**Βάσεις Δεδομένων Project 2022-2023**

1ο Μέλος(με αλφαβητική σειρά επωνύμων):

Όνομα: Αγγουρά Ρουμπίνη – Μαρία

ΑΜ: 1084634

Έτος Φοίτησης: 3ο

2ο Μέλος:

Όνομα: Παυλόπουλος Ιάσονας

ΑΜ: 1084565

Έτος Φοίτησης: 3ο

**Κεφάλαιο 1**

1. Φασδαδς
2. Stored Procedures

3.1.3.1

**Κώδικας:**

DROP PROCEDURE IF EXISTS new\_driver;

DELIMITER $

CREATE PROCEDURE new\_driver(drvAT CHAR(10), first\_name VARCHAR(20), last\_name VARCHAR(20), salary FLOAT(7,2), license ENUM('A', 'B', 'C', 'D'), dr\_route ENUM('LOCAL', 'ABROAD'), experience TINYINT(4))

BEGIN

    DECLARE not\_found INT;

    DECLARE min\_br TINYINT;

    DECLARE temp TINYINT;

    DECLARE branches TINYINT;

    DECLARE first\_br TINYINT;

    DECLARE drvcursor CURSOR FOR

    SELECT br\_code FROM branch;

    DECLARE CONTINUE HANDLER FOR NOT FOUND

    SET not\_found=1;

    SET min\_br=1;

    SELECT COUNT(\*) INTO first\_br FROM driver

    INNER JOIN worker ON drv\_AT=wrk\_AT

    WHERE wrk\_br\_code=1;

    SET not\_found=0;

    OPEN drvcursor;

    REPEAT

        FETCH drvcursor INTO branches;

        IF(not\_found=0)

        THEN

            SELECT COUNT(\*) INTO temp FROM driver

            INNER JOIN worker ON drv\_AT=wrk\_AT

            WHERE wrk\_br\_code=branches;

            IF(temp<first\_br)

            THEN

                SELECT branches INTO min\_br;

                SET first\_br=temp;

            END IF;

        END IF;

    UNTIL(not\_found=1)

    END REPEAT;

    INSERT INTO worker VALUES

    (drvAT, first\_name, last\_name, salary, min\_br);

    INSERT INTO driver VALUES

    (drvAT, license, dr\_route, experience);

END$

DELIMITER ;

**Η παραδοχή μας:** Σε αυτό το stored procedure, ο στόχος είναι η προσθήκη ενός νέου οδηγού(driver) στο υποκατάστημα(branch) που έχει τους λιγότερους.

Παραδοχή που ακολουθήσαμε: Αν υπάρχει παραπάνω από ένα υποκατάστημα με τον μικρότερο αριθμό οδηγών, ο καινούριος οδηγός προστίθεται αυτόματα σε αυτό με το μικρότερο br\_code.

Επίσης θεωρούμε πως από την στιγμή που ζητείται να δοθούν ως ορίσματα, χαρακτηριστικά που ανήκουν στον πίνακα worker, ότι αυτός ο driver δεν έχει εισαχθεί ακόμα ούτε ως worker. Για αυτόν τον λόγο στο τέλος τον εισάγουμε και στον πίνακα worker και στον πίνακα driver.

**Επεξήγηση Κώδικα:** Για την επίτευξη του παραπάνω ζητήματος, δημιουργήσαμε έναν cursor ο οποίος κρατάει τα br\_code από όλα τα branches.

Έπειτα με ένα Select Into Statement, βάλαμε στην τοπική μεταβλητή first\_br, τον αριθμό των drivers που υπάρχουν στο υποκατάστημα με br\_code=1. Αυτό θα το χρησιμοποιήσουμε για να συγκρίνουμε αργότερα το πλήθος των οδηγών των υπόλοιπων υποκαταστημάτων και να κρατήσουμε αυτό με το μικρότερο.

Με την χρήση του cursor λοιπόν περνάμε από όλα τα υποκαταστήματα και μετράμε το πλήθος των οδηγών που ανήκουν σε καθένα από αυτά (count(\*)). Έπειτα, αν το πλήθος αυτό είναι μικρότερο από το “default” που ορίσαμε, δηλαδή αυτό του υποκαταστήματος με br\_code=1, τότε αποθηκεύουμε στην μεταβλητή min\_br το br\_code αυτού του υποκαταστήματος. (Η min\_br έχει αρχικοποιηθεί νωρίτερα με την τιμή 1, ώστε αν δεν βρεθεί branch με μικρότερο αριθμό οδηγών από το “default” να κρατηθεί το 1)

Όταν συγκρίνουμε όλα τα branches, βγαίνουμε από το loop του cursor, και κάνουμε add και στον worker και στον driver τα στοιχεία του driver που δόθηκαν σαν ορίσματα στο stored procedure.

Table

Description automatically generated**Παράδειγμα Υλοποίησης:**

Πρώτα ας παρατηρήσουμε το πλήθος των οδηγών σε κάθε υποκατάστημα πριν καλέσουμε την Stored Procedure, με την χρήση ενός Select Statement:

Text

Description automatically generated

Έπειτα ας δούμε τους πίνακες worker και driver:

Text

Description automatically generated(O πίνακας worker είναι πολύ μεγάλος για να χωρέσει στην αναφορά όποτε θα κάνουμε ένα Select με count για να δούμε πόσοι workers υπάρχουν πριν καλέσουμε την συνάρτηση):

Table

Description automatically generated

Με βάση τα παραπάνω στοιχεία και τις παραδοχές που έχουμε κάνει, παρατηρούμε πως ο καινούριος driver που θα προσθέσουμε θα πρέπει να εισαχθεί στο branch με br\_code = 1.

Καλούμε την Stored Procedure:

Table

Description automatically generatedTable

Description automatically generatedΑς συγκρίνουμε τους πίνακες τώρα με τους αντίστοιχους πίνακες πριν:

Graphical user interface, text, application

Description automatically generated

Παρατηρούμε λοιπόν απευθείας ότι το branch με br\_code=1, έχει έναν καινούριο driver.

Από τον πίνακα driver και worker παρατηρούμε πως έχει προστεθεί όντως ο σωστός driver με τα σωστά στοιχεία. Επίσης παρατηρούμε πως όντως αυτός ο worker δεν υπήρχε πριν καλέσουμε την Stored Procedure καθώς βλέπουμε ότι ο αριθμός των workers αυξήθηκε από 43 σε 44.

3.1.3.2

**Κώδικας:**

DROP PROCEDURE IF EXISTS date\_check;

DELIMITER $

CREATE PROCEDURE date\_check(br\_code INT, date1 DATE, date2 DATE)

BEGIN

DECLARE dates DATETIME;

DECLARE tripid INT;

DECLARE reservations INT;

DECLARE max\_seats tinyINT;

DECLARE seatdiff tinyINT;

DECLARE not\_found INT;

DECLARE tridcursor CURSOR FOR

SELECT tr\_id FROM trip WHERE tr\_br\_code=br\_code;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET not\_found=1;

SET not\_found=0;

DROP TABLE IF EXISTS new;

CREATE TABLE new(

id INT AUTO\_INCREMENT NOT NULL,

trcost FLOAT(7,2),

maxseats\_ INT,

reservations\_ INT,

seatdiff\_ TINYINT,

drv\_name\_ CHAR(20),

drv\_lname\_ CHAR(20),

gui\_name\_ CHAR(20),

gui\_lname\_ CHAR(20),

tr\_dep\_ DATETIME,

tr\_ret\_ DATETIME,

PRIMARY KEY(id)

);

OPEN tridcursor;

REPEAT

FETCH tridcursor INTO tripid;

IF(not\_found=0)

THEN

SELECT tr\_departure INTO dates FROM trip

WHERE tr\_id=tripid;

IF (DATEDIFF(dates, date1)>0 && DATEDIFF(dates, date2)<0)

THEN

SELECT COUNT(\*) INTO reservations FROM reservation

WHERE tripid=res\_tr\_id;

SELECT tr\_maxseats INTO max\_seats FROM trip

WHERE tr\_id=tripid;

SET seatdiff=max\_seats-reservations;

INSERT INTO new

SELECT null, tr\_cost, tr\_maxseats, reservations,

seatdiff, a.wrk\_name, a.wrk\_lname, b.wrk\_name,

b.wrk\_lname, tr\_departure, tr\_return

FROM trip

INNER JOIN worker AS a ON tr\_drv\_AT=a.wrk\_AT

INNER JOIN worker AS b ON tr\_gui\_AT=b.wrk\_AT

WHERE tr\_id=tripid;

END IF;

END IF;

UNTIL(not\_found=1)

END REPEAT;

SELECT \* FROM new;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:** Αρχικά δημιουργούμε έναν cursor, ο οποίος κρατάει τα tr\_ids των trips τα οποία διοργανώνονται από το υποκατάστημα που έχει δοθεί ως όρισμα. Δημιουργούμε έναν πίνακα στον οποίο θα αποθηκεύσουμε όλες τις τιμές που θα πάρουμε μέσα από το repeat του cursor ώστε να μπορέσουμε να τις τυπώσουμε όλες στο τέλος σε έναν πίνακα. Έπειτα με repeat του cursor, περνάμε για καθένα από αυτά τα ταξίδια την τιμή tr\_departure τους, στην μεταβλητή dates. Στην συνέχεια με την χρήση ενός If, ελέγχουμε αν αυτή η ημερομηνία βρίσκεται ανάμεσα στις δύο που δόθηκαν στην αρχή. Αν βρίσκεται, μετράμε πόσα reservations έχουν γίνει σε αυτό το ταξίδι και αποθηκεύουμε την τιμή αυτή στην μεταβλητή reservations. Επιπρόσθετα, αποθηκεύουμε στην μεταβλητή max\_seats, τον μέγιστο αριθμό θέσεων που υποστηρίζει το ταξίδι και πραγματοποιούμε και μία αφαίρεση για να υπολογίσουμε πόσες από αυτές τις θέσεις είναι ακόμα κενές. Τέλος περνάμε όλες τις πληροφορίες στον πίνακα new. Βγαίνουμε από το repeat του cursor και κάνουμε έναν SELECT \* στον πίνακα new για να τυπωθούν τα επιθυμητά αποτελέσματα.

**Παράδειγμα Υλοποίησης:**

Αρχικά ας δούμε τον πίνακα trip για να δούμε αναλυτικά όλα τα ταξίδια και τις ημερομηνίες αναχώρησης του καθενός από αυτά:

A picture containing table

Description automatically generated

Graphical user interface, application, Teams

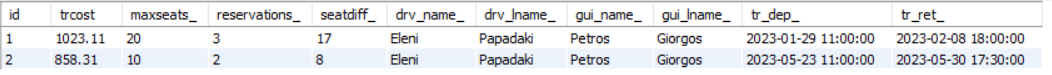
Description automatically generated

Καλούμε την συνάρτηση:



Από τον παραπάνω πίνακα παρατηρούμε πως μόνο ένα ταξίδι καλύπτει τις προϋποθέσεις των ορισμάτων της Stored Procedure, και αυτό είναι και αυτό που εμφανίζει και η Stored Procedure.

Καλούμε ξανά την Stored Procedure με άλλα ορίσματα για να σιγουρευτούμε ότι δουλεύει για για περισσότερα από 1 ταξίδια:



Παρατηρούμε πως ακόμα και για αυτήν την περίπτωση τα αποτελέσματα είναι τα σωστά.

3.1.3.3

**Κώδικας:**

DROP PROCEDURE IF EXISTS admin\_check;

DELIMITER $

CREATE PROCEDURE admin\_check(work\_name CHAR(20), work\_lname CHAR(20))

BEGIN

    DECLARE detail ENUM('LOGISTICS', 'ADMINISTRATIVE', 'ACCOUNTING');

    DECLARE admid CHAR(10);

    SELECT adm\_type, adm\_AT INTO detail, admid FROM admin

    INNER JOIN worker ON adm\_AT=wrk\_AT

    WHERE work\_name=wrk\_name AND work\_lname=wrk\_lname;

    /\*SELECT detail;\*/

    IF (detail!='ADMINISTRATIVE' AND detail IS NOT NULL)

    THEN

        DELETE FROM admin WHERE adm\_AT=admid;

        DELETE FROM worker WHERE wrk\_AT=admid;

    ELSEIF (detail IS NULL)

    THEN

        SELECT('This Worker is not an Admin!');

    ELSE

        SELECT('You cannot Delete this User. This User is an Administrative.');

    END IF;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:** Αρχικά δημιουργούμε δύο μεταβλητές, την detail και την admid και περνάμε σε αυτές τις τιμές adm\_type και adm\_AT αντίστοιχα (του πίνακα admin) για τις περιπτώσεις που το όνομα και το επώνυμο στον πίνακα admin είναι ίδια με αυτά που δόθηκαν σαν όρισμα στην Stored Procedure.

Έπειτα δημιουργούμε ένα If, και ελέγχουμε αν το type του worker που δόθηκε δεν είναι null και ταυτόχρονα δεν είναι και ‘ADMINISTRATIVE’. Αυτό το κάνουμε για να ελέγξουμε αν ο εργάτης που δόθηκε είναι admin αλλά και ταυτόχρονα αν ανήκει στις κατηγορίες των Admin που μπορούμε να διαγράψουμε ή όχι. Αν λοιπόν δεν είναι null και δεν ανήκει στην κατηγορία ‘ADMINISTRATIVE’ τότε κάνουμε αυτόν τον εργάτης delete και από τον πίνακα worker αλλά και από τον πίνακα admin. Επιπρόσθετα, έχουμε και ένα IF ELSE, το οποίο ελέγχει αν ο εργάτης που δόθηκε σαν όρισμα είναι admin, αν δεν είναι τυπώνει κατάλληλο μήνυμα. Τέλος έχουμε την περίπτωση που ο worker είναι admin και ταυτόχρονα ανήκει στην κατηγορία ‘ADMINISTRATIVE’, τότε τυπώνουμε μήνυμα που λέει ότι δεν μπορεί να διαγραφεί αυτός ο Worker καθώς ανήκει στην παραπάνω κατηγορία.

**Παράδειγμα Υλοποίησης:**

Graphical user interface, text

Description automatically generatedΠρώτα ας κάνουμε Print όλους τους Admin, πριν καλέσουμε την stored procedure:

Έπειτα θα κάνουμε 3 διαφορετικές δοκιμές.

Η πρώτη θα καλεί την Stored Procedure με ορίσματα που ανήκουν σε Admin τύπου ADMINISTRATIVE. Η δεύτερη θα καλεί την Stored Procedure με ορίσματα Admin που δεν είναι τύπου ADMINISTRATIVE. Και η τρίτη θα καλεί την Stored Procedure με ορίσματα που δεν ανήκουν σε κάποιον Admin.

1ο Call:



2ο Call:



Graphical user interface, text, application

Description automatically generatedΠαρατηρούμε ότι ο Admin με το όνομα Anastasia Petropoulou, δεν υπάρχει πια στην λίστα με τους Admin.

3ο Call:



Παρατηρούμε πως ο εργάτης Petros Giorgos, δεν έχει εμφανιστεί στον πιο πάνω πίνακα, δηλαδή δεν είναι Admin, αρά τυπώνεται σωστά το παραπάνω μήνυμα και δεν επιτρέπει την διαγραφή του.

3.1.3.4 α)

**Κώδικας:**

DROP PROCEDURE IF EXISTS prepaid;

DELIMITER $

CREATE PROCEDURE prepaid(cost1 FLOAT(7,2), cost2 FLOAT(7,2))

BEGIN

    DECLARE costs FLOAT(7,2);

    DECLARE res\_id INT(11);

    DECLARE not\_found INT;

    /\*DECLARE min FLOAT(7,2);

    declare max FLOAT(7,2);\*/

    DECLARE precursor CURSOR FOR

    SELECT res\_of\_id FROM reservation\_offers;

    DECLARE CONTINUE HANDLER FOR NOT FOUND

    SET not\_found=1;

    SET not\_found=0;

    DROP TABLE IF EXISTS final\_paid;

    CREATE TABLE final\_paid(

        last\_name CHAR(20),

        first\_name CHAR(20),

        id INT(11),

        PRIMARY KEY(id)

    );

    OPEN precursor;

    REPEAT

        FETCH precursor INTO res\_id;

        IF(not\_found=0)

        THEN

            SELECT adv\_pay INTO costs FROM reservation\_offers

            WHERE res\_of\_id=res\_id;

            /\*SET min=cost1-costs;

            SET max=costs-cost2; \*/

                INSERT INTO final\_paid

                SELECT last\_name, first\_name, res\_of\_id FROM reservation\_offers

                USE INDEX (reservation\_offers\_adv\_pay\_idx)

                WHERE res\_of\_id=res\_id && adv\_pay >=cost1 && adv\_pay <= cost2;

        END IF;

    UNTIL(not\_found=1)

    END REPEAT;

    SELECT last\_name AS Last\_Name, first\_name AS First\_Name FROM final\_paid;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:** Αρχικά δημιουργούμε έναν cursor στον οποίο περνάμε όλα τα res\_od\_id από τον πίνακα reservation\_offers. Έπειτα δημιουργούμε ένα καινούριο table, το final\_paid στο οποίο θα περάσουμε στην συνέχεια όλα τα δεδομένα που θέλουμε να τυπώσουμε, για να τυπωθούν όλα μαζί σε έναν πίνακα. Έπειτα ξεκινάει το repeat του cursor. Σε αυτό περνάμε για κάθε τιμή του cursor, το ποσό που έχουν δώσει ως προκαταβολή, στην μεταβλητή costs. Έπειτα εισάγουμε στον πίνακα final\_paid τα δεδομένα που θα τυπώσουμε. Σε αυτό το κομμάτι χρησιμοποιούμε και τον index, παρόλο που δεν κάνει καμία διαφορά στον χρόνο που χρειάζεται για να εμφανιστούν τα αποτελέσματα. Τέλος βγαίνουμε από το repeat και τυπώνουμε όλο τον πίνακα final\_paid

**Παράδειγμα Υλοποίησης:**

Εδώ καθώς το πλήθος των εγγραφών είναι απίστευτα μεγάλο, δεν γίνεται να έχουμε παράδειγμα πριν την υλοποίηση του stored procedure. Οπότε θα παραθέσουμε μόνο τον πίνακα που τυπώνεται αφού καλέσουμε την συνάρτηση (και αυτόν όχι ολόκληρο)



A picture containing table

Description automatically generated

3.1.3.4 β)

**Κώδικας:**

DROP PROCEDURE IF EXISTS check\_offers;

DELIMITER $

CREATE PROCEDURE check\_offers(lname CHAR(20))

BEGIN

    DECLARE res\_id INT(11);

    DECLARE not\_found INT;

    DECLARE ofcursor CURSOR FOR

    SELECT res\_of\_id FROM reservation\_offers WHERE last\_name=lname;

    DECLARE CONTINUE HANDLER FOR NOT FOUND

    SET not\_found=1;

    SET not\_found=0;

    DROP TABLE IF EXISTS new\_res\_off;

    CREATE TABLE new\_res\_off(

        offid TINYINT(4),

        resid INT(11),

        first\_name CHAR(20),

        last\_name CHAR(20),

        PRIMARY KEY(resid)

    );

    OPEN ofcursor;

    REPEAT

        FETCH ofcursor INTO res\_id;

        IF(not\_found=0)

        THEN

            INSERT INTO new\_res\_off

            SELECT off\_id, res\_of\_id, first\_name, last\_name FROM reservation\_offers

            WHERE last\_name=lname AND res\_of\_id=res\_id;

        END IF;

    UNTIL(not\_found=1)

    END REPEAT;

    SELECT offid AS Offer\_ID, first\_name AS First\_Name, last\_name AS Last\_Name FROM new\_res\_off

    ORDER BY offid;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:** Αρχικά ξεκινάμε δημιουργώντας έναν cursor ο οποίος κρατάει όλα τα res\_of\_it από τον πίνακα reservation\_offers, των οποίων τα lname είναι ίδιο με το lastname που δόθηκε σαν όρισμα στο Stored Procedure. Έπειτα όπως και στο προηγούμενο παράδειγμα δημιουργούμε έναν πίνακα στον οποίο θα αποθηκεύσουμε αργότερα όλες τις τιμές που θα θέλουμε να εκτυπώσουμε, για να μπορέσουμε να τις εμφανίσουμε όλες στον ίδιο πίνακα. Ξεκινάμε το repeat του cursor, και για κάθε διαφορετική τιμή του περνάμε μέσα στον νέο μας πίνακα, τα στοιχεία που ζητούνται από την εκφώνηση. Όταν περάσουμε από όλες τις τιμές του cursor, βγαίνουμε από το repeat και κάνουμε έναν Select για τον πίνακα που δημιουργήσαμε. Επίσης ορίζουμε να εμφανιστούν με βάση το offid τους.

**Παράδειγμα Υλοποίησης:**

Όπως και στο προηγούμενο παράδειγμα, καθώς το πλήθος των εγγραφών είναι απίστευτα μεγάλο, δεν γίνεται να έχουμε παράδειγμα πριν την υλοποίηση του stored procedure. Οπότε θα παραθέσουμε μόνο τον πίνακα που τυπώνεται αφού καλέσουμε την συνάρτηση (και αυτόν όχι ολόκληρο):



Table

Description automatically generatedΌπως βλέπουμε τυπώνονται με βάση το offid όπως ακριβώς ζητείται στην εκφώνηση. (Αυτή δεν είναι η αρχή της εκτύπωσης του πίνακα αλλά το κομμάτι στο οποίο βλέπουμε ότι περνάει από τις εγγραφές με offid=1 σε αυτές με offid=2.

Έξτρα Stored Procedures

* **Κώδικας:**

DROP PROCEDURE IF EXISTS seats\_trip;

DELIMITER $

CREATE PROCEDURE seats\_trip(IN res\_id INT, OUT res TINYINT)

BEGIN

DECLARE currentSeats TINYINT;

DECLARE rsvid INT;

DECLARE not\_found INT;

DECLARE max TINYINT;

DECLARE tripcursor CURSOR FOR

SELECT res\_tr\_id FROM reservation WHERE res\_tr\_id=res\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET not\_found=1;

SELECT tr\_maxseats INTO max FROM trip WHERE tr\_id=res\_id;

SET not\_found=0;

OPEN tripcursor;

REPEAT

FETCH tripcursor INTO rsvid;

IF(not\_found=0)

THEN

SELECT COUNT(\*) INTO currentSeats FROM reservation WHERE res\_tr\_id=rsvid;

END IF;

UNTIL(not\_found=1)

END REPEAT;

IF (currentSeats < max OR currentSeats IS NULL)

THEN SET res=1;

ELSE SET res=0;

END IF;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:** Για την καλύτερη λειτουργία του προγράμματος κρατήσαμε στον κώδικας μας το stored procedure που είχε ζητηθεί να δημιουργήσουμε ως 5η εργαστηριακή άσκηση. Αυτό το stored procedure έχει ως στόχο, πριν από κάθε insert στον πίνακα reservation, να ελέγχει αν υπάρχουν διαθέσιμες θέσεις για το ταξίδι στο οποίο αναφέρεται το insert. Έπειτα με ένα trigger που θα αναλύσουμε στην συνέχεια, απαγορεύει το insert σε περίπτωση που δεν υπάρχουν άλλες διαθέσιμες θέσεις.

Για να υλοποιήσουμε αυτό το Stored Procedure, ξεκινάμε δημιουργώντας έναν cursor, που ονομάζουμε tripcursor. Σε αυτόν περνάμε τα res\_tr\_id των reservation που έχουν την ίδια τιμή με το δοσμένο res\_id.

Στην συνέχεια, περνάμε στην μεταβλητή max, το tr\_maxseats από τον πίνακα trip.

Έπειτα ξεκινάμε το repeat του tripcursor και με count(\*) μετράμε όλες τις κρατήσεις που έχουν γίνει για το συγκεκριμένο trip. Τέλος βγαίνουμε από το repeat και με ένα If Statement, ελέγχουμε αν υπάρχουν διαθέσιμες θέσεις ή όχι. Αν υπάρχουν, θέτουμε την μεταβλητή res=1, ενώ αν δεν υπάρχουν, θέτουμε res=0. Αυτά θα μας χρειαστούν στην συνέχεια στον trigger.

**Παράδειγμα Υλοποίησης:**

Καθώς αυτό το Stored Procedure συνδέεται άμεσα με ένα trigger δεν μπορούμε να δούμε την πραγματική του λειτουργία χωρίς αυτό. Παρόλαυτα, μπορούμε να δούμε αν η μεταβλητή res, παίρνει στο τέλος την τιμή που πρέπει να πάρει.

Θα καλέσουμε λοιπόν αυτό το Stored Procedure δύο φορές, μία για ένα ταξίδι που δεν έχει άλλες διαθέσιμες θέσεις, και μία για ένα που έχει.

Για τις ανάγκες του παραδείγματος θα πρέπει να προσθέσουμε στον κώδικα μια έξτρα εντολή select για να εκτυπώσουμε το res στην οθόνη μας.

Ας ξεκινήσουμε με αυτό που δεν έχει άλλες διαθέσιμες θέσεις, το res του θα πρέπει να είναι 0.



Και τώρα ας δούμε για ένα trip που έχει ακόμα διαθέσιμες θέσεις.



1. Triggers

3.1.4.1.

**Κώδικας:**

DROP TRIGGER IF EXISTS logUTrip;

DELIMITER $

CREATE TRIGGER logUTrip AFTER UPDATE ON trip

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Update', 'trip', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logDTrip;

DELIMITER $

CREATE TRIGGER logDTrip AFTER DELETE ON trip

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Delete', 'trip', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logITrip;

DELIMITER $

CREATE TRIGGER logITrip AFTER INSERT ON trip

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Insert', 'trip', last\_name, curr\_date);

END$

DELIMITER ;

/\*UPDATE trip SET tr\_departure='2023-01-30 11:00:00' WHERE tr\_id=21;

DELETE FROM trip WHERE tr\_id=21;

INSERT INTO trip VALUES

(null, '2023-11-04 10:30:00', '2023-11-15 18:00:00', 23, 219.99, 4, 'FG91992776', 'AT53041686');

SELECT \* FROM logging;\*/

DROP TRIGGER IF EXISTS logURes;

DELIMITER $

CREATE TRIGGER logURes AFTER UPDATE ON reservation

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Update', 'reservation', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logDRes;

DELIMITER $

CREATE TRIGGER logDRes AFTER DELETE ON reservation

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Delete', 'reservation', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logIRes;

DELIMITER $

CREATE TRIGGER logIRes AFTER INSERT ON reservation

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Insert', 'reservation', last\_name, curr\_date);

END$

DELIMITER ;

/\*UPDATE reservation SET res\_isadult='MINOR' WHERE res\_tr\_id=21;

DELETE FROM reservation WHERE res\_tr\_id=21;

INSERT INTO reservation VALUES

(21, 21, 'Sam', 'Lerroy', 'ADULT');

SELECT \* FROM logging;\*/

DROP TRIGGER IF EXISTS logUEv;

DELIMITER $

CREATE TRIGGER logUEv AFTER UPDATE ON event

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Update', 'event', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logDEv;

DELIMITER $

CREATE TRIGGER logDEv AFTER DELETE ON event

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Delete', 'event', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logIEv;

DELIMITER $

CREATE TRIGGER logIEv AFTER INSERT ON event

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Insert', 'event', last\_name, curr\_date);

END$

DELIMITER ;

/\*UPDATE event SET ev\_descr='Visiting The Colloseum.' WHERE ev\_tr\_id=21;

DELETE FROM event WHERE ev\_tr\_id=21;

INSERT INTO event VALUES

(21, '2023-11-04 10:30:00', '2023-11-15 18:00:00', 'Visiting The Louvre Museum.');

SELECT \* FROM logging;\*/

DROP TRIGGER IF EXISTS logUTravel;

DELIMITER $

CREATE TRIGGER logUTravel AFTER UPDATE ON travel\_to

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Update', 'travel\_to', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logDTravel;

DELIMITER $

CREATE TRIGGER logDTravel AFTER DELETE ON travel\_to

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Delete', 'travel\_to', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logITravel;

DELIMITER $

CREATE TRIGGER logITravel AFTER INSERT ON travel\_to

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Insert', 'travel\_to', last\_name, curr\_date);

END$

DELIMITER ;

/\*UPDATE travel\_to SET to\_dst\_id='12' WHERE to\_tr\_id=21;

DELETE FROM travel\_to WHERE to\_tr\_id=21;

INSERT INTO travel\_to VALUES

(21, 16, '2023-11-05 10:30:00', '2023-11-05 18:00:00');

SELECT \* FROM logging;\*/

DROP TRIGGER IF EXISTS logUDest;

DELIMITER $

CREATE TRIGGER logUDest AFTER UPDATE ON destination

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Update', 'destination', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logDDest;

DELIMITER $

CREATE TRIGGER logDDest AFTER DELETE ON destination

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Delete', 'destination', last\_name, curr\_date);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS logIDest;

DELIMITER $

CREATE TRIGGER logIDest AFTER INSERT ON destination

FOR EACH ROW

BEGIN

    DECLARE curr\_date DATETIME;

    DECLARE last\_name CHAR(20);

    SELECT lname INTO last\_name FROM user;

    SELECT now() INTO curr\_date;

    INSERT INTO logging VALUES

    (null, 'Insert', 'destination', last\_name, curr\_date);

END$

DELIMITER ;

/\*UPDATE destination SET dst\_name='Venice' WHERE dst\_id=16;

DELETE FROM destination WHERE dst\_id=16;

SELECT \* FROM destination;

INSERT INTO destination VALUES

(null, 'Rome', 'There’s no place like Rome and of course when in Rome, do as the Romans do.', 'ABROAD', 'Italian', 17);

SELECT \* FROM logging;\*/

**Επεξήγηση Κώδικα:** Για να υλοποιήσουμε το ζητούμενο του ερωτήματος, δημιουργήσαμε 15 triggers. Δηλαδή 3 triggers για κάθε έναν από τους πίνακες που αναφέρονται στο ερώτημα. Πιο συγκεκριμένα, για κάθε πίνακα έχουμε ένα After Insert, ένα After Delete και ένα After Update trigger.

Στόχος των trigger είναι μετά από κάθε κίνηση σε οποιονδήποτε από τους ζητούμενους πίνακες, να ενημερώνεται ο πίνακας logging για την κίνηση αυτή.

Σε κάθε trigger από τους παραπάνω, παίρνουμε με δύο μεταβλητές τις τιμές curr\_date και last\_name, και έπειτα χρησιμοποιώντας τες, κάνουμε INSERT στον πίνακα logging με τις απαραίτητες πληροφορίες. Το last\_name που κρατάμε είναι το επώνυμο του it manager που έκανε κάθε φορά την αλλαγή και το παίρνουμε από έναν πίνακα user που έχουμε δημιουργήσει και κρατάμε κάθε φορά το επώνυμο του user που είναι συνδεδεμένος κάθε φορά στην εφαρμογή.

**Παράδειγμα Υλοποίησης:**

3.1.4.2

**Κώδικας:**

DROP TRIGGER IF EXISTS no\_change;

DELIMITER $

CREATE TRIGGER no\_change BEFORE UPDATE ON trip

FOR EACH ROW

BEGIN

    DECLARE temp INT(11);

    IF (NEW.tr\_departure<>OLD.tr\_departure OR NEW.tr\_return<>OLD.tr\_return OR NEW.tr\_cost<>OLD.tr\_cost)

    THEN

        SELECT res\_tr\_id INTO temp FROM reservation

        WHERE res\_tr\_id = OLD.tr\_id LIMIT 0,1;

        IF (temp IS NOT NULL)

        THEN

            SIGNAL SQLSTATE '45000'

            SET MESSAGE\_TEXT = 'You cannot update this trip because there is alredy a registration.';

        END IF;

    END IF;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:**

Το συγκεκριμένο trigger ενεργοποιείται πριν κάθε Update στον πίνακα trip. Δουλειά του είναι να ελέγχει αν έχουν γίνει κρατήσεις σε αυτό το trip, αν έχουν γίνει και το Update που πάνε να κάνουμε είναι πάνω σε ημερομηνία ή στο κόστος, τότε δεν μας αφήνει να το κάνουμε.

Για να υλοποιήσουμε το ζητούμενο, ελέγχουμε ως παράμετρο στο if, αν κάποια από τις μεταβλητές που δεν θέλουμε να αλλάξει, πάει να αλλάξει με το Update. Αν γίνεται κάτι τέτοιο, τότε περνάμε μέσα στο If, και με ένα Select κοιτάμε αν υπάρχουν reservations για το συγκεκριμένο ταξίδι. Αν υπάρχουν τότε ακυρώνουμε την ενημέρωση και βγάζουμε μήνυμα λάθους.

**Παράδειγμα Υλοποίησης:**

1. Πηγαίνοντας να αλλάξουμε την ημερομηνία αναχώρησης σε ένα ταξίδι που έχει ήδη reservations, παίρνουμε το παρακάτω μήνυμα:





1. Πηγαίνοντας να αλλάξουμε την ημερομηνία επιστροφής σε ένα ταξίδι που έχει ήδη reservations, παίρνουμε το παρακάτω μήνυμα:





1. Πηγαίνοντας να αλλάξουμε το κόστος σε ένα ταξίδι που έχει ήδη reservations, παίρνουμε το παρακάτω μήνυμα:





1. Πηγαίνοντας να αλλάξουμε όμως την ημερομηνία αναχώρησης σε ένα ταξίδι που δεν έχει reservations:



Παρατηρούμε πως σε αυτήν την περίπτωση μας αφήνει να αλλάξουμε την ημερομηνία αναχώρησης καθώς σε αυτό το trip δεν έχει γίνει κανένα reservation ακόμα.

3.1.4.3

**Κώδικας:**

DROP TRIGGER IF EXISTS no\_decrease;

DELIMITER $

CREATE TRIGGER no\_decrease BEFORE UPDATE ON worker

FOR EACH ROW

BEGIN

IF (NEW.wrk\_salary<OLD.wrk\_salary)

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'You cannot decrease the salary of a worker';

END IF;

END$

DELIMITER ;

**Επεξήγηση Κώδικα:** Αυτό το Trigger ενεργοποιείται κάθε φορά πριν γίνει κάποια ενημέρωση στον πίνακα worker. Ως παράμετρο σε ένα If Statement ελέγχουμε αν το update γίνεται πάνω στον μισθό του υπαλλήλου. Αν γίνεται εκεί, τότε μέσα στο If Statement ελέγχουμε αν το update έχει στόχο να μειώσει ή να αυξήσει τον μισθό. Αν προσπαθεί να τον μειώσει, τότε βγάζουμε μήνυμα λάθους και απαγορεύουμε την αλλαγή, αν προσπαθεί να τον αυξήσει τότε προχωράμε με το Update.

**Παράδειγμα Υλοποίησης:**