



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.

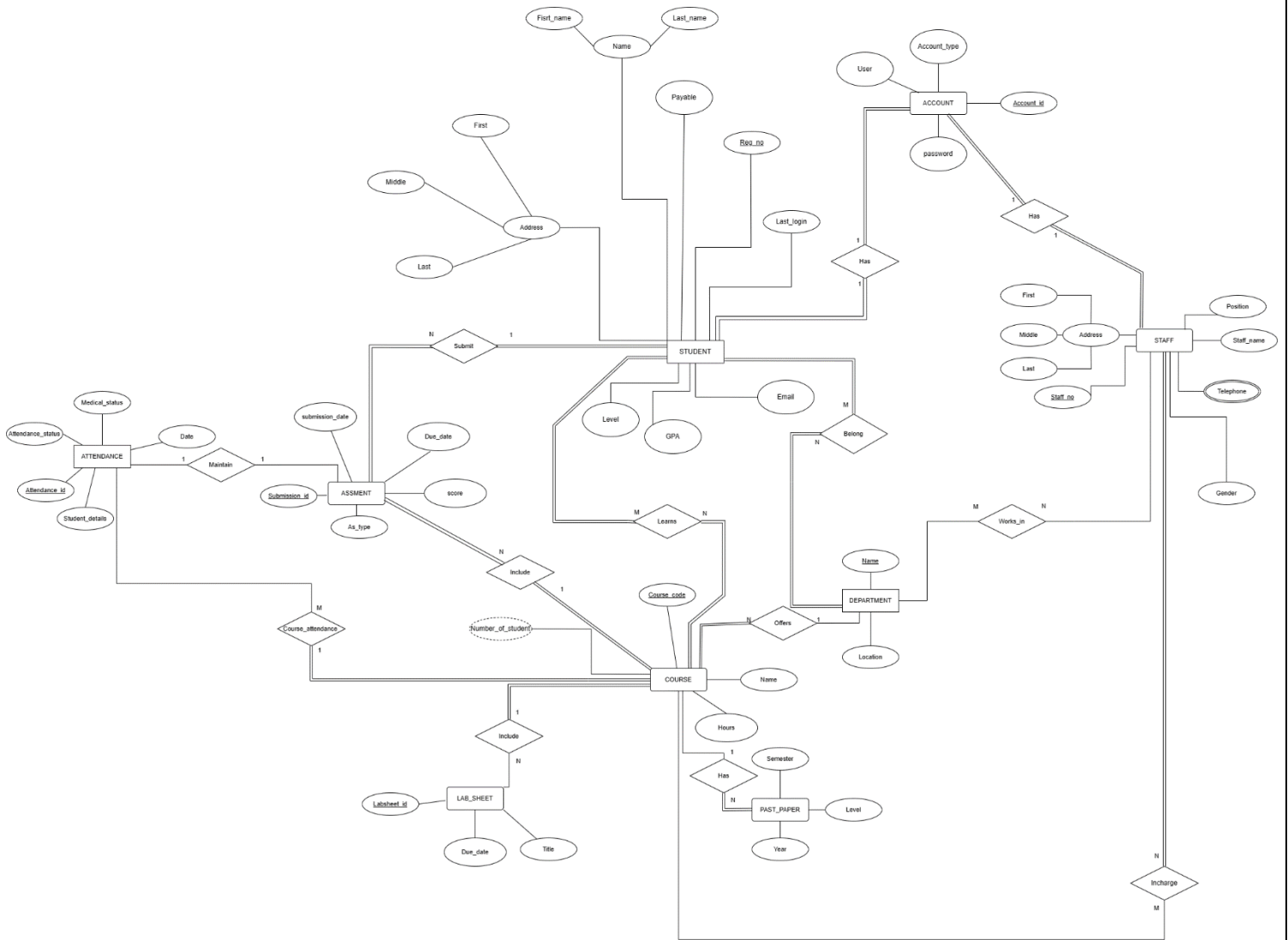
01. Should be able to manage all academic and non-academic personal data separately.
02. Should be able to handle course data and related student data.
03. Should be able to manage academic documents.
04. Should be able to handle attendance data.
05. Should be able to manage and handle different submissions separately.
06. One administrator should be able to manage the entire database.
07. A hierarchy of privileges to manage and access the database is crucial.

- **Brief introduction to the solution**

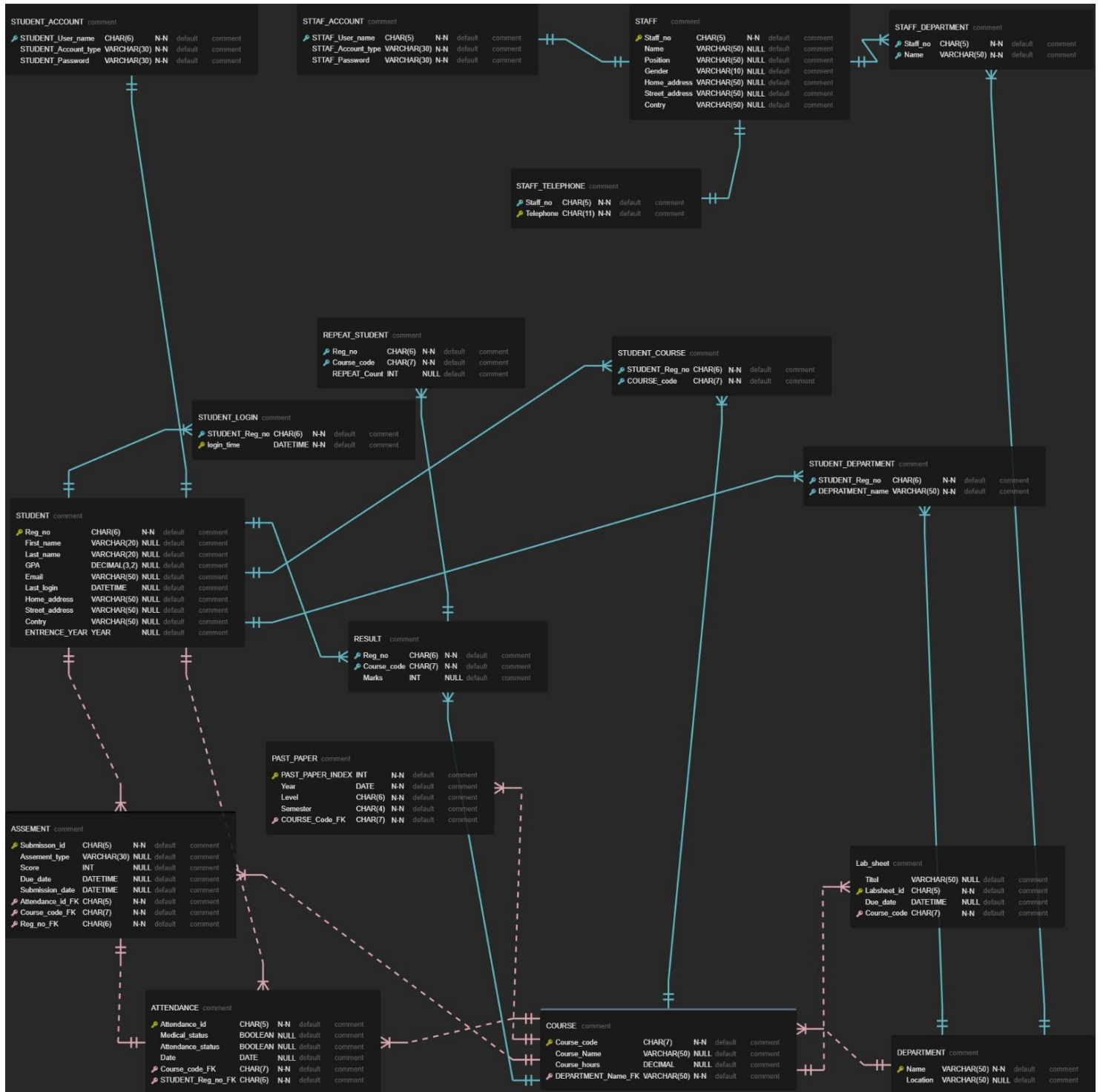
The solution that was developed to meet the requirements mentioned above fulfills those requirements by providing,

01. Separate student related data tables and staff related data tables to manage data separately.
02. Different course tables to separate information for more clarity.
03. Different forms of academic assessments are given different tables with separate tables for results.
04. A past paper table for managing past paper separately from other academic documents.
05. Due to implementing a separate attendance table allows to log all attendance for different courses.
06. Assessment table for marking academic sessions and labsheet table for non-marking academic sessions.
07. 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**

01. **STUDENT** table is 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.
02. **DEPARTMENT** table is also connected to tables such as **STUDENT_DEPARTMENT**, **STAFF_DEPARTMENT**.
03. **STAFF** table holds strings to **STAFF_ACCOUNT**, **STAFF_DEPARTMENT** and **STAFF_TELEPHONE** tables.
04. **COURSE** table connects to **RESULT** table as well as **STUDENT_COURSE** table.
05. **ASSEMENT**, **ATTENDANCE**, and **PAST_PAPER** tables connects to many tables because data within those tables corresponds many data.
06. As for internal table structure regarding data types are used for data to be consistent.

- **Architecture of the solution**

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

- **Tools and technologies that were used**

01. DrawIO online tool – To draw ER diagram.
02. ERD EDITOR Visual Studio Code extention – To draw the relational schema.
03. Microsoft Word – To create the SRS report and the final report.
04. Microsoft Teams – Using video conferencing technology to create the database coordinating with each other.
05. XAMPP with MYSQL – To develop the database this API was used.
06. 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**

01. **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.

02. Dean – This user has the same privileges as the Admin but 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.

03. Lecturer – Lecturer only requires the privilege to concern himself with academic data manipulation so no need to create users or other administrator level functionalities.

04. 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.

05. 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 attendance_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_attendance_of_a_student` (IN
STUDENT_Reg_no char(6))

```
BEGIN
SELECT * FROM attendance_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 @privileges=grant_select_sql;
    PREPARE grant_select FROM @privileges;
    EXECUTE grant_select;
    DEALLOCATE PREPARE grant_select;

    SET @privileges=revoke_select_sql;
    PREPARE revoke_select FROM @privileges;

```

```
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 attendance_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 lecturecontact 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.

01. Cloud databases gives the flexibility that allows for easy scale up the database as required.
02. As for the famous high security methods that are implemented by cloud providers denotes a big green flag to use cloud hosting.
03. Backup and recovery is big asset of cloud platform that pushes to use cloud systems.
04. Utility tools to version control allows to record progress of the development.
05. 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**

01. Data storage method – From offline storage mechanism to online storage might be similar UIs but different methods of implementations.
02. Data need to be migrated – By using some method needs to migrate offline data to the online database.
03. 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.
04. Redundancy check – Ensure that the backend meets data redundancy strategies and data integrity checks
05. 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. KARUNATHILAKA contributed to creating this document.

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

01. TG/2021/1010 - VIDANAARACHCHI L.S.R.

01. 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.

02. 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.

03. 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.

02. TG/2021/1030 - ANANDAKUMARA A.B.D.

01. 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.

02. 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.

03. 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.

04. 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.

05. 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.

03. TG/2021/1038 - DEVENDRA B.D.D.

01. 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.

02. 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.

03. 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.

04. 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.

05. 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.

04. TG/2021/1056 - KARUNATHILAKA M.A.S.V.

01. 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.

02. 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.

03. 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.

04. 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**

01. Teclms FOT. (n.d.). Teclms –
[https:// http://teclms.ruh.ac.lk/login/index.php](http://teclms.ruh.ac.lk/login/index.php)
02. W3Schools. (n.d.). MySQL Tutorial. W3Schools.
https://www.w3schools.com/sql/sql_intro.asp
03. MySQL. (2023, October 29). Resource Groups. MySQL.
<https://dev.mysql.com/doc/refman/8.0/en/resource-groups.html>