(StudentID)

Try it yourself:

CALL GetStudentGrades(1)

CALL GetStudentGrades(33)

4) Procedure 4 - Cursors and Loop



```


-- PROCEDURE 4 - USING CURSORS TO STORE EMAILS IN A LIST AS PER COURSE

DELIMITER $$
CREATE PROCEDURE GetCourseEmailList (
    IN incourse_id INT, INOUT emailList varchar(4000)
)
BEGIN
    DECLARE finished INTEGER DEFAULT 0;
    DECLARE emailAddress varchar(255) DEFAULT "";

    -- declare cursor for employee email
    DECLARE curEmail
        CURSOR FOR
            SELECT email FROM students WHERE course_id = incourse_id;

    -- declare NOT FOUND handler
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET finished = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finished = 1 THEN
            LEAVE getEmail;
        END IF;
        -- build email list
        SET emailList = CONCAT(emailAddress," ; ",emailList);
    END LOOP getEmail;
    CLOSE curEmail;

END$$
DELIMITER ;

```

This procedure takes as input the course ID and gives us an output list with the email addresses of all the students registered for the relevant course. This procedure can be used to help administrative staff create contact lists and email groups.

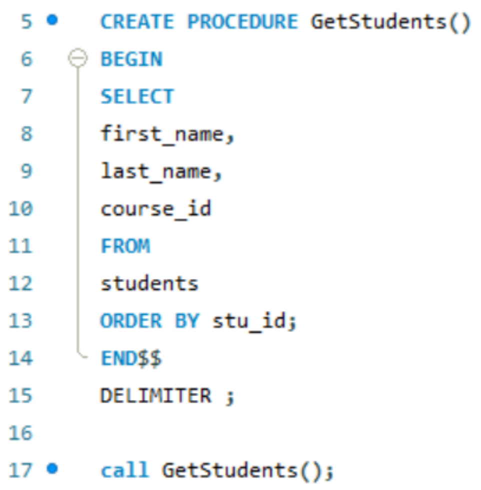
Usage Syntax : CALL GetCourseEmailList(CourseID)

Try it yourself:

CALL GetCourseEmailList(110)

CALL GetCourseEmailList(111)

5) Procedure 5 - Simple Statement



```
CREATE PROCEDURE GetStudents()
BEGIN
SELECT
first_name,
last_name,
course_id
FROM
students
ORDER BY stu_id;
END$$
DELIMITER ;

call GetStudents();
```

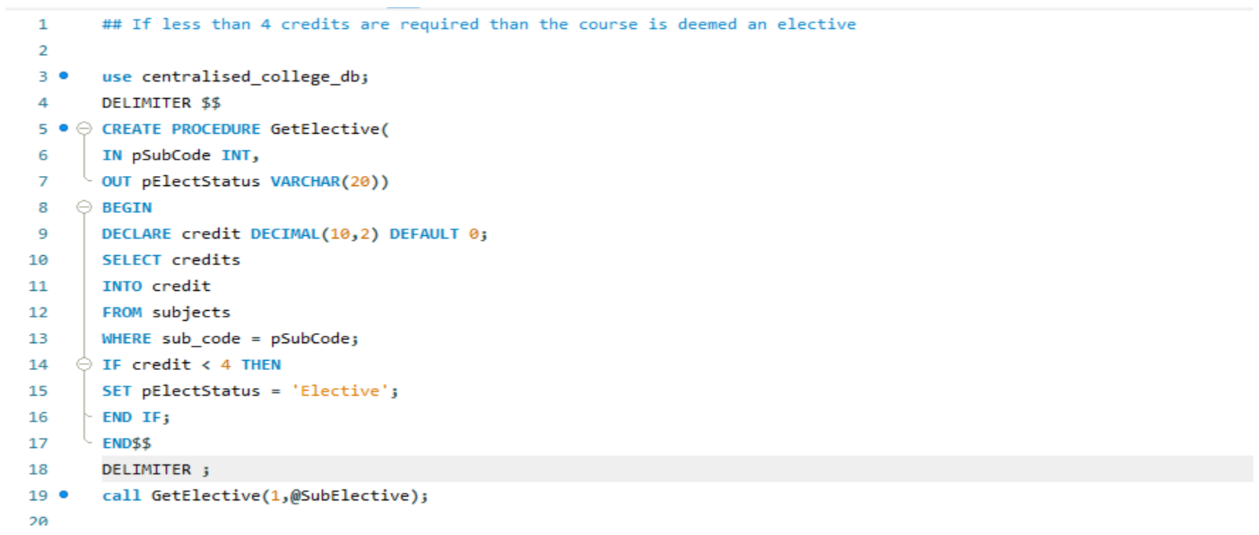
This is a procedure for displaying the name and course id of each student. It takes no input and selects all values from the students table and displays the student name along with their course id..

Usage Syntax : CALL GetStudents();

Try it yourself:

CALL GetStudents();

6) Procedure 6 - If-Else Statement



```
## If less than 4 credits are required than the course is deemed an elective

use centralised_college_db;
DELIMITER $$
CREATE PROCEDURE GetElective(
IN pSubCode INT,
OUT pElectStatus VARCHAR(20))
BEGIN
DECLARE credit DECIMAL(10,2) DEFAULT 0;
SELECT credits
INTO credit
FROM subjects
WHERE sub_code = pSubCode;
IF credit < 4 THEN
SET pElectStatus = 'Elective';
END IF;
END$$
DELIMITER ;
call GetElective(1,@SubElective);
```

This is a procedure for displaying whether the course that is input in the procedure is an elective course (or a core course). It takes the course subject code as an input and gives the output as “Elective” if that code is of a course with credits less than 4.

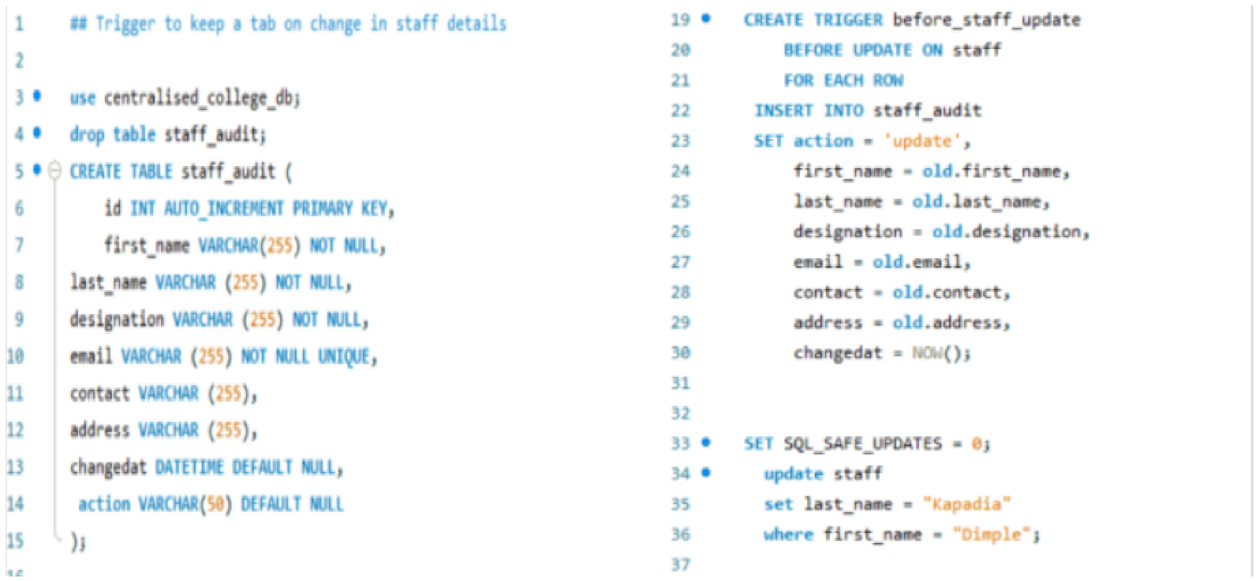
Usage Syntax : CALL GetElective(<subject code>,@SubElective);

Try it yourself:

CALL GetElective(1,@SubElective);

CALL GetElective(4,@SubElective);

7) Procedure 7 - Trigger



```
## Trigger to keep a tab on change in staff details

use centralised_college_db;
drop table staff_audit;
CREATE TABLE staff_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
last_name VARCHAR (255) NOT NULL,
designation VARCHAR (255) NOT NULL,
email VARCHAR (255) NOT NULL UNIQUE,
contact VARCHAR (255),
address VARCHAR (255),
changedat DATETIME DEFAULT NULL,
 action VARCHAR(50) DEFAULT NULL
);
```

```
CREATE TRIGGER before_staff_update
    BEFORE UPDATE ON staff
    FOR EACH ROW
 INSERT INTO staff_audit
 SET action = 'update',
     first_name = old.first_name,
     last_name = old.last_name,
     designation = old.designation,
     email = old.email,
     contact = old.contact,
     address = old.address,
     changedat = NOW();


SET SQL_SAFE_UPDATES = 0;
  update staff
  set last_name = "Kapadia"
  where first_name = "Dimple";
```

This procedure tracks the staff table and BEFORE any changes are made to the details of any staff members it stores the original record in a table called staff-audit.

Usage Syntax : UPDATE staff <update to be made>;
Try it yourself: UPDATE staff set last_name = “Kapadia”
Where first_name = “Dimple”;

Learnings - Riddhi Bhatt

The project has helped me fully understand the added advantages of PLSQL over the traditional Structured query language. Other programming languages had much more flexibility compared to SQL due to which SQL was restricted to basic functionality in database management. PL/SQL has added the edge needed by adding functionalities like stores procedures, functions etc that greatly improve the flexibility to get more out of the database.
I have personally learnt the importance of getting the syntax right and understanding the general structure of different types of plsql blocks without which its nearly impossible to successfully execute the plsql comments given the esoteric nature of the language.

Learnings - Urvi Vaidya

This project has helped me understand how we can query the database quite easily for multiple inputs without having to write the query each time. The procedures allow us to create functions in the same way that we would for any programming language like python or C++. We can use the functionality of a programming language in the database itself. I have also realized that we have to be very methodical in writing our procedures and have to test them with various inputs to make sure they work correctly for each. I have also learnt to look at the procedures from a different perspective, being that of my partner. At the end of the day the procedure must be easy to use and understand and that there are various ways of achieving the same output.

Bibliography

1) https://dev.mysql.com
2) https://www.mysqltutorial.org
3) https://www.w3schools.com
4) stackexchange