MySQL Labs

**MySQL (Day3):**

**insert into students\_courses**

**values**

**(1,4,60,NULL),**

**(2,1,NULL,NULL),**

**(2,4,75,NULL),**

**(3,1,NULL,NULL),**

**(3,2,NULL,NULL),**

**(3,3,75,NULL);**

**Table

Description automatically generated**

**Table

Description automatically generated**

|  |  |
| --- | --- |
| *1* | ***Create function to calculate the number of students who get grade less than 80 in a certain exam (course id will be sent as a parameter)*** |
|  | -- if function already created, delete it  **DROP** **FUNCTION** **IF** **EXISTS** COURSE\_GRADE\_NUMBER;  -- change end query delimiter  **DELIMITER $**  /\*  function to calculate the number of students who get grade less than 80 in a certain exam  [ARGUMENTS]  p\_course\_id => INTEGER => the id of the course  [RETURN]  v\_students\_number => INTEGER => the number of students  \*/  **CREATE** **FUNCTION** COURSE\_GRADE\_NUMBER(p\_course\_id **INTEGER**)  **RETURNS** **INT**(10)  **BEGIN**  -- variable to store the number of students returned from select statement  **DECLARE** v\_students\_number **INT**(11) **DEFAULT** 0;  -- get the number of students from the data and store the result in the variable  **SELECT** **count**(student\_id)  **INTO** v\_students\_number  **FROM** students\_courses  **WHERE** course\_id = p\_course\_id  **AND** grade < 80;  -- return the number of students  **RETURN** v\_students\_number;  END$  **DELIMITER ;**  **SELECT** COURSE\_GRADE\_NUMBER(4); |
| *2* | ***Create stored procedure to display the names of the absence students of a certain courses.(Absent means has no grades)*** |
|  | -- if procedure already created, delete it  **DROP** **PROCEDURE** **IF** **EXISTS** SHOW\_ABSENT\_STUDENTS;  -- change end query delimiter  **DELIMITER $**  /\*  procedure to display the names of the absent students of a certain course  [ARGUMENTS]  p\_course\_id => INTEGER => the id of the course  \*/  **CREATE** **PROCEDURE** SHOW\_ABSENT\_STUDENTS(p\_course\_id **INTEGER**)  **BEGIN**  -- get the names of students and show them  **SELECT** **CONCAT\_WS**(' ', s.first\_name, s.last\_name) **AS** 'Students Names'  **FROM** students s, students\_courses sc  **WHERE** s.student\_id = sc.student\_id  **AND** sc.course\_id = p\_course\_id  **AND** sc.grade **IS** **NULL** ;  END$  **DELIMITER ;**  -- execute the created procedure  **CALL** SHOW\_ABSENT\_STUDENTS(2); |
| *3* | ***Create stored procedure to calculate the average grades for certain course.*** |
|  | -- if procedure already created, delete it  **DROP** **PROCEDURE** **IF** **EXISTS** CALCULATE\_AVERAGE\_GRADE;  -- change end query delimiter  **DELIMITER $**  /\*  procedure to display the calculation of avarage grades for a certain course  [ARGUMENTS]  p\_course\_id => INTEGER => the id of the course  \*/  **CREATE** **PROCEDURE** CALCULATE\_AVERAGE\_GRADE(p\_course\_id **INTEGER**)  **BEGIN**  -- get the names of students and show them  **SELECT** **AVG**(sc.grade) **AS** 'Average Grade'  **FROM** students\_courses sc  **WHERE** sc.course\_id = p\_course\_id;  END$  **DELIMITER ;**  -- execute the created procedure  **CALL** CALCULATE\_AVERAGE\_GRADE(3); |
| *4* | ***Create trigger to keep track the changes(updates) of the grades in the studnets\_courses table***  ***( create changes table with the following fields:***  ***id int primary key ,***  ***user varchar(30),***  ***action varchar(40),***  ***old\_grade int,***  ***new\_grade int,***  ***change\_date date).***  ***Test the trigger by updating grade int the “Students\_courses” table***  ***Confirm that the row is added in the” change\_table”*** |
|  | -- create table to store changes  **CREATE** **TABLE** **IF** **NOT** **EXISTS** grades\_changes\_log (  id **INTEGER** **PRIMARY** **KEY** **AUTO\_INCREMENT**,  **user** **VARCHAR**(30),  **action** **VARCHAR**(40),  old\_grade **INTEGER**,  new\_grade **INTEGER**,  change\_date **DATE**  );  -- create trigger to listen for changes in students\_courses table after any update  **DELIMITER $**  **CREATE** **TRIGGER** grades\_changes  **AFTER** **UPDATE** **ON** students\_courses  -- loop on each row in the result  **FOR** **EACH** **ROW**  **BEGIN**  -- if the grade column is updated  **IF** !(**OLD**.grade = **NEW**.grade)  **THEN**  -- insert the changes in the grades\_changes\_log table  **INSERT** **INTO** grades\_changes\_log (**user**, **action**, old\_grade, new\_grade, change\_date)  **VALUES** (  CURRENT\_USER(),  "Update",  **OLD**.grade,  **NEW**.grade,  **CURDATE**()  );  **END** **IF**;  -- end of for each  **END**;  -- end of trigger  $  **DELIMITER ;**  **UPDATE** students\_courses  **SET** grade = 60  **WHERE** student\_id = 3 **AND** course\_id = 4;  **SELECT** \* **FROM** grades\_changes\_log; |
| *5* | ***Create event to delete the changes tables every 5 minute*** |
|  | **CREATE** EVENT delete\_grade\_changes  **ON** SCHEDULE **EVERY** 5 **MINUTE**  DO  **DELETE** **FROM** grades\_changes\_log; |