

**Department of Information and Communication Technology Faculty of Technology**

**University of Ruhuna**

**Database Management Systems ICT 1222**

**Assignment 02**

Group 10

Submitted to: Mr.P.H.P. Nuwan Laksiri

Submitted by: TG/2021/1010 L.S.R. Vidanarachchi

* **Introduction about the problem/group project**

Faculty of Technology, University of Ruhuna, Learning Management System contains many features that explain many functionalities for a university LMS. This project is based upon that LMS system and aims to manifest a replica of the back-end database of the FoT LMS.

The main concerns for a LMS is to manage student data. That main concern can be categorized into even more sub concerns.

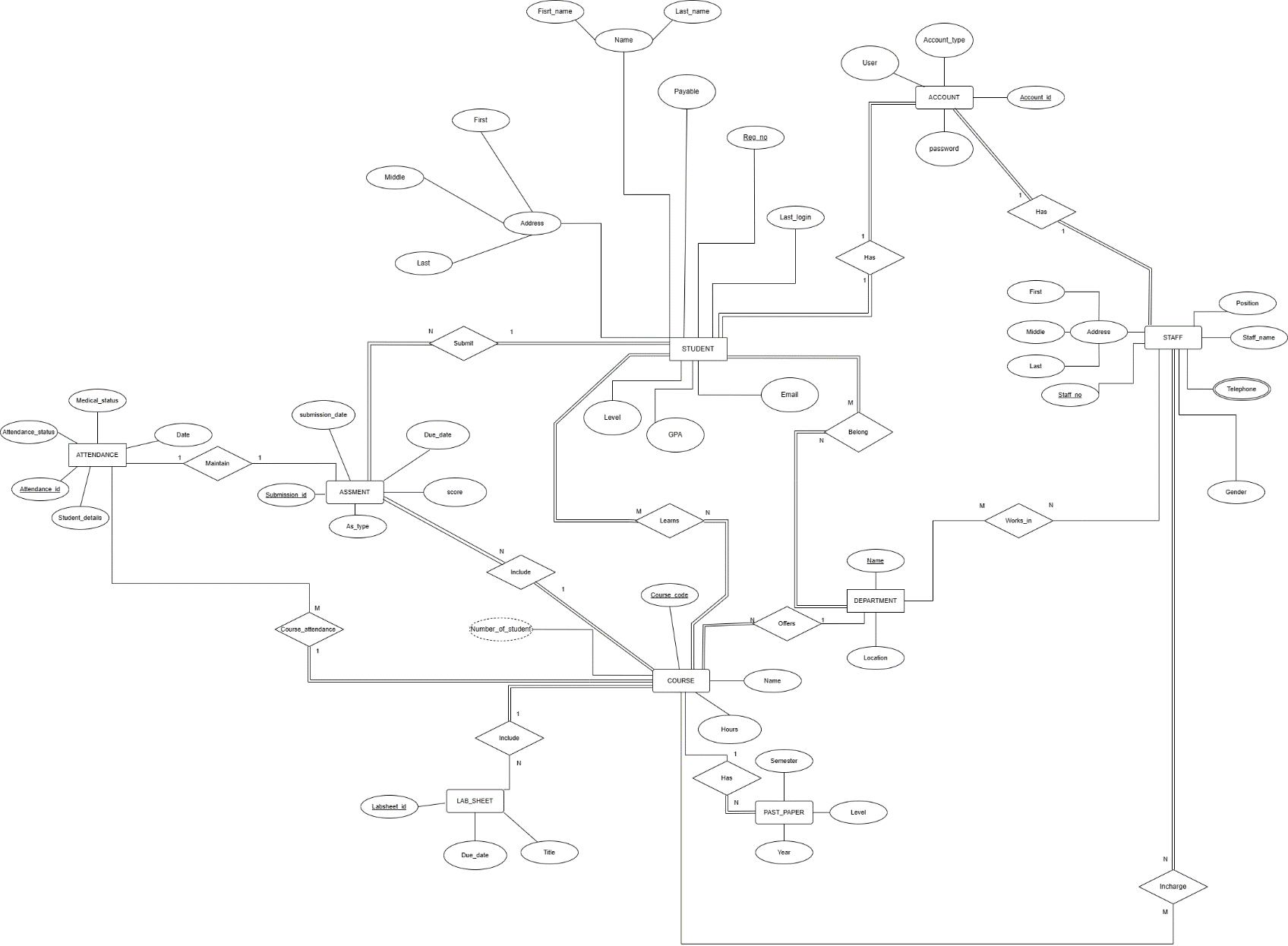
1. Should be able to manage all academic and non-academic personal data separately.
2. Should be able to handle course data and related student data.
3. Should be able to manage academic documents.
4. Should be able to handle attendance data.
5. Should be able to manage and handle different submissions separately.
6. One administrator should be able to manage the entire database.
7. A hierarchy of privileges to manage and access the database is crucial.

* **Brief introduction to the solution**

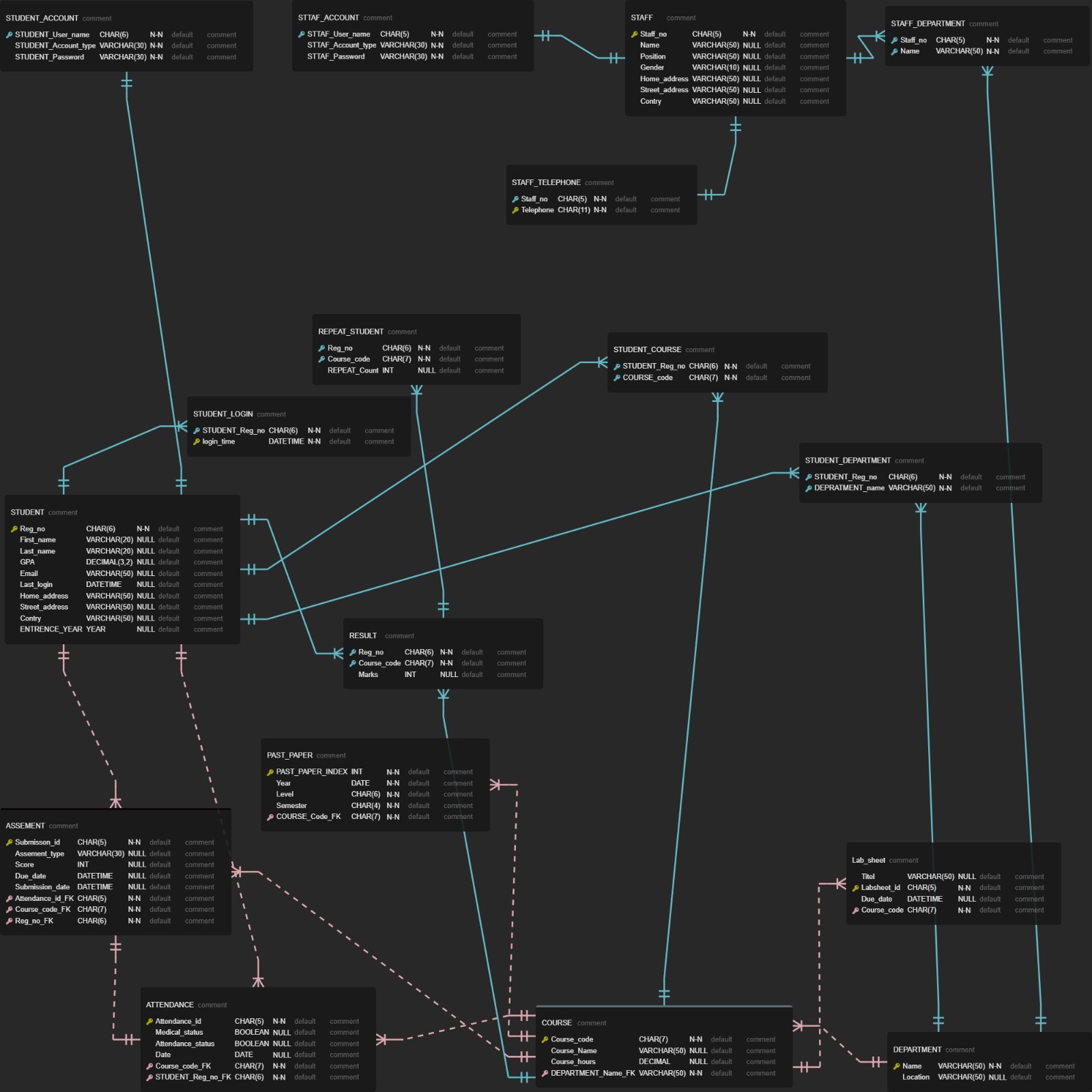
The solution that was developed to meet the requirements mentioned above fulfills those requirements by providing,

1. Separate student related data tables and staff related data tables to manage data separately.
2. Different course tables to separate information for more clarity.
3. Different forms of academic assessments are given different tables with separate tables for results.
4. A past paper table for managing past paper separately from other academic documents.
5. Due to implementing a separate attendance table allows to log all attendance for different courses.
6. Assement table for marking academic sessions and labsheet table for non-marking academic sessions.
7. Different procedure based view systems to enhance security and understandability of the solution.

* **Proposed ER/EER diagram**

****

* **Proposed Relational mapping diagram**

****

* **Table structure the solution**

1. **STUDENT** tableis acts as central hub to many tables such as **STUDENT\_ACCOUNT, STUDENT\_LOGIN, REPEAT\_STUDENT, STUDENT\_COURSE, STUDENT\_DEPARTMENT, COURSE** and **REPEAT\_STUDENT** which contains different data but tables are connected together.
2. **DEPARTMENT** table is also connected to tables such as **STUDENT\_DEPARTMENT, STAFF\_DEPARTMENT.**
3. **STAFF** table holds strings to **STAFF\_ACCOUNT, STAFF\_DEPARTMENT** and **STAFF\_TELEPHONE** tables.
4. **COURSE** table connects to **RESULT** table as well as **STUDENT\_COURSE** table.
5. **ASSEMENT, ATTENDANCE,** and **PAST\_PAPER** tables connects to many tables because data within those tables corresponds many data.
6. As for internal table structure regarding data types are used for data to be consistent.

* **Architecture of the solution**

1. **Database Management System –** A DBMS software is used to implement and maintain this database.
2. **Tables** – Tables are the most important part of the architecture which holds all data.
3. **Views** – Views are used to encapsulate data to prevent unnecessary data leaks.

* **Tools and technologies that were used**

1. DrawIO online tool – To draw ER diagram.
2. ERD EDITOR Visual Studio Code extention – To draw the relational schema.
3. Microsoft Word – To create the SRS report and the final report.
4. Microsoft Teams – Using video conferencing technology to create the database coordinating with each other.
5. XAMPP with MYSQL – To develop the database this API was used.
6. GitHub – Acts as a portfolio to show the progress route of the database development.

* **Security measures that have taken to protect your DB**

Mainly creating different users with different access levels are used to give security to the database. **Least privilege principle** is thoroughly followed to maximize the security for the database.

* **Brief description about DB Accounts/Users and the reasons for creating such Accounts/Users**

1. **Admin –** This user account is the only person with each and every access method to the database and the only person that can give others access privileges to the database.
2. **Dean –** This user has the same privileges as the Adminbut cannot give other privilege access to the database. It’s not considered in the standard operating procedures for the Dean to act as the distributor of privileges while an Admin user is present.
3. **Lecturer –** Lecturer only requires the privilege to concern himself with academic data manipulation so no need to create users or other administrator level functionalities.
4. **Technical Officer –** The job of a technical officer is to manage attendance details so in order to increase data security and ease the job of user in subject only access to attendance related tables are given.
5. **Student –** Student only requires to access academic related data and to find out whether the student is eligible for the final examination and the final results of the examination, thus tables such as attendance, lab sheets and some derived views of marks and results are required.

* **Code snippets to support your work**

This section contains tools that were developed within the database to assist the users to query data more effectively and efficiently. While stored procedures simplifies querying and views eases understanding and encapsulates data.

**Stored procedures**

**TG1010**

/////////////////// To Get Course This Student Enrollment /////////////////////

DELIMITER //

CREATE PROCEDURE GetStudentCourses(IN studentRegNo CHAR(6))

BEGIN

SELECT c.course\_code, c.name

FROM Student\_Course sc

JOIN course c ON sc.COURSE\_code = c.course\_code

WHERE sc.STUDENT\_Reg\_no = studentRegNo;

END //

DELIMITER ;

/////////////// To get Upcoming Labsheet for This Student ///////////

DELIMITER //

CREATE PROCEDURE StudentUpcomeLab (IN studentRegNo CHAR(6))

BEGIN

SELECT ls.labsheet\_id, ls.title, ls.due\_date

FROM Student\_Course sc

JOIN labsheet ls ON sc.COURSE\_code = ls.course\_code

WHERE sc.STUDENT\_Reg\_no = studentRegNo AND ls.due\_date > NOW();

END //

DELIMITER ;

/////////////// To get Previous Labsheet for This Student ///////////

DELIMITER //

CREATE PROCEDURE StudentPreLab (IN studentRegNo CHAR(6))

BEGIN

SELECT ls.labsheet\_id, ls.title, ls.due\_date

FROM Student\_Course sc

JOIN labsheet ls ON sc.COURSE\_code = ls.course\_code

WHERE sc.STUDENT\_Reg\_no = studentRegNo AND ls.due\_date < NOW();

END //

DELIMITER ;

///////////////// To add Labsheet as Lecture ////////////////////

DELIMITER //

CREATE PROCEDURE AddLabSheet(

IN labsheet\_id CHAR(9),

IN title VARCHAR(255),

IN due\_date DATETIME,

IN course\_code CHAR(7)

)

BEGIN

INSERT INTO labsheet (labsheet\_id, title, due\_date, course\_code)

VALUES (labsheet\_id, title, due\_date, course\_code);

END //

DELIMITER ;

CALL AddLabSheet('ICT424310', 'TEST Lab Sheet', '2023-12-31 14:00:00', 'ICT4243');

///////////////// To add Past Papers as Lecture ////////////////////

DELIMITER //

CREATE PROCEDURE AddPastpapers(

IN paper\_title VARCHAR(255),

IN year YEAR,

IN level CHAR(6),

IN semester CHAR(4),

IN course\_code CHAR(7)

)

BEGIN

INSERT INTO Past\_Papers (paper\_title, year, level, semester, course\_code)

VALUES (paper\_title, year, level, semester, course\_code );

END //

DELIMITER ;

CALL AddPastPapers ('TEST Paper 1', 2022, 'level4', 'sem1', 'ICT4243');

**TG1030**

01. Procedure to change department name and the course code for a given student tg number.

DELIMITER //

CREATE PROCEDURE updateacademics(IN Reg\_No CHAR(5), IN dep\_name VARCHAR(10), IN course\_code CHAR(7))

BEGIN

UPDATE student\_department SET DEPARTMENT\_name=dep\_name WHERE STUDENT\_Reg\_no=Reg\_NO;

UPDATE student\_course SET COURSE\_code=course\_code WHERE STUDENT\_Reg\_no = Reg\_No;

END//

DELIMITER ;

CALL updateacademics("TG0200", "ICT", "ICR4243");

02. Procedure to insert a student record

DELIMITER //

CREATE PROCEDURE insertstudent(IN Reg\_no CHAR(6), IN First\_name VARCHAR(25), IN Last\_name VARCHAR(25),IN Email VARCHAR(30),IN Home\_address VARCHAR(30),IN Street\_address VARCHAR(50), IN Country VARCHAR(25))

BEGIN

SELECT 'Example format- ('TG9999', 'First\_name', 'Last\_name', 'person@gmail.com', 'No99', 'Province/city - Road name', 'Country name');' As 'Format';

INSERT INTO student(Reg\_no, First\_name, Last\_name, Email, Home\_address, Street\_address, country)

VALUES(Reg\_no, First\_name, Last\_name, Email, Home\_address, Street\_address, country);

UPDATE Student

SET Entrance\_year = '2018'

WHERE Reg\_No BETWEEN 'TG0200' AND 'TG0399'

ORDER BY Reg\_No ASC;

UPDATE Student

SET Entrance\_year = '2019'

WHERE Reg\_No BETWEEN 'TG0400' AND 'TG0599'

ORDER BY Reg\_No ASC;

UPDATE Student

SET Entrance\_year = '2020'

WHERE Reg\_No BETWEEN 'TG0600' AND 'TG0999'

ORDER BY Reg\_No ASC;

UPDATE Student

SET Entrance\_year = '2021'

WHERE Reg\_No BETWEEN 'TG1000' AND 'TG1200'

ORDER BY Reg\_No ASC;

END//

DELIMITER ;

CALL insertStudent('TG1067', 'Tharindu', 'Sithum', 'TharidSitu88@example.com', '123 Main St', 'Mathale - Temple Road', 'Sri Lanka');

03. A procedure to select students for a given street address

DELIMITER //

CREATE PROCEDURE stud\_place(IN province\_name VARCHAR(15))

BEGIN

SELECT 'Enter a city or a province starting from capital letters' AS 'Instructions';

SELECT Reg\_no As 'Student Register Number', First\_name AS 'Student First Name', Last\_name AS 'Student Last name'

FROM Student

WHERE Street\_address LIKE province\_name;

END//

DELIMITER ;

CALL stud\_place('Piliyandala');

04. Select the count of students for a given department

DELIMITER //

CREATE PROCEDURE stud\_count(IN Dep\_name VARCHAR(3))

BEGIN

SELECT COUNT(STUDENT\_Reg\_no) AS 'Number of Students over all levels within that department'

FROM Student\_department

WHERE DEPARTMENT\_name = Dep\_name;

END//

DELIMITER ;

CALL stud\_count('ICT');

05. Select the tg numbers of students for a given course module

DELIMITER //

CREATE PROCEDURE enrol\_stud(IN course\_code VARCHAR(7))

BEGIN

SELECT STUDENT\_Reg\_no AS 'Students enrolled to the given course'

FROM student\_course

WHERE COURSE\_code = course\_code;

END//

DELIMITER ;

CALL enrol\_stud('ICT4123');

06. Procedure to enter a course credit value

DELIMITER //

CREATE PROCEDURE Credit()

BEGIN

ALTER TABLE course ADD COLUMN Credits INT;

UPDATE course

SET Credits = CASE

WHEN course\_code = 'BST1113' THEN 1

WHEN course\_code = 'BST1222' THEN 2

WHEN course\_code = 'BST2113' THEN 3

WHEN course\_code = 'BST2213' THEN 4

WHEN course\_code = 'BST3133' THEN 1

WHEN course\_code = 'BST3222' THEN 2

WHEN course\_code = 'BST4113' THEN 3

WHEN course\_code = 'BST4222' THEN 4

WHEN course\_code = 'ENG1212' THEN 1

WHEN course\_code = 'ET1112' THEN 2

WHEN course\_code = 'ET1223' THEN 3

WHEN course\_code = 'ET2122' THEN 4

WHEN course\_code = 'ET2243' THEN 1

WHEN course\_code = 'ET3112' THEN 2

WHEN course\_code = 'ET3222' THEN 3

WHEN course\_code = 'ET4114' THEN 4

WHEN course\_code = 'ET4242' THEN 1

WHEN course\_code = 'ICT1112' THEN 2

WHEN course\_code = 'ICT1123' THEN 3

WHEN course\_code = 'ICT1242' THEN 4

WHEN course\_code = 'ICT1243' THEN 1

WHEN course\_code = 'ICT1244' THEN 2

WHEN course\_code = 'ICT1245' THEN 3

WHEN course\_code = 'ICT1246' THEN 4

WHEN course\_code = 'ICT1247' THEN 1

WHEN course\_code = 'ICT2112' THEN 2

WHEN course\_code = 'ICT2223' THEN 3

WHEN course\_code = 'ICT3112' THEN 4

WHEN course\_code = 'ICT3234' THEN 1

WHEN course\_code = 'ICT4123' THEN 2

WHEN course\_code = 'ICT4243' THEN 3

ELSE 4

END;

END //

DELIMITER ;

CALL Credit();

07. Add Grade Point Value Column

DELIMITER //

CREATE PROCEDURE Gpv()

BEGIN

ALTER TABLE result ADD GPV\_Value DECIMAL(3, 1);

UPDATE result SET GPV\_Value = ROUND(0.1 + (RAND() \* 3.8), 1);

END//

DELIMITER ;

CALL Gpv();

**TG1038**

DELIMITER $$

CREATE PROCEDURE `find\_Result\_of\_a\_course` (IN Course\_code char(7))

BEGIN

SELECT \* FROM result\_details r WHERE r.Course\_code= Course\_code ;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `find\_Result\_of\_a\_student` (IN STUDENT\_Reg\_no char(6))

BEGIN

SELECT \* FROM result\_details r

WHERE r.STUDENT\_Reg\_no = STUDENT\_Reg\_no ;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `find\_Result\_of\_student\_course` (IN STUDENT\_Reg\_no char(6),IN Course\_code char(7))

BEGIN

SELECT \* FROM result\_details r

WHERE r.STUDENT\_Reg\_no = STUDENT\_Reg\_no AND r.Course\_code= Course\_code ;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `find\_attendance\_for\_a\_batch`(IN batch INT)

BEGIN

SELECT a.\*

FROM attendence\_detalis a

JOIN student s ON s.Entrance\_year = batch AND s.Reg\_no = a.STUDENT\_Reg\_no;

END$$

DELIMITER ;

s

DELIMITER $$

CREATE PROCEDURE `find\_attendence\_of\_a\_student` (IN STUDENT\_Reg\_no char(6))

BEGIN

SELECT \* FROM attendence\_detalis a

WHERE a.STUDENT\_Reg\_no = STUDENT\_Reg\_no ;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `CA\_DETAILS\_of\_a\_COURSE` (IN Course\_code char(7))

BEGIN

select \* FROM ca\_eligibility c where c.Course\_code = Course\_code;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `CA\_DETAILS\_of\_a\_Course\_for\_STUDENT` (IN STUDENT\_Reg\_no char(6),IN Course\_code char(7))

BEGIN

select \* FROM ca\_eligibility c

where c.Course\_code = Course\_code AND c.STUDENT\_Reg\_no = STUDENT\_Reg\_no ;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `CA\_DETAILS\_of\_a\_\_STUDENT` (IN STUDENT\_Reg\_no char(6))

BEGIN

select \* FROM ca\_eligibility c

where c.STUDENT\_Reg\_no = STUDENT\_Reg\_no ;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `CA\_DETAILS\_of\_a\_\_batch` (IN batch INT(4))

BEGIN

SELECT c.\* FROM ca\_eligibility c,student s

WHERE s.Entrance\_year = batch AND s.Reg\_no = c.STUDENT\_Reg\_no;

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE `insert\_attendance\_data`(

IN Attendance\_id VARCHAR(5),

IN Medical\_status boolean,

IN Attendance\_status boolean,

IN Date DATE,

IN Course\_code CHAR(7),

IN STUDENT\_Reg\_no CHAR(6)

)

BEGIN

INSERT INTO attendance(Attendance\_id, Medical\_status, Attendance\_status, Date, Course\_code, STUDENT\_Reg\_no)

VALUES (Attendance\_id, Medical\_status, Attendance\_status, Date, Course\_code, STUDENT\_Reg\_no);

END$$

DELIMITER ;

**TG1056**

ACCOUNT DETAILS

DELIMITER //

CREATE PROCEDURE lectureAccount(IN lecid char(5))

BEGIN

DECLARE lec\_exists INT;

SELECT COUNT(\*) INTO lec\_exists FROM LectureAccount WHERE Staff\_user\_name = lecid;

IF lec\_exists = 0 THEN

SELECT "Lecture not found";

ELSE

SELECT "Successful account details";

SELECT \* FROM LectureAccount WHERE Staff\_user\_name = lecid;

END IF;

END//

DELIMITER ;

ACCOUNT PASSWORD CHANGE

DELIMITER //

CREATE PROCEDURE updatepassword(IN username CHAR(5), IN password VARCHAR(30))

BEGIN

DECLARE i INT;

UPDATE staff\_account

SET staff\_password = password

WHERE Staff\_user\_name = username;

SET i = ROW\_COUNT();

IF i = 0 THEN

SELECT "User not found";

ELSE

SELECT "Password updated";

END IF;

END //

DELIMITER ;

CALL updatepassword("LEC92", "lecture23");

ACCOUNT CREATE AND PRIVILLAGES

DELIMITER //

CREATE PROCEDURE Create\_users\_account(IN usertype VARCHAR(30), IN username CHAR(5), IN password VARCHAR(30))

BEGIN

DECLARE i INT;

DECLARE create\_user\_sql VARCHAR(255);

DECLARE selectusername VARCHAR(255);

DECLARE grant\_select\_sql VARCHAR(255);

DECLARE revoke\_select\_sql VARCHAR(255);

SET create\_user\_sql = CONCAT('CREATE USER ', username, '@''localhost'' IDENTIFIED BY ''', password, ''';');

SET @create\_user\_sql = create\_user\_sql;

PREPARE create\_user\_stmt FROM @create\_user\_sql;

EXECUTE create\_user\_stmt;

DEALLOCATE PREPARE create\_user\_stmt;

SET grant\_select\_sql = CONCAT('GRANT ALL PRIVILEGES ON final.\* TO ', username, '@''localhost'';');

SET revoke\_select\_sql = CONCAT('REVOKE GRANT OPTION ON final.\* FROM ', username, '@''localhost'';');

SET @privilleges=grant\_select\_sql;

PREPARE grant\_select FROM @privilleges;

EXECUTE grant\_select;

DEALLOCATE PREPARE grant\_select;

SET @privilleges=revoke\_select\_sql;

PREPARE revoke\_select FROM @privilleges;

EXECUTE revoke\_select;

DEALLOCATE PREPARE revoke\_select;

SELECT staff\_no INTO selectusername FROM staff WHERE staff\_no = username;

INSERT INTO staff\_account (Staff\_account\_type, Staff\_user\_name, Staff\_password) VALUES (usertype, username, password);

SET i = ROW\_COUNT();

IF i = 0 THEN

SELECT "User not found";

ELSE

SELECT "Successful insert account";

END IF;

END //

DELIMITER ;

call create\_users\_account("Lecture","LEC07","password1");

INSERT STAFF

DELIMITER //

CREATE PROCEDURE staff\_insert(IN staff\_no char(5),IN name varchar(50), IN position varchar(20),IN gender varchar(6),IN home\_address varchar(20), IN street\_address varchar(20), IN country varchar(25))

BEGIN

INSERT INTO staff VALUES(staff\_no,name,position,gender,home\_address,street\_address,country);

END//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE lecture\_in\_bst(IN lecid char(5))

BEGIN

DECLARE lec\_exists INT;

SELECT COUNT(\*) INTO lec\_exists FROM staffdepartment WHERE staff\_no LIKE "LEC\_\_" AND staff\_no=lecid AND Department\_name="BST";

IF lec\_exists = 0 THEN

SELECT "Lecture not found";

ELSE

SELECT "Successful Found details";

SELECT \* FROM staffdepartment WHERE staff\_no = lecid;

END IF;

END//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE lecture\_in\_ET(IN lecid char(5))

BEGIN

DECLARE lec\_exists INT;

SELECT COUNT(\*) INTO lec\_exists FROM staffdepartment WHERE staff\_no LIKE "LEC\_\_" AND staff\_no=lecid AND Department\_name="ET";

IF lec\_exists = 0 THEN

SELECT "Lecture not found";

ELSE

SELECT "Successful Found details";

SELECT \* FROM staffdepartment WHERE staff\_no = lecid;

END IF;

END//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE lecture\_in\_ICT(IN lecid char(5))

BEGIN

DECLARE lec\_exists INT;

SELECT COUNT(\*) INTO lec\_exists FROM staffdepartment WHERE staff\_no LIKE "LEC\_\_" AND staff\_no=lecid AND Department\_name="ict";

IF lec\_exists = 0 THEN

SELECT "Lecture not found";

ELSE

SELECT "Successful Found details";

SELECT \* FROM staffdepartment WHERE staff\_no = lecid;

END IF;

END//

DELIMITER ;

..............................................................................................................

/LECTURE\_DEPARTMENT

DELIMITER //

CREATE PROCEDURE lecture\_department(IN department varchar(10))

BEGIN

DECLARE lec\_exists INT;

SELECT COUNT(\*) INTO lec\_exists FROM staffdepartment WHERE staff\_no LIKE "LEC\_\_" AND Department\_name=department;

IF lec\_exists = 0 THEN

SELECT "Department not found";

ELSE

SELECT "Successful Found details";

SELECT \* FROM staffdepartment WHERE department\_name = department AND staff\_no LIKE "LEC\_\_";

END IF;

END//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE lectureAccount(IN id char(5),IN )

BEGIN

DECLARE lec\_exists INT;

SELECT COUNT(\*) INTO lec\_exists FROM LectureAccount WHERE Staff\_user\_name = lecid;

IF lec\_exists = 0 THEN

SELECT "Lecture not found";

ELSE

SELECT "Successful account details";

SELECT \* FROM LectureAccount WHERE Staff\_user\_name = lecid;

END IF;

END//

DELIMITER ;

**Views**

**TG1010**

//// See all the courses a student is enrolled

CREATE VIEW StudentCourses AS

SELECT s.Reg\_no, s.First\_name, s.Last\_name, c.course\_code, c.name

FROM Student s

INNER JOIN Student\_Course sc ON s.Reg\_no = sc.STUDENT\_Reg\_no

INNER JOIN course c ON sc.COURSE\_code = c.course\_code;

//// Without Student Names

CREATE VIEW CourseDetailsView AS

SELECT student\_course.COURSE\_code, student\_course.STUDENT\_Reg\_no, COURSE.name

FROM student\_course INNER JOIN COURSE ON student\_course.COURSE\_code=COURSE.course\_code;

////// See all the labsheets a student needs to complete

CREATE VIEW StudentLabsheets AS

SELECT s.Reg\_no, s.First\_name, s.Last\_name, ls.labsheet\_id, ls.title, ls.due\_date

FROM Student s

INNER JOIN Student\_Course sc ON s.Reg\_no = sc.STUDENT\_Reg\_no

INNER JOIN labsheet ls ON sc.COURSE\_code = ls.course\_code;

///// See all the past papers available for a student's courses /////

CREATE VIEW StudentPastPapers AS

SELECT s.Reg\_no, s.First\_name, s.Last\_name, pp.past\_paper\_index, pp.paper\_title, pp.year, pp.level, pp.semester

FROM Student s

INNER JOIN Student\_Course sc ON s.Reg\_no = sc.STUDENT\_Reg\_no

INNER JOIN Past\_Papers pp ON sc.COURSE\_code = pp.course\_code;

**TG1030**

01. GPA view

CREATE VIEW StudentGPA AS

SELECT STUDENT\_Reg\_no AS 'Student ID number', SUM(Credits \* GPV\_Value) / SUM(Credits) AS 'GPA'

FROM Course c, result s

WHERE c.course\_code = s.Course\_code

GROUP BY STUDENT\_Reg\_no;

SELECT \* FROM StudentGPA;

02. View to define levels of students

CREATE VIEW Level AS

SELECT Entrance\_year,

CASE

WHEN Entrance\_year = '2018' THEN 'Level 4'

WHEN Entrance\_year = '2019' THEN 'Level 3'

WHEN Entrance\_year = '2020' THEN 'Level 2'

WHEN Entrance\_year = '2021' THEN 'Level 1'

ELSE 0

END AS Level,

COUNT(\*) AS StudentCount

FROM Student

GROUP BY Entrance\_year, Level

ORDER BY Level;

SELECT \* FROM Level;

**TG1038**

CREATE VIEW Result\_Details AS

SELECT R.STUDENT\_Reg\_no , S.First\_name , C.Course\_code, R.Marks,

CASE

WHEN marks >= 90 THEN 'A'

WHEN marks >= 80 THEN 'B'

WHEN marks >= 70 THEN 'C'

WHEN marks >= 60 THEN 'D'

ELSE 'F'

END AS grade,

CASE

WHEN RS.RESULT\_STUDENT\_Reg\_no = R.STUDENT\_Reg\_no THEN 'repeat'

ELSE 'PASS'

END AS repet\_or\_not,

CASE

WHEN RS.Repeat\_count is NULL THEN '0'

ELSE RS.Repeat\_count

END AS repet\_count

FROM STUDENT S

JOIN RESULT R ON S.Reg\_no = R.STUDENT\_Reg\_no

JOIN COURSE C ON C.course\_code = R.Course\_code

LEFT JOIN REPEAT\_STUDENT RS ON RS.RESULT\_STUDENT\_Reg\_no = R.STUDENT\_Reg\_no;

create view Assement\_details AS

SELECT asse.\*,CASE WHEN A.Attendance\_status = 1 OR A.Medical\_status = 1 THEN 'attend' ELSE 'not\_attend' END AS "ATTENDECE\_for\_Assement"

FROM assement asse ,attendance a

WHERE asse.Attendance\_id\_FK=a.Attendance\_id;

CREATE VIEW CA\_ELIGIBILITY AS

SELECT

STUDENT\_Reg\_no,

Course\_code,

COUNT(Assement\_type) AS Num\_Assessments,

SUM(

CASE

WHEN Assement\_type = 'Quize' AND ATTENDECE\_for\_Assement = 'attend' THEN (Score / 4)

WHEN Assement\_type = 'MID\_exam' AND ATTENDECE\_for\_Assement = 'attend' THEN (Score / 2)

ELSE 0

END

) AS Final\_CA\_MARKES,

CASE

WHEN SUM(

CASE

WHEN Assement\_type = 'Quize' AND ATTENDECE\_for\_Assement = 'attend' THEN (Score / 4)

WHEN Assement\_type = 'MID\_exam' AND ATTENDECE\_for\_Assement = 'attend'THEN (Score / 2)

ELSE 0

END

) >= 50 THEN 'eligible'

ELSE 'not eligible'

END AS CA\_Eligibility

FROM Assement\_details

WHERE (Assement\_type = 'MID\_exam' OR Assement\_type = 'Quize')

GROUP BY STUDENT\_Reg\_no, Course\_code;

CREATE VIEW attendence\_detalis AS

SELECT

a.STUDENT\_Reg\_no,

a.Course\_code,

SUM(CASE WHEN a.Attendance\_status=1 OR a.Medical\_status = 1 THEN 1 ELSE 0 END) AS 'Attendance\_Count',

(((SUM(CASE WHEN a.Attendance\_status=1 OR a.Medical\_status = 1 THEN 1 ELSE 0 END)/15)\*100)) AS 'Attendance\_AS\_A\_presentage',

CASE

WHEN (((SUM(CASE WHEN a.Attendance\_status=1 OR a.Medical\_status = 1 THEN 1 ELSE 0 END)/15)\*100)) >= 80 THEN 'eligible'

ELSE 'not eligible'

END AS ATTENDANCE\_Eligibility

FROM attendance a

GROUP BY STUDENT\_Reg\_no,Course\_code;

**TG1056**

Create View LectureAccount As

SELECT

Staff\_user\_name,

Name,

Staff\_account\_type,

Staff\_password from staff,staff\_account Where staff\_account.staff\_user\_name=staff.staff\_no;

CREATE view lecturecountact AS

SELECT

staff\_no,

telephone FROM staff\_telephone;

CREATE VIEW Staffdepartment AS

SELECT

staff.staff\_no,

staff.Name,

staff.Position,

Department\_name

FROM staff

RIGHT JOIN staff\_department ON staff.staff\_no = staff\_department.staff\_no;

* **Problems that occurred during the development of the solution**

Issue with implementing foreign keys between tables.

* **Solutions to overcome the above identified problems**

Had to use queries to create them again not to mention recreating some tables.

* **If you are going to host your backend where are you going to host it and reasons for the selection**

The best place to host the database is cloud environment.

1. Cloud databases gives the flexibility that allows for easy scale up the database as required.
2. As for the famous high security methods that are implemented by cloud providers denotes a big green flag to use cloud hosting.
3. Backup and recovery is big asset of cloud platform that pushes to use cloud systems.
4. Utility tools to version control allows to record progress of the development.
5. Ability to create multiple accounts so multiple users can contribute to the project.

* **If you are going to host your backend in a cloud environment what are the things/changes that you have to do in your backend**

1. Data storage method – From offline storage mechanism to online storage might be similar UIs but different methods of implementations.
2. Data need to be migrated – By using some method needs to migrate offline data to the online database.
3. Consider and research about database security – Online security methods are different from offline data security methods, so in order to achieve security backend security needs to be re-examined.
4. Redundancy check – Ensure that the backend meets data redundancy strategies and data integrity checks
5. Optimize querying – Remove unwanted queries or scripts to improve performance in the cloud environment.

* **Individual contribution to the backend development**

**Software Requirement Specification –** The whole group contributed to making this document.

**Entity Diagram -** L.S.R. VIDANAARACHCHI, A.B.D. ANANDAKUMARA contributed to creating this document.

**Relational Mapping** - B.D.D.DEVENDRA**,** M.A.S.V. KARUNATHILAKAcontributed to creating this document.

**Final Project -** A.B.D. ANANDAKUMARA contributed to creating this document.

1. **TG/2021/1010 - VIDANAARACHCHI L.S.R.**

1. Course table

+-----------------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------------+---------------+------+-----+---------+-------+

| course\_code | char(7) | NO | PRI | NULL | |

| name | varchar(255) | YES | | NULL | |

| course\_hours | decimal(10,0) | YES | | NULL | |

| department\_name | varchar(10) | YES | MUL | NULL | |

| Credits | int(11) | YES | | NULL | |

+-----------------+---------------+------+-----+---------+-------+

This table contains information regarding course unit individual data.

1. Lab sheet table

+-------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+--------------+------+-----+---------+-------+

| labsheet\_id | char(9) | NO | PRI | NULL | |

| title | varchar(255) | YES | | NULL | |

| due\_date | datetime | YES | | NULL | |

| course\_code | char(7) | NO | MUL | NULL | |

+-------------+--------------+------+-----+---------+-------+

Data stored in here contains lab sheet data that are issued with deadlines and this table helps in assessing those lab sheets.

1. Past papers table

+------------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------------+--------------+------+-----+---------+----------------+

| past\_paper\_index | int(3) | NO | PRI | NULL | auto\_increment |

| paper\_title | varchar(255) | NO | | NULL | |

| year | year(4) | NO | | NULL | |

| level | char(6) | NO | | NULL | |

| semester | char(4) | NO | | NULL | |

| course\_code | char(7) | NO | MUL | NULL | |

+------------------+--------------+------+-----+---------+----------------+

An individual table which contains past paper data which are meant to help manage them in a proper manner.

1. **TG/2021/1030 - ANANDAKUMARA A.B.D.**
2. Student table

+----------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------------+-------------+------+-----+---------+-------+

| Reg\_no | char(6) | NO | PRI | NULL | |

| First\_name | varchar(25) | NO | | NULL | |

| Last\_name | varchar(25) | NO | | NULL | |

| Email | varchar(30) | NO | | NULL | |

| Home\_address | varchar(30) | NO | | NULL | |

| Street\_address | varchar(50) | NO | | NULL | |

| Country | varchar(25) | NO | | NULL | |

| Entrance\_year | year(4) | YES | | NULL | |

+----------------+-------------+------+-----+---------+-------+

The cardinal data table in the LMS which is the student data table which contains basic student data which helps to manage their information.

1. Student login table

+------------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+------------+----------+------+-----+---------+-------+

| Reg\_no | char(6) | NO | PRI | NULL | |

| Last\_login | datetime | NO | | NULL | |

+------------+----------+------+-----+---------+-------+

In order to measure time about student login time to the LMS this table helps to store that data.

1. Department table

+----------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+-------+

| Name | varchar(10) | NO | PRI | NULL | |

| Location | varchar(40) | NO | | NULL | |

+----------+-------------+------+-----+---------+-------+

Information here contains the department data which is used to identify basic department related data.

1. Student course table

+----------------+---------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------------+---------+------+-----+---------+-------+

| STUDENT\_Reg\_no | char(6) | NO | PRI | NULL | |

| COURSE\_code | char(7) | NO | PRI | NULL | |

+----------------+---------+------+-----+---------+-------+

In order to normalize and connect student table with course table this table was established.

1. Student department table

+-----------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------------+-------------+------+-----+---------+-------+

| STUDENT\_Reg\_no | char(6) | NO | PRI | NULL | |

| DEPARTMENT\_name | varchar(10) | NO | PRI | NULL | |

+-----------------+-------------+------+-----+---------+-------+

This table achieves connecting student table with the department table and as well as normalizing the initial student table as well.

1. **TG/2021/1038 - DEVENDRA B.D.D.**
2. Assement table

+------------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+------------------+-------------+------+-----+---------+-------+

| Submisson\_id | char(5) | NO | PRI | NULL | |

| Assement\_type | varchar(30) | YES | | NULL | |

| Score | int(11) | YES | | NULL | |

| Due\_date | datetime | YES | | NULL | |

| Submission\_date | datetime | YES | | NULL | |

| Attendance\_id\_FK | char(5) | NO | MUL | NULL | |

| Course\_code | char(7) | NO | MUL | NULL | |

| STUDENT\_Reg\_no | char(6) | NO | MUL | NULL | |

+------------------+-------------+------+-----+---------+-------+

This table helps to store mark based assessment data values in order to identify and manage them.

1. Attendance table

+-------------------+------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------------+------------+------+-----+---------+-------+

| Attendance\_id | char(5) | NO | PRI | NULL | |

| Medical\_status | tinyint(1) | YES | | NULL | |

| Attendance\_status | tinyint(1) | YES | | NULL | |

| Date | date | YES | | NULL | |

| Course\_code | char(7) | NO | MUL | NULL | |

| STUDENT\_Reg\_no | char(6) | NO | MUL | NULL | |

+-------------------+------------+------+-----+---------+-------+

It’s a base requirement to store attendance data about the students mainly the job of maintaining these data to check eligibility for the examination.

1. Student account table

+----------------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------------------+-------------+------+-----+---------+-------+

| STUDENT\_USERNAME | char(6) | NO | PRI | NULL | |

| STUDENT\_Account\_type | varchar(30) | NO | | NULL | |

| STUDENT\_Password | varchar(30) | NO | | NULL | |

+----------------------+-------------+------+-----+---------+-------+

To contain the details of student account meaning whether the student is an internal or an external student and with sensitive data are stored in this table.

1. Repeat student table

+-----------------------+---------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------------------+---------+------+-----+---------+-------+

| RESULT\_STUDENT\_Reg\_no | char(6) | NO | PRI | NULL | |

| RESULT\_Course\_code | char(7) | NO | PRI | NULL | |

| Repeat\_count | int(11) | YES | | NULL | |

+-----------------------+---------+------+-----+---------+-------+

Repeat students are also a possibility within the faculty so there details also must be separately managed and this table achieves it.

1. Result table

+----------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------------+--------------+------+-----+---------+-------+

| STUDENT\_Reg\_no | char(6) | NO | PRI | NULL | |

| Course\_code | char(7) | NO | PRI | NULL | |

| Marks | int(11) | YES | | NULL | |

| GPV\_Value | decimal(3,1) | YES | | NULL | |

+----------------+--------------+------+-----+---------+-------+

Also as a main requirement, a result table is mandatory to store final examination result data.

1. **TG/2021/1056 - KARUNATHILAKA M.A.S.V.**
2. Staff table

+----------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------------+-------------+------+-----+---------+-------+

| Staff\_no | char(5) | NO | PRI | NULL | |

| Name | varchar(25) | YES | | NULL | |

| Position | varchar(20) | YES | | NULL | |

| Gender | varchar(6) | YES | | NULL | |

| Home\_address | varchar(30) | YES | | NULL | |

| Street\_address | varchar(20) | YES | | NULL | |

| Country | varchar(25) | YES | | NULL | |

+----------------+-------------+------+-----+---------+-------+

In order to contain academic staff related data, a separate table should also be maintained.

1. Staff department table

+-----------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------------+-------------+------+-----+---------+-------+

| Staff\_no | char(5) | NO | PRI | NULL | |

| Department\_name | varchar(10) | NO | PRI | NULL | |

+-----------------+-------------+------+-----+---------+-------+

To connect staff table with the department table to establish a connection this table serves its purpose.

1. Staff telephone table

+-----------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------+----------+------+-----+---------+-------+

| Staff\_no | char(5) | NO | PRI | NULL | |

| Telephone | char(11) | NO | PRI | NULL | |

+-----------+----------+------+-----+---------+-------+

This table achieves encapsulation of data by separating staff telephone data from the main staff table.

1. Staff account table

+--------------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------------------+-------------+------+-----+---------+-------+

| Staff\_account\_type | varchar(30) | YES | | NULL | |

| Staff\_user\_name | char(5) | NO | PRI | NULL | |

| Staff\_password | varchar(30) | NO | | NULL | |

+--------------------+-------------+------+-----+---------+-------+

To manage and separate student credential data from staff credential data this table was created.

* **References**

1. Teclms FOT. (n.d.). Teclms –

[https://](https://www.teclms.com/database/mysql/) <http://teclms.ruh.ac.lk/login/index.php>

1. W3Schools. (n.d.). MySQL Tutorial. W3Schools. <https://www.w3schools.com/sql/sql_intro.asp>
2. MySQL. (2023, October 29). Resource Groups. MySQL. <https://dev.mysql.com/doc/refman/8.0/en/resource-groups.html>