|  |
| --- |
| Ράικος Ιάσων  ΑΜ: 1084552  Έτος: 3o  Μακρής Αντώνιος Ορέστης  ΑΜ: 1084516  Έτος: 3o  Δελημπαλταδάκης Γρηγόρης  ΑΜ: 1084647  Έτος: 3o |

|  |
| --- |
|  |
| Project Εργαστηρίου Βάσεων Δεδομένων 2022-2023 |
|  |

Περιεχόμενα

[Μέρος Α: Σχεδιασμός ΒΔ και SQL 2](#_Toc126362599)

[Κεφάλαιο 1 - Περιγραφή Βάσης 2](#_Toc126362600)

[**Περιγραφή παραδοχών** 2](#_Toc126362601)

[**Νέοι Πίνακες** 8](#_Toc126362602)

[Κεφάλαιο 2 - Stored Procedures 9](#_Toc126362603)

[Κεφάλαιο 3 - Triggers 32](#_Toc126362604)

[**3.1.4.1** 32](#_Toc126362605)

[**3.1.4.2** 38](#_Toc126362606)

[**3.1.4.3** 39](#_Toc126362607)

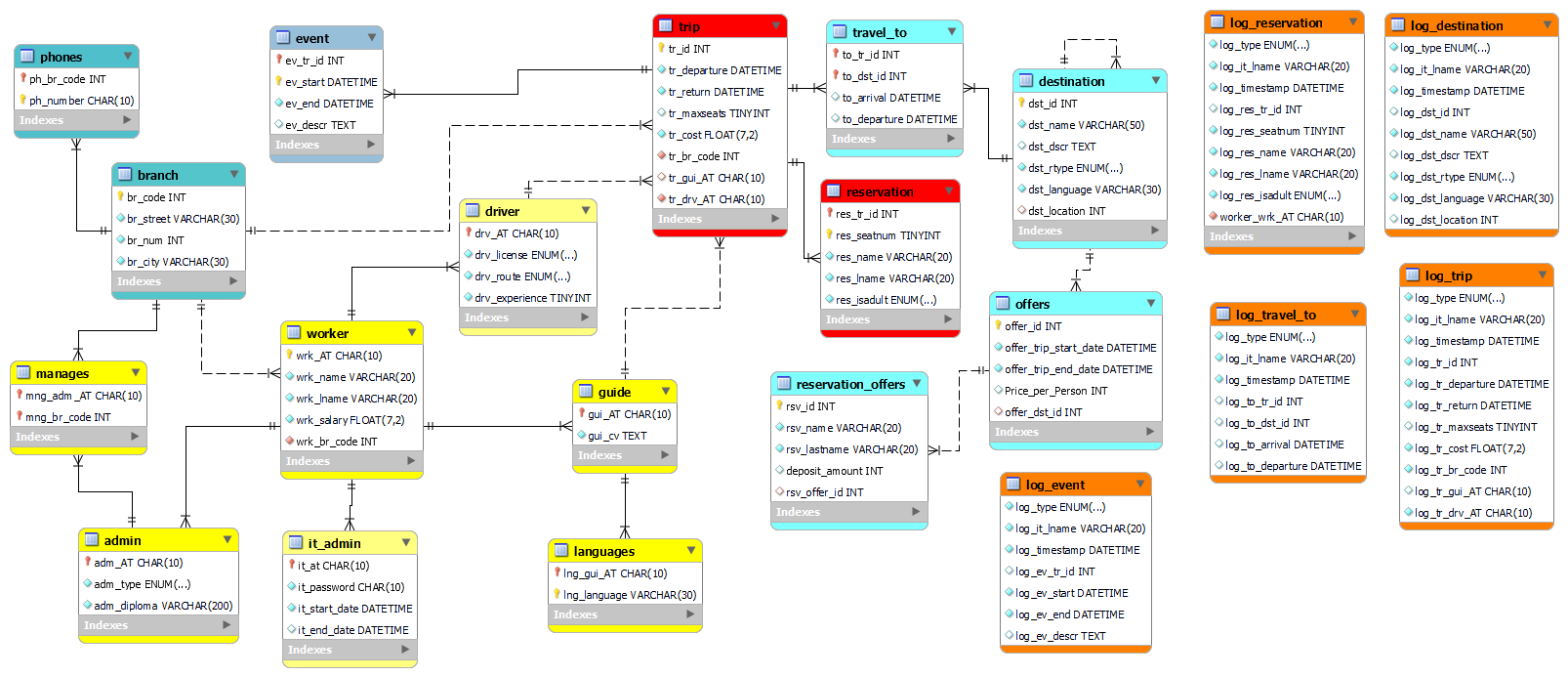
[Μέρος Β: GUI 42](#_Toc126362608)

[Κεφάλαιο 4 - Java 42](#_Toc126362609)

[Κεφάλαιο 5 43](#_Toc126362610)

# Μέρος Α: Σχεδιασμός ΒΔ και SQL

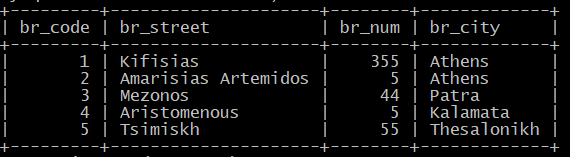
# Κεφάλαιο 1 - Περιγραφή Βάσης

*Σχεσιακό διάγραμμα της συνολικής αναθεωρημένης ΒΔ*

## **Περιγραφή παραδοχών**

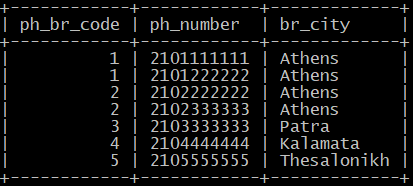
Για την σχεδίαση της βάσης δεδομένων μας κάναμε τις παρακάτω παραδοχές:

* *Υποκαταστήματα (Branches)*



Δημιουργήσαμε πέντε branches. Δύο στην Αθήνα και από ένα στην Πάτρα, Καλαμάτα και Θεσσαλονίκη.

* *Τηλέφωνα (Phones)*



Τα δύο υποκαταστήματα της Αθήνας έχουν από δύο τηλέφωνα, επειδή τα γραφεία τους είναι μεγαλύτερα, ενώ τα υπόλοιπα υποκαταστήματα έχουν από ένα.

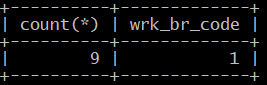
* *Εργαζόμενοι (Workers)*

*Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα*

Συνολικά όλα τα υποκαταστήματα μαζί έχουν 39 εργαζόμενους. Οι εργαζόμενοι χωρίζονται σε admins (διαχειριστές), it\_admins (υπεύθυνοι πληροφορικής), drives (οδηγούς) και guides (ξεναγούς). Κάθε branch έχει τρεις διαχειριστές, από έναν για την κάθε κατηγορία διαχείρισης (logistics, accounting, administrative). Σε κάποια υποκαταστήματα (branches 1 και 5) έχουμε θέσει κάποιους διαχειριστές να εκτελούν και αρμοδιότητες ξεναγού.

**Branch 1:**



Το Branch 1, έχει συνολικά 9 εργαζόμενους.

Admins:

Εικόνα που περιέχει κείμενο

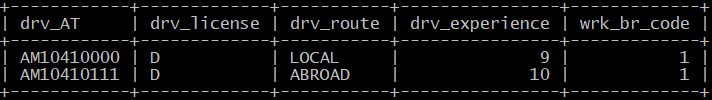
Περιγραφή που δημιουργήθηκε αυτόματα

IT Admins:

Εικόνα που περιέχει κείμενο

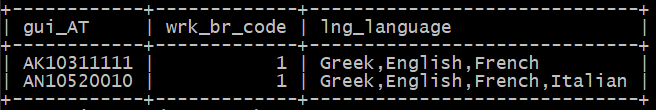
Περιγραφή που δημιουργήθηκε αυτόματα

Drivers:



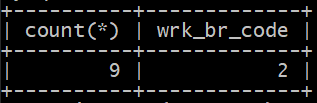
Έχουμε δύο drivers για το branch 1 έναν για ταξίδια στο εξωτερικό και έναν για το εξωτερικό.

Guides:



Παρατηρούμε πως ο logistics admin του branch 1 με ΑΤ = ΑΚ10311111 είναι και guide.

**Branch 2:**



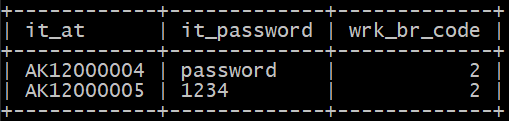
Το Branch 2, έχει επίσης 9 εργαζόμενους.

Admins:

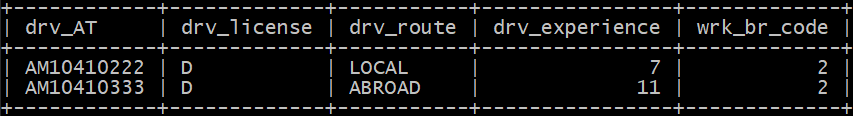
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

IT Admins:

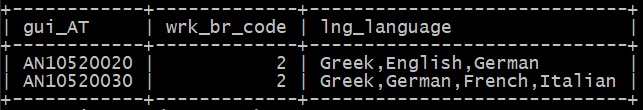


Drivers:

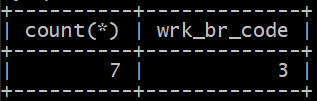


Έχουμε δύο drivers για το branch 2 έναν για ταξίδια στο εξωτερικό και έναν για το εξωτερικό.

Guides:



**Branch 3:**



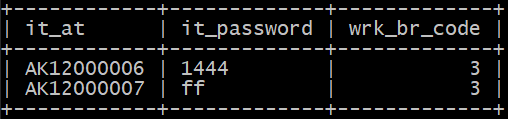
Το Branch 3, που είναι στην Πάτρα έχει συνολικά 7 εργαζόμενους.

Admins:

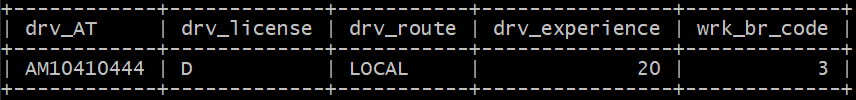
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

IT Admins:

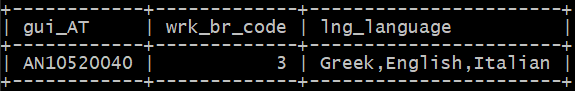


Drivers:

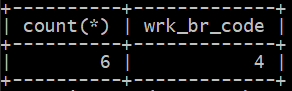


Έχουμε έναν driver για το branch 3, ο οποίος ταξιδεύει μόνο στο εσωτερικό. Το υποκατάστημα 3 στην Πάτρα δηλαδή κάνει μόνο ταξίδια στο εσωτερικό.

Guides:



**Branch 4:**



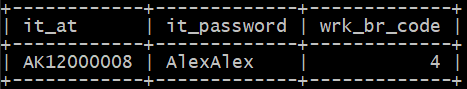
Το Branch 5, που είναι στην Καλαμάτα έχει συνολικά 4 εργαζόμενους.

Admins:

Εικόνα που περιέχει κείμενο

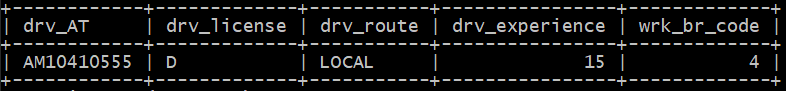
Περιγραφή που δημιουργήθηκε αυτόματα

IT Admins:



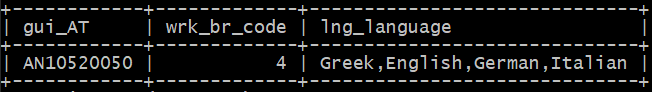
Βλέπουμε πως αυτό το branch έχει μόνο έναν υπεύθυνο πληροφορικής.

Drivers:

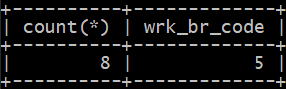


Έχουμε έναν driver για το branch 4, ο οποίος ταξιδεύει μόνο στο εσωτερικό. Το υποκατάστημα 4 στην Καλαμάτα δηλαδή κάνει μόνο ταξίδια στο εσωτερικό.

Guides:



**Branch 5:**



Το Branch 5, που είναι στην Θεσσαλονίκη έχει συνολικά 8 εργαζόμενους.

Admins:

Εικόνα που περιέχει κείμενο

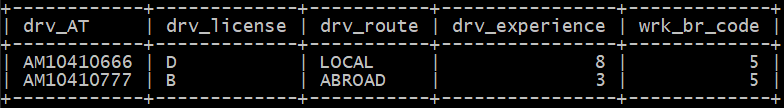
Περιγραφή που δημιουργήθηκε αυτόματα

IT Admins:

Εικόνα που περιέχει κείμενο

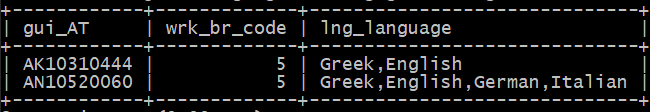
Περιγραφή που δημιουργήθηκε αυτόματα

Drivers:



Έχουμε δύο driver για το branch 5, έναν για ταξίδια στο εξωτερικό και έναν για το εσωτερικό. Το υποκατάστημα 5 στην Θεσσαλονίκη δηλαδή διοργανώνει ταξίδια και των δύο ειδών.

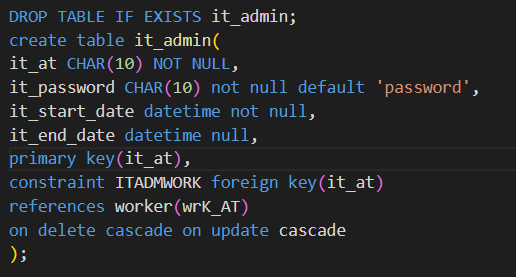
Guides:



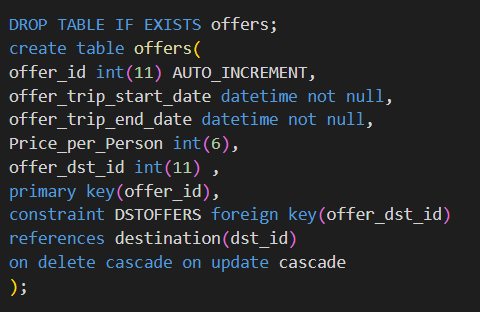
Παρατηρούμε, ότι όπως και στο branch 1 ένας από τους ξεναγούς του branch 5 είναι logistics admin.

## **Νέοι Πίνακες**

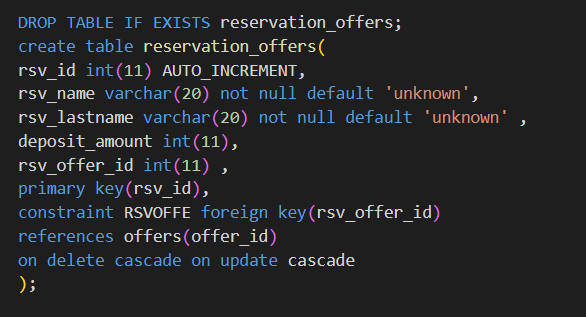
Φτιάχνουμε τον πίνακα **it\_admin** που έχει τέσσερα πεδία και τον χρειαζόμαστε για το **GUI** και για να αναπαραστήσουμε έναν **Υπεύθυνο πληροφορικής (IT)** ενός πρακτορείου. Τη βάση δεδομένων την χειρίζεται ο **Υπεύθυνος πληροφορικής (IT)** του πρακτορείου. Είναι **worker**, και έχει μοναδικό κλειδί το **it\_at** (**char(10)** ) το οποίο είναι ξένο κλειδί στο **worker.wrk\_AT**. Επίσης έχει ένα κωδικό 10 χαρακτήρων με τιμή **default** **‘password’** με τον οποίο θα κάνουμε **login** στο **GUI**. Αποθηκεύεται κιόλας η ημερομηνία (**start\_date**) που ανέλαβε τον ρόλο αυτόν και το πεδίο αυτό είναι **not null**. Επίσης υπάρχει **end\_date** ημερομηνία για όσους **IT** έχουν φύγει από το πρακτορείο. Μπορεί να υπάρχουν περισσότεροι από ένας **Υπεύθυνοι πληροφορικής** την ίδια χρονική περίοδο.



Ο πίνακας **offers** έχει πέντε πεδία και τον χρειαζόμαστε για να υπάρχει η δυνατότητα να καταχωρούνται προσφορές ταξιδιών. Το πρώτο πεδίο είναι ο κωδικός της προσφοράς ο οποίος είναι μοναδικός. Το δεύτερο και τρίτο πεδίο είναι η περίοδος που μπορεί να πραγματοποιηθεί ένα ταξίδι (δεν προσδιορίζει την διάρκεια του ταξιδιού αλλά το διάστημα που είναι δυνατόν να πραγματοποιηθεί). Το τέταρτο πεδίο είναι το κόστος ανά άτομο και το πέμπτο πεδίο είναι ο κωδικός του προορισμού(**dst\_id**) ο οποίος είναι ξένο κλειδί στο **destination.dst\_id**.



Ο πίνακας **reservation\_offers** έχει πέντε πεδία και αναπαριστά τις κρατήσεις στις προσφορές ταξιδιών. Το πρώτο πεδίο είναι ο κωδικό κράτησης ο οποίος είναι μοναδικός. Το δεύτερο και τρίτο πεδίο είναι το όνομα και επώνυμο του πελάτη. Το τέταρτο πεδίο είναι το ποσό προκαταβολής και το πέμπτο πεδίο είναι ο κωδικός προσφοράς ταξιδιού ο οποίος είναι ξένο κλειδί στο **offers.offer\_id**. Με τους δύο πίνακες **offers** και **reservation\_offers** το ταξιδιωτικό πρακτορείο κάνει σχεδιασμό μελλοντικών εκδρομών.



# Κεφάλαιο 2 - Stored Procedures

**3.1.3.1**

DROP PROCEDURE IF EXISTS insert\_driver;

DELIMITER $

CREATE PROCEDURE insert\_driver(IN AT char(10),IN name varchar(20),IN lname varchar(20),IN salary float(7,2),IN license ENUM('A','B','C','D'),IN route ENUM('LOCAL','ABROAD'),IN experience tinyint(4))

BEGIN

DECLARE code INT;

SELECT br\_code INTO code FROM (SELECT branch.br\_code,IFNULL(count(wrk\_AT),0) AS 'Number of Drivers' FROM driver INNER JOIN worker ON drv\_AT=wrk\_AT RIGHT JOIN branch ON br\_code=wrk\_br\_code GROUP BY br\_code ORDER BY IFNULL(count(wrk\_AT),0) ASC LIMIT 1) AS minimum\_drivers;

INSERT INTO worker VALUES (AT,name,lname,salary,code);

INSERT INTO driver VALUES (AT,license,route,experience);

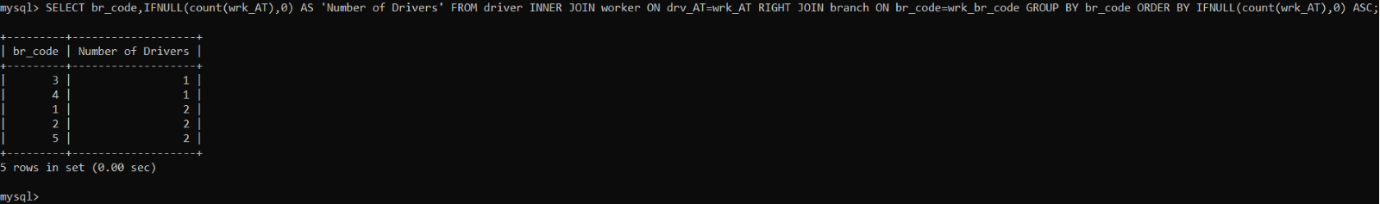
END$

DELIMITER ;

Στο **procedure** αυτό δίνονται ως είσοδοι τα στοιχεία ενός νέου οδηγού (αριθμός ταυτότητας, όνομα, επώνυμο, μισθός τύπος άδειας, τύπος δρομολογίου και μήνες εμπειρίας). Η **procedure** εντοπίζει το υποκατάστημα με τους λιγότερους οδηγούς και εισάγει τον οδηγό ως υπάλληλο του συγκεκριμένου υποκαταστήματος(τον εισάγει στους πίνακες worker και driver).Ο κώδικας στην ουσία αναθέτει την τιμή του br\_code του υποκαταστήματος με τους λιγότερους οδηγούς στην τοπική μεταβλητή code και μετά κάνει τα κατάλληλα **insert**.

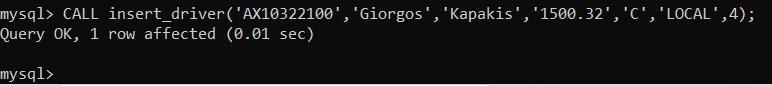
SELECT br\_code,IFNULL(count(wrk\_AT),0) AS 'Number of Drivers' FROM driver INNER JOIN worker ON drv\_AT=wrk\_AT RIGHT JOIN branch ON br\_code=wrk\_br\_code GROUP BY br\_code ORDER BY IFNULL(count(wrk\_AT),0) ASC;

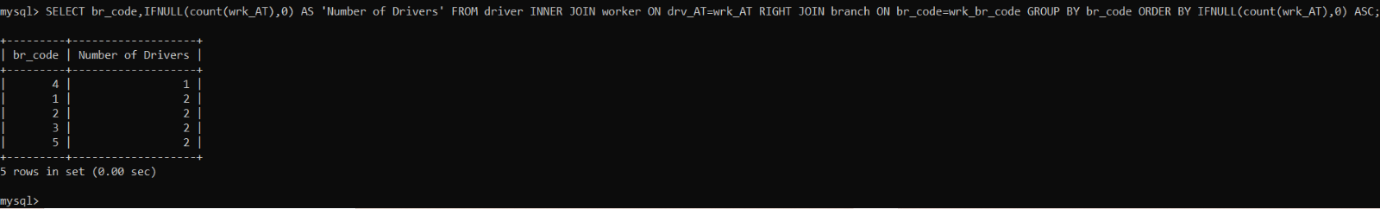
Πρώτα εκτελούμε αυτό το **select** για να δούμε πόσους οδηγούς έχει κάθε branch.

****

CALL insert\_driver('AX10322100','Giorgos','Kapakis','1500.32','C','LOCAL',4);

Έπειτα καλούμε την **insert\_driver**.

****Και ξανακαλούμε το **select** που είχαμε αναφέρει.

**** Βλέπουμε ότι ο οδηγός προστέθηκε στο branch με br\_code=3, γιατί ήταν ένα από τα branch με τους λιγότερους οδηγούς. Αν κάποια branch έχουν ίσο σε αριθμό οδηγούς και είναι αυτά με τους λιγότερους οδηγούς σε σχέση με τα άλλα τότε ο νέος οδηγός προστίθεται στο branch με το μικρότερο id.

**3.1.3.2**

DROP PROCEDURE IF EXISTS search\_trip;

DELIMITER $

CREATE PROCEDURE search\_trip(IN code INT,IN start\_date datetime,IN end\_date datetime)

BEGIN

DECLARE trip\_id INT;

DECLARE dep DATETIME;

DECLARE ret DATETIME;

DECLARE maxseats TINYINT;

DECLARE cost FLOAT(7,2);

DECLARE br\_code INT;

DECLARE guide\_AT CHAR(10);

DECLARE driver\_AT CHAR(10);

DECLARE finishedFlag INT;

DECLARE no\_of\_reservations INT;

DECLARE driver\_name VARCHAR(20);

DECLARE driver\_lname VARCHAR(20);

DECLARE guide\_name VARCHAR(20);

DECLARE guide\_lname VARCHAR(20);

DECLARE empty\_seats INT;

DECLARE tripCurs CURSOR FOR SELECT \* FROM trip WHERE tr\_departure>=start\_date AND tr\_departure<=end\_date AND tr\_br\_code = code;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finishedFlag=1;

OPEN tripCurs;

SET finishedFlag=0;

FETCH tripCurs INTO trip\_id,dep,ret,maxseats,cost,br\_code,guide\_AT,driver\_AT;

WHILE(finishedFlag=0)

DO

SELECT COUNT(\*) INTO no\_of\_reservations FROM reservation INNER JOIN trip ON res\_tr\_id=trip\_id WHERE tr\_departure>=start\_date AND tr\_departure<=end\_date AND tr\_br\_code=code AND res\_tr\_id=tr\_id;

SELECT wrk\_name,wrk\_lname INTO driver\_name,driver\_lname FROM trip INNER JOIN worker ON tr\_drv\_AT=wrk\_AT WHERE tr\_id=trip\_id;

SELECT wrk\_name,wrk\_lname INTO guide\_name,guide\_lname FROM trip INNER JOIN worker ON tr\_gui\_AT=wrk\_AT WHERE tr\_id=trip\_id;

SET empty\_seats=maxseats-no\_of\_reservations;

SELECT trip\_id AS 'Trip Id',cost AS 'Cost',maxseats AS 'Maxseats',no\_of\_reservations AS 'Number of Reservations',empty\_seats AS 'Empty Seats',driver\_lname AS 'Driver Lastname',driver\_name AS 'Driver Name',guide\_lname AS 'Guide Lastname',guide\_name AS 'Guide Name',dep AS 'Departure',ret AS 'Return';

FETCH tripCurs INTO trip\_id,dep,ret,maxseats,cost,br\_code,guide\_AT,driver\_AT;

END WHILE;

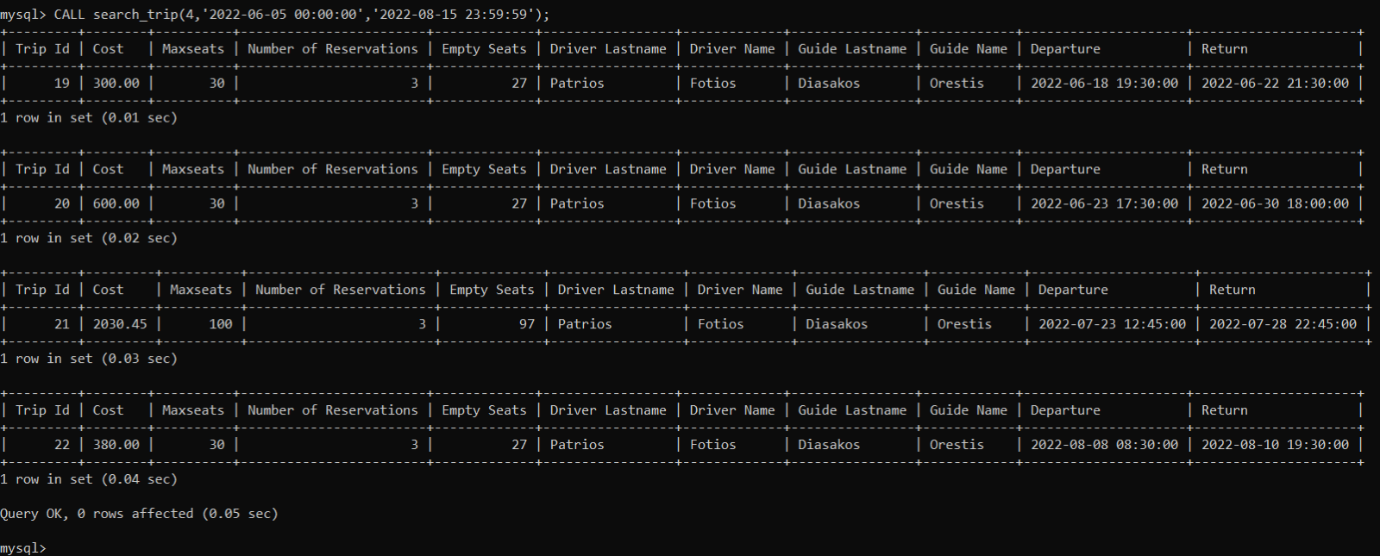
CLOSE tripCurs;

END$

DELIMITER ;

Στο **procedure** αυτό δίνονται ως ορίσματα ο κωδικός ενός υποκαταστήματος και δύο ημερομηνίες. Για τα ταξίδια (trip) που διοργανώνονται από το υποκατάστημα του οποίου δόθηκε ο κωδικός, και των οποίων η ημερομηνία αναχώρησης είναι μέσα στο διάστημα που δόθηκε, επιστρέφονται το κόστος ταξιδιού (tr\_cost ), οι μέγιστες θέσεις(maxseat), το σύνολο κρατήσεων (reservations), οι κενές θέσεις (maxseat – σύνολο κρατήσεων), το επώνυμο και το όνομα του οδηγού και του ξεναγού και τέλος η ημερομηνία αναχώρησης και επιστροφής. Δημιουργούμε έναν **cursor** με τον οποίο διατρέχουμε τον πίνακα **trip** και αποθηκεύουμε τις ζητούμενες πληροφορίες στις τοπικές μεταβλητές που δηλώσαμε και στο τέλος τις εκτυπώνουμε.

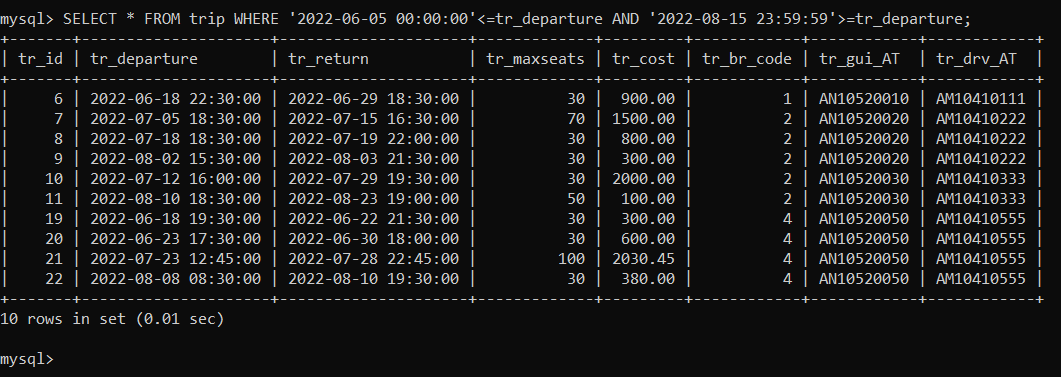
CALL search\_trip(3,'2022-01-05 00:00:00','2022-08-15 23:59:59');

****Καλούμε την συνάρτηση  **search\_trip**.

SELECT \* FROM trip WHERE '2022-06-05 00:00:00'<=tr\_departure AND '2022-08-15 23:59:59'>=tr\_departure;

SELECT \* FROM reservation;

Μετά εκτελούμε τα παραπάνω **select** για να ελέγξουμε αν λειτουργεί σωστά το **procedure**.

****



Βλέπουμε ότι επιλέγει τα σωστά **trips**.

**3.1.3.3**

DROP PROCEDURE IF EXISTS delete\_worker;

DELIMITER $

CREATE PROCEDURE delete\_worker(IN name varchar(20),IN lname varchar(20))

BEGIN

IF exists(SELECT \* FROM worker INNER JOIN admin ON wrk\_AT=adm\_AT WHERE wrk\_lname=lname AND wrk\_name=name AND adm\_type="ADMINISTRATIVE")

THEN

SELECT "This person is an admin in a branch therefore the deletion is restricted!" AS Error;

ELSEIF exists(SELECT \* FROM worker WHERE wrk\_lname=lname AND wrk\_name=name)

THEN

DELETE FROM worker WHERE wrk\_lname=lname AND wrk\_name=name;

ELSE

SELECT "This person isn't a worker in any branch!" AS Error;

END IF;

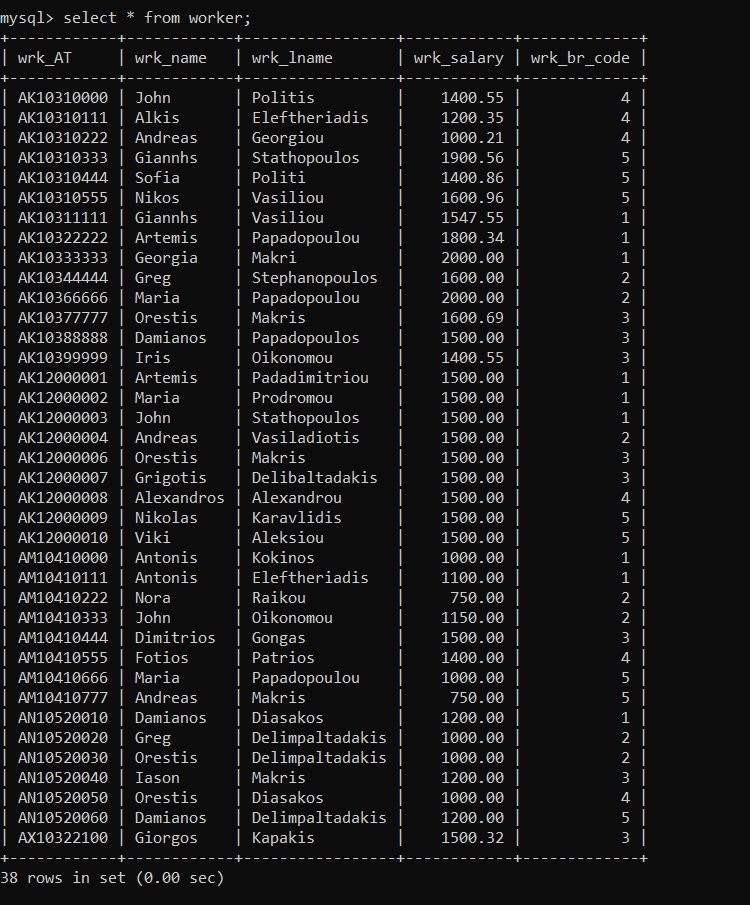
END$

DELIMITER ;

Το **procedure** αυτό παίρνει ως όρισμα το όνομα και το επώνυμο ενός υπαλλήλου. Αν είναι διοικητικός τον διαγράφει. Αν ο υπάλληλος είναι διευθυντής υποκαταστήματος τότε εμφανίζεται μήνυμα ότι είναι διευθυντής του υποκαταστήματος και δεν επιτρέπει τη διαγραφή. Με ένα if ελέγχουμε αν ένας υπάλληλος είναι διευθυντής και αν ισχύει τότε εκτυπώνουμε ένα error.Αν δεν είναι τότε μεταβαίνουμε στο επόμενο if και ελέγχουμε αν υπάρχει κάποιος υπάλληλος με το ονοματεπώνυμο που δόθηκε. Τέλος, αν δεν υπάρχει υπάλληλος τότε εκτυπώνουμε αντίστοιχο error.

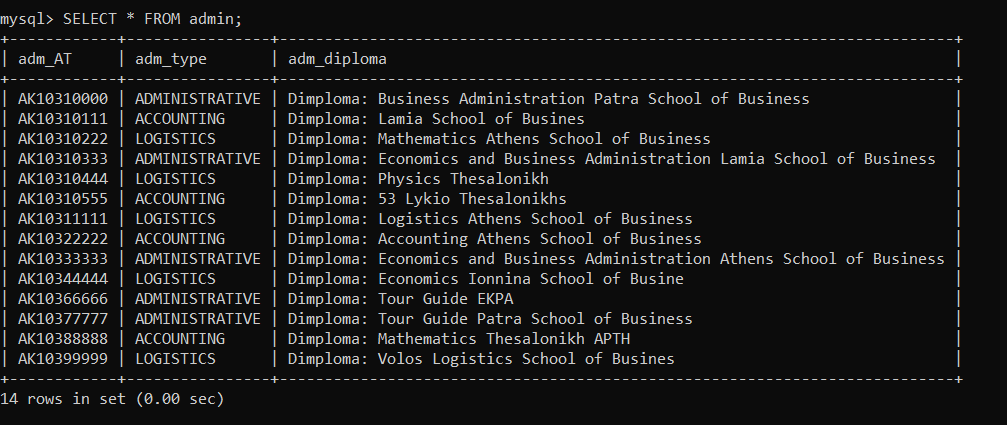
SELECT \* FROM worker;

Εκτελούμε το παραπάνω **select**, για να δούμε τους workers.

****

SELECT \* FROM admin;

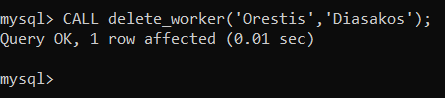
Εκτελούμε το παραπάνω select, για να δούμε τους admins.

****

Έτσι μπορούμε να δόυμε τους workers από τους οποίους θα επιλέξουμε να κάνουμε **delete\_worker**.

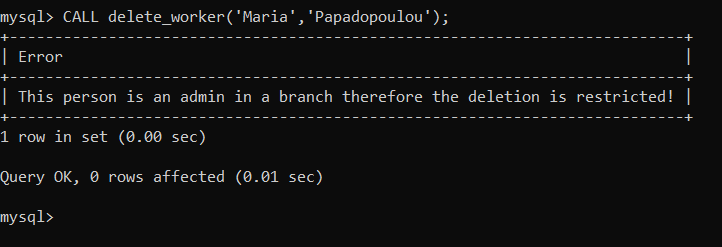
CALL delete\_worker('Orestis','Diasakos');

Εκτελούμε το παραπάνω **call**.

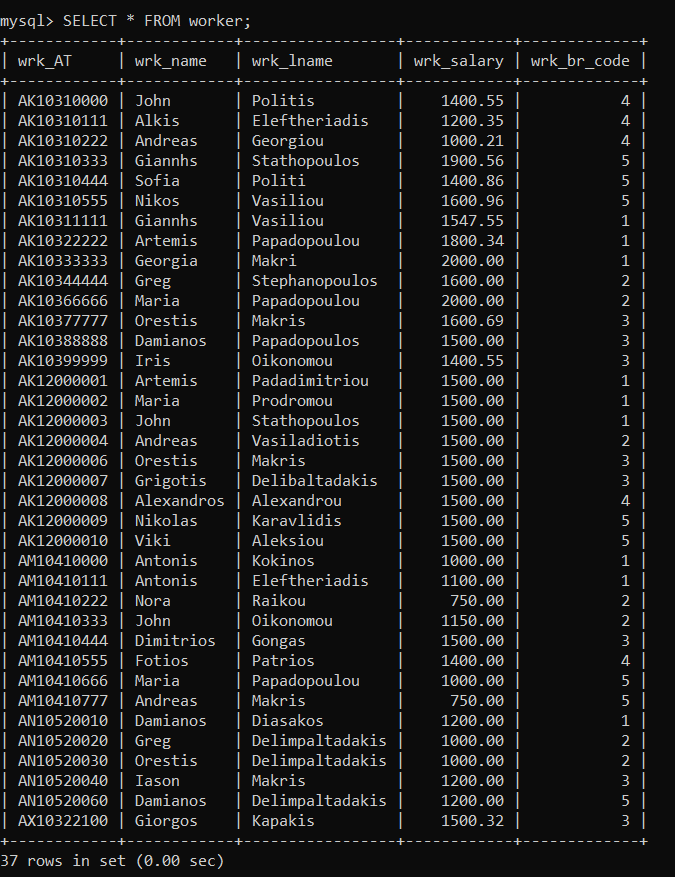
****

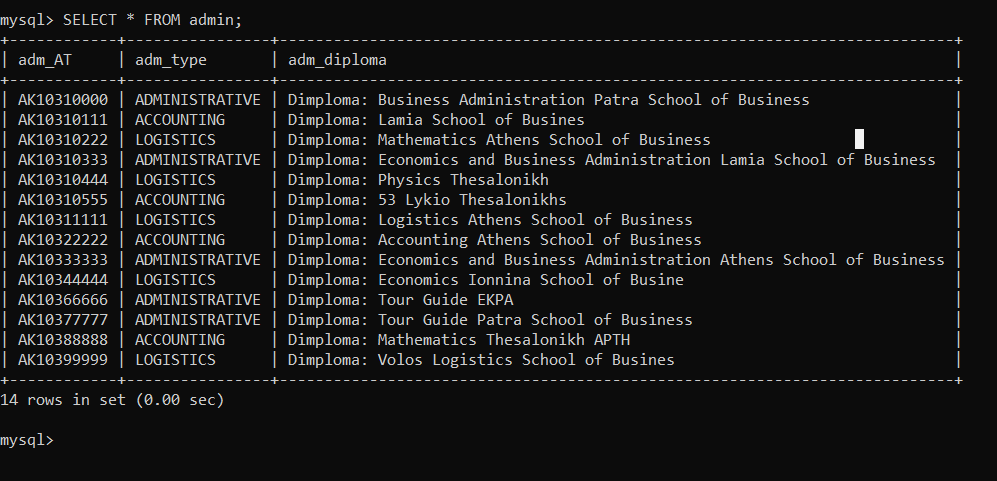
CALL delete\_worker('Maria','Papadopoulou');

Κάνουμε πάλι κλήση της **delete\_worker** αλλά με το παραπάνω **call**.

****

Εκτελούμε ξανά τα δύο προηγούμενα **select** με την ίδια σειρά.

****

Βλέπουμε ότι ο Orestis Diasakos που διαγράψαμε στην αρχή, διαγράφηκε με επιτυχία ενώ η Maria Papadopoulou δεν διαγράφηκε επειδή είναι διευθύντρια του υποκαταστήματος με branch\_id = 2.

**3.1.3.4**

DROP INDEX res\_ind ON reservation\_offers;

CREATE INDEX res\_ind ON reservation\_offers(deposit\_amount, rsv\_lastname, rsv\_offer\_id);

Πρώτα από όλα λόγω του μεγάλου πλήθους εγγραφών, δημιουργούμε ένα ευρετήριο στον πίνακα reservation\_offers ώστε το πρόγραμμα να αναζητεί πιο αποδοτικά τις εγγραφές του πίνακα. Κάνουμε πρώτα **drop** σε περίπτωση που ήδη υπάρχει και το δημιουργούμε για τα πεδία **rsv\_lastname** ,**deposit\_amount** και **rsv\_offer\_id**. Δημιουργούμε το index για την στήλη **deposit\_amount,** επειδή χρησιμοποιούμε αυτό το πεδίο μέσα σε **WHERE** **condition** για το **procedure** του υποερωτήματος **α)** και για τις **rsv\_lastname, rsv\_offer\_id,** αντίστοιχα επειδή τα χρησιμοποιούμε σε πολλά **conditions** του **procedure** του υποερωτήματος **β)**.

**α)**

DROP PROCEDURE IF EXISTS res\_range;

DELIMITER $

CREATE PROCEDURE res\_range(IN r1 INT, IN r2 INT)

BEGIN

    SELECT rsv\_name AS 'Firstname', rsv\_lastname AS 'Lastname'

    FROM reservation\_offers

    WHERE deposit\_amount >= r1 AND deposit\_amount<=r2;

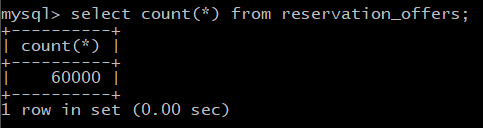
END$

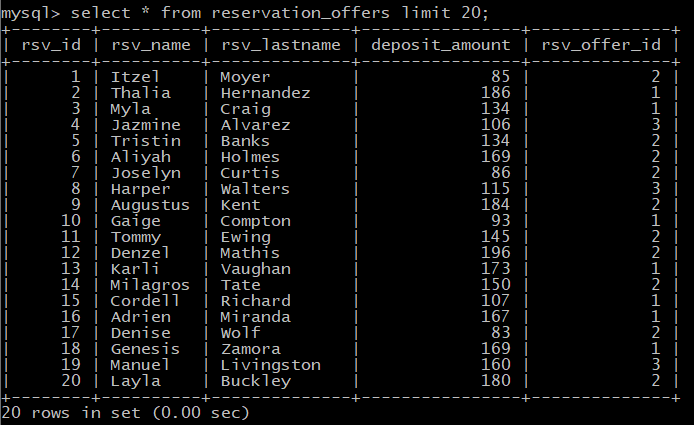
DELIMITER ;

Το **procedure** αυτό παίρνει ως παραμέτρους εισόδου δύο τιμές (π.χ. 50, 200) και τυπώνει τους πελάτες (επώνυμο, όνομα) που έκαναν κρατήσεις σε προσφορές ταξιδιών και πλήρωσαν για προκαταβολή κάποιο ποσό ανάμεσα στις δύο τιμές.

*Έλεγχος*

Έχουμε τον πίνακα reservation\_offers, ο οποίος έχει 60000 εγγραφές.

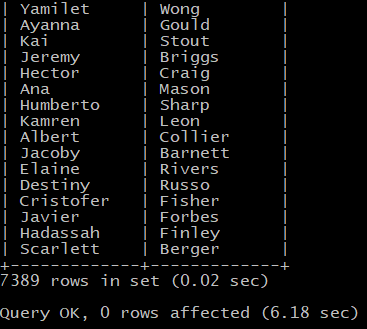
****

****

Έστω ότι καλούμε την procedure res\_range για το εύρος προκαταβολών [80,95].

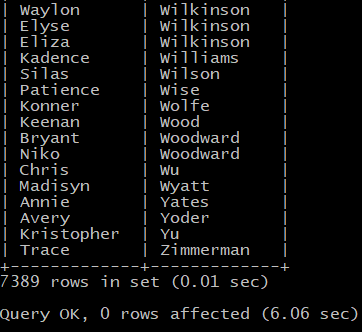
**Χωρίς το ευρετήριο**:

****

****

Βλέπουμε ότι το procedure εκτελέστηκε σε 0.02 s.

**Με το ευρετήριο:**

****

Το procedure εκτελείται σε 0.01 s.

**β)**

DROP PROCEDURE IF EXISTS res\_names;

DELIMITER $

CREATE PROCEDURE res\_names(IN lname VARCHAR(20))

BEGIN

    DECLARE flag INT;

    DECLARE offer\_id INT;

    DECLARE no\_of\_res INT;

    DECLARE finishedFlag INT;

    DECLARE multResCurs CURSOR FOR

    SELECT rsv\_offer\_id, COUNT(rsv\_offer\_id)

    FROM reservation\_offers

    WHERE rsv\_lastname=lname

    GROUP BY rsv\_offer\_id

    HAVING COUNT(rsv\_offer\_id)>1

    ORDER BY rsv\_offer\_id;

    DECLARE singleResCurs CURSOR FOR

    SELECT rsv\_offer\_id, COUNT(rsv\_offer\_id)

    FROM reservation\_offers

    WHERE rsv\_lastname=lname

    GROUP BY rsv\_offer\_id

    HAVING COUNT(rsv\_offer\_id)=1

    ORDER BY rsv\_offer\_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finishedFlag=1;

    OPEN multResCurs;

    SET finishedFlag=0;

    FETCH multResCurs INTO offer\_id, no\_of\_res;

    SET flag = 0;

    WHILE(finishedFlag=0) DO

        IF(flag = 0) THEN

            SELECT 'Reserved offers for last name' AS '', lname AS '';

        END IF;

        SELECT offer\_id AS 'Offer Id', no\_of\_res AS 'Number of reserved offers';

        FETCH multResCurs INTO offer\_id,no\_of\_res;

        SET flag =  1;

    END WHILE;

    CLOSE multResCurs;

    OPEN singleResCurs;

    SET finishedFlag=0;

    SET flag = 0;

    FETCH singleResCurs INTO offer\_id, no\_of\_res;

    WHILE(finishedFlag=0) DO

        IF(flag = 0) THEN

            SELECT 'Reserved offers with one reservation per offer' AS '';

        END IF;

        SELECT rsv\_name, rsv\_lastname, rsv\_offer\_id

        FROM reservation\_offers

        WHERE rsv\_offer\_id = offer\_id AND rsv\_lastname = lname;

        SET flag = 1;

        FETCH singleResCurs INTO offer\_id,no\_of\_res;

    END WHILE;

END$

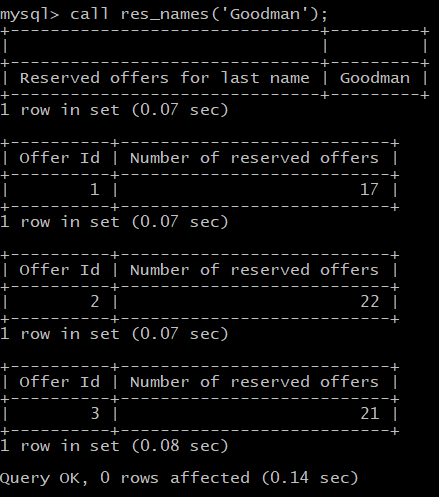
DELIMITER ;

Στο **procedure** αυτό δίνεται ως παράμετρος εισόδου το επώνυμο ενός πελάτη και εκτυπώνει τα ονοματεπώνυμα των πελατών με το επώνυμο αυτό και την προσφορά ταξιδιού στην οποία έχουν κάνει κράτηση. Σε περίπτωση που υπάρχουν πάνω από ένας επιστρέφει το πλήθος των πελατών με το επώνυμο αυτό, ανά προσφορά ταξιδιού.

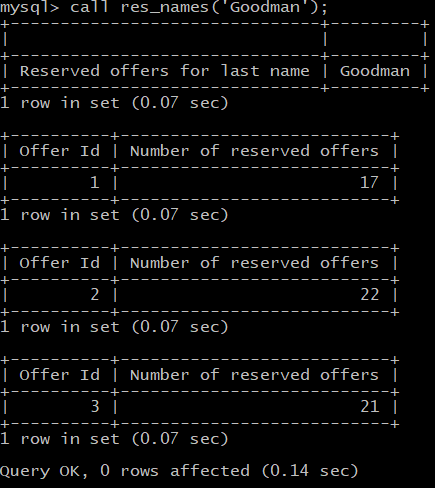
Η υλοποίηση έγινε με χρήση δύο cursors, **DECLARE multResCurs CURSOR** και **DECLARE singleResCurs CURSOR** τα οποία διατρέχουν το πίνακα reservation\_offers και εντοπίζουν και εκτυπώνουν πολλαπλές και μονές κρατήσεις αντίστοιχα.

*Έλεγχος:*

**Χωρίς ευρετήριο:**

****

**Με ευρετήριο:**

****

Με την χρήση ευρετηρίου η εκτέλεση είναι κατά 0.01 s πιο γρήγορη.

Ο πίνακας reservation\_offers δημιουργήθηκε με την χρήση του δοθέντος αρχείου names.txt που περιέχει 10000 ονόματα, με τα οποία εκτελέσαμε τα INSERT για την εισαγωγή 60000 εγγραφών. Έτσι κάθε επώνυμο έχει κάνει πολλαπλές κρατήσεις σε κάθε προσφορά και για αυτό παραπάνω εκτυπώνεται ένας πίνακας με πολλαπλές κρατήσεις ανά offer.

Για να γίνει σωστός έλεγχος του procedure λοιπόν, εισάγουμε μερικές επιπλέον κρατήσεις:

INSERT INTO reservation\_offers VALUES(null,'Greg','Deli',50,1);

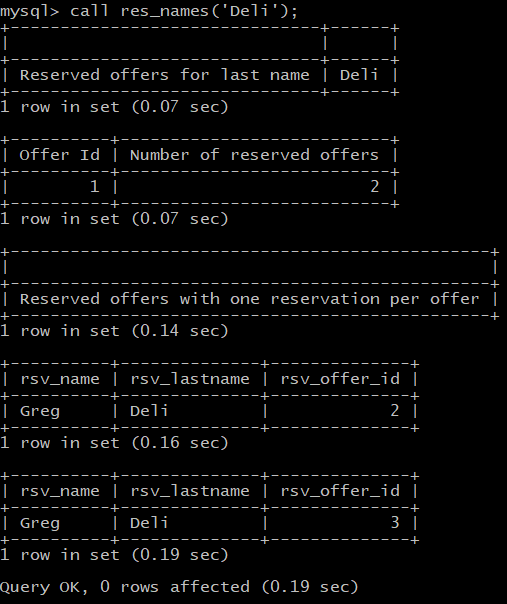
INSERT INTO reservation\_offers VALUES(null,'Maria','Deli',50,1);

INSERT INTO reservation\_offers VALUES(null,'Greg','Deli',50,2);

INSERT INTO reservation\_offers VALUES(null,'Greg','Deli',50,3);

Δηλαδή εισάγουμε 3 κρατήσεις για τον πελάτη Greg Deli, μία σε κάθε offer και μία κράτηση για την Maria Deli στο offer me id=1.

Αποτελέσματα:

****

Βλέπουμε λοιπόν ότι για το επώνυμο Deli έχουμε δύο κρατήσεις για την προσφορά 1 (μία από τον Greg και μία από την Maria) και από μία κράτηση στις άλλες δυο προσφορές από τον Greg.

**Επιπλέον Procedures**

1. **create\_user()**

Πέρα από τα stored procedures που ζητούνταν προσθέσαμε δυο επιπλέον τα οποία δημιουργούν ένα user για την βάση με όλα τα privileges ή μόνο με select privileges.

/\*Procedure pou dhmioyrgei enan neo user gia tin vasi travel\_agency. Prepei na tin kalesoume afou kanoume insert enan it\_admin\*/

DROP PROCEDURE IF EXISTS create\_user;

DELIMITER $

CREATE PROCEDURE create\_user(IN it\_username VARCHAR(20),IN it\_password CHAR(10))

BEGIN

    /\*prepei na xrisimopoiisw tin CONCAT() giati den mas epitrepei na ektelesoume tin CREATE USER me metavlites\*/

    SET @cr\_user = CONCAT

    ('

    CREATE USER "',it\_username,'"@"localhost" IDENTIFIED BY "',it\_password,'" '

    );

    PREPARE cr FROM @cr\_user;

    EXECUTE cr;

    DEALLOCATE PREPARE cr;

    /\*omoiws gia to GRANT PRIVILEGES\*/

    SET @gr\_user = CONCAT

    ('

    GRANT ALL ON travel\_agency.\* TO "',it\_username,'"@"localhost" '

    );

    PREPARE gr FROM @gr\_user;

    EXECUTE gr;

    DEALLOCATE PREPARE gr;

    /\*gia na mporei na kanei select kai delete to user ston pinaka mysql.db\*/

    SET @gr\_user = CONCAT

    ('

    GRANT SELECT,DELETE ON mysql.db TO "',it\_username,'"@"localhost" '

    );

    PREPARE gr FROM @gr\_user;

    EXECUTE gr;

    DEALLOCATE PREPARE gr;

    /\*gia na mporei na kanei create user\*/

    SET @gr\_cr\_user = CONCAT

    ('

    GRANT CREATE USER ON \*.\* TO "',it\_username,'"@"localhost" '

    );

    PREPARE gr\_cr FROM @gr\_cr\_user;

    EXECUTE gr\_cr;

    DEALLOCATE PREPARE gr\_cr;

    SET @flush = "FLUSH PRIVILEGES";

    PREPARE stmt FROM @flush;

    EXECUTE stmt;

    DEALLOCATE PREPARE stmt;

END$

DELIMITER ;

/\*user mono me select privilages\*/

DROP PROCEDURE IF EXISTS create\_select\_user;

DELIMITER $

CREATE PROCEDURE create\_select\_user(IN it\_username VARCHAR(20),IN it\_password CHAR(10))

BEGIN

    #prepei na xrisimopoiisw tin CONCAT() giati den mas epitrepei na ektelesoume tin CREATE USER me metavlites

    SET @cr\_user = CONCAT

    ('

    CREATE USER "',it\_username,'"@"localhost" IDENTIFIED BY "',it\_password,'" '

    );

    PREPARE cr FROM @cr\_user;

    EXECUTE cr;

    DEALLOCATE PREPARE cr;

    /\*PRIVILEGES SELECT kai EXECUTE gia na mporei na kalei procedures\*/

    SET @gr\_user = CONCAT

    ('

    GRANT EXECUTE, SELECT ON travel\_agency.\* TO "',it\_username,'"@"localhost" '

    );

    PREPARE gr FROM @gr\_user;

    EXECUTE gr;

    DEALLOCATE PREPARE gr;

    /\*gia na mporei na kanei select kai delete to user ston pinaka mysql.db\*/

    SET @gr\_user = CONCAT

    ('

    GRANT SELECT ON mysql.db TO "',it\_username,'"@"localhost" '

    );

    PREPARE gr FROM @gr\_user;

    EXECUTE gr;

    DEALLOCATE PREPARE gr;

    SET @flush = "FLUSH PRIVILEGES";

    PREPARE stmt FROM @flush;

    EXECUTE stmt;

    DEALLOCATE PREPARE stmt;

END$

DELIMITER ;

Το ιδανικό θα ήταν να μπορούσαμε να δημιουργήσουμε τον user μέσα σε ένα trigger για τον πίνακα it\_admin κάθε φορά που κάνουμε insert σε αυτόν. Όμως η mySQL δεν επιτρέπει την εκτέλεση της εντολής CREATE USER ούτε την κλήση procedure μέσα σε trigger.

Tα procedures παίρνουν ως ορίσματα ένα username και password και εκτελούνε τα queries:

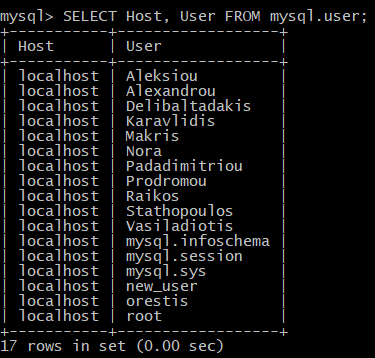
* CREATE USER <username> IDENTIFIED BY <password>, για την δημιουργία του user.
* GRANT ALL ON travel\_agency.\* TO <username>, για να έχει όλα τα δικαιώματα στην βάση μας
* GRANT EXECUTE, SELECT ON travel\_agency.\* TO <username>, στην περίπτωση που δημιουργούμε έναν χρήστη με δικαιώματα select και execute μόνο.
* GRANT SELECT,DELETE ON mysql.db TO <username> και GRANT CREATE USER ON \*.\* TO <username>, για να μπορεί να δημιουργεί και να διαγράφει χρήστες.

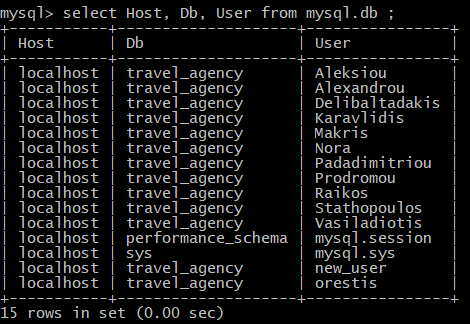
*Έλεγχος:*

Κλήση του procedure:

****

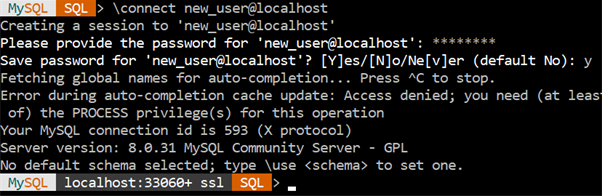
Έλεγχος του πίνακα mysql.user και mysql.db:

****

****

Βλέπουμε ότι ο new\_user προστέθηκε και στους δύο.

Είσοδος στην βάση ως τον καινούριος user:

****

Εκτέλεση ενός query:

****

Συνδεθήκαμε και εκτελέσαμε με επιτυχία ένα querry στην βάση μας.

1. **branch\_info()**

Αυτό το stored procedure εμφανίζει για κάθε υποκατάστημα τα στοιχεία του, το ονοματεπώνυμο του διευθυντή του, το σύνολο κρατήσεων και το σύνολο εσόδων (κράτησεις επί κόστος ταξιδιού). Δημιουργήθηκε για να διευκολύνει την εμφάνιση των ζητουμένων πληροφοριών των υποκαταστημάτων στο GUI.

DROP PROCEDURE IF EXISTS branch\_info;

DELIMITER $

CREATE PROCEDURE branch\_info()

BEGIN

DECLARE branch\_code INT(11);

DECLARE branch\_street VARCHAR(30);

DECLARE branch\_num INT(4);

DECLARE branch\_city VARCHAR(30);

DECLARE finishedFlag INT;

DECLARE admin\_name VARCHAR(20);

DECLARE admin\_lname VARCHAR(20);

DECLARE total\_of\_reservations INT;

DECLARE income FLOAT;

DECLARE branchCurs CURSOR FOR SELECT \* FROM branch;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finishedFlag=1;

OPEN branchCurs;

SET finishedFlag=0;

FETCH branchCurs INTO branch\_code,branch\_street,branch\_num,branch\_city;

WHILE(finishedFlag=0)

DO

SELECT COUNT(\*) INTO total\_of\_reservations FROM reservation INNER JOIN trip ON res\_tr\_id=tr\_id WHERE tr\_br\_code=branch\_code;

SELECT wrk\_name,wrk\_lname INTO admin\_name,admin\_lname FROM admin INNER JOIN worker ON adm\_AT=wrk\_AT WHERE wrk\_br\_code=branch\_code AND adm\_type='ADMINISTRATIVE';

SET income=(SELECT SUM(tr\_cost\*res\_count) FROM trip JOIN (SELECT res\_tr\_id, COUNT(\*) as res\_count FROM reservation GROUP BY res\_tr\_id) as res\_counts ON trip.tr\_id = res\_counts.res\_tr\_id WHERE tr\_br\_code = branch\_code);

SELECT branch\_code AS 'Branch code',branch\_street AS 'Street',branch\_num AS 'Number',branch\_city AS 'City',admin\_name AS 'Admin name',admin\_lname AS 'Admin last name',total\_of\_reservations AS 'Total of Reservations',income AS 'Income';

FETCH branchCurs INTO branch\_code,branch\_street,branch\_num,branch\_city;

END WHILE;

CLOSE branchCurs;

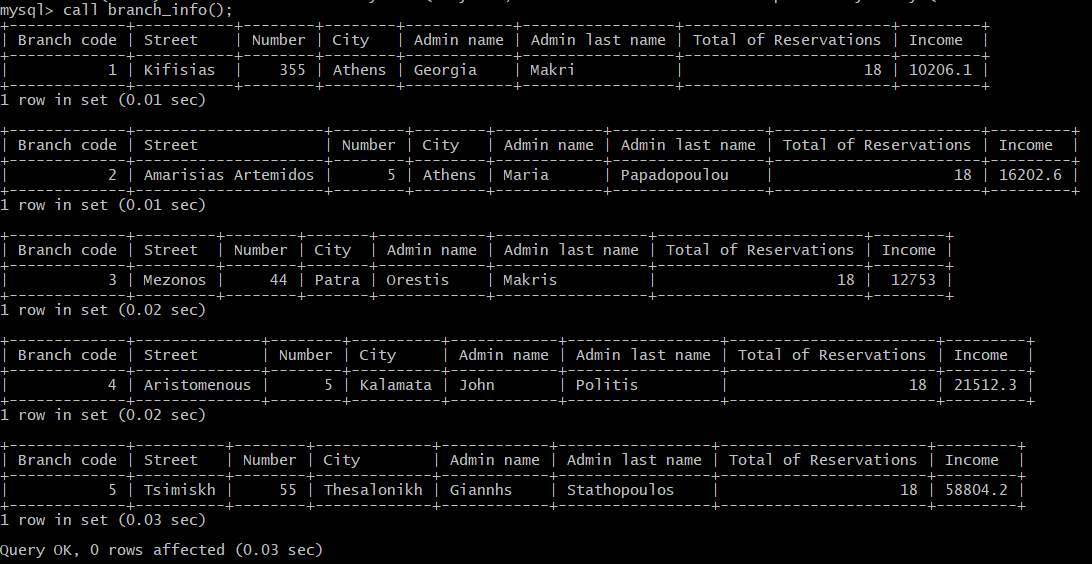
END$

DELIMITER ;

Το **procedure** αυτό δεν παίρνει κάποιο όρισμα. Δημιουργούμε έναν **cursor** με τον οποίο διατρέχουμε τον πίνακα **branch** και αποθηκεύουμε τις ζητούμενες πληροφορίες στις τοπικές μεταβλητές που δηλώσαμε και στο τέλος τις εκτυπώνουμε.

CALL branch\_info();

Καλούμε τον branch\_info με την παραπάνω εντολή.

****

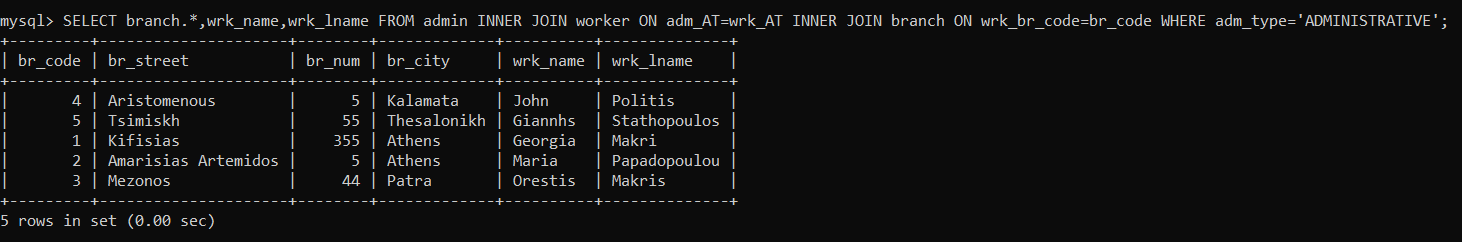
*Έλεγχος:*

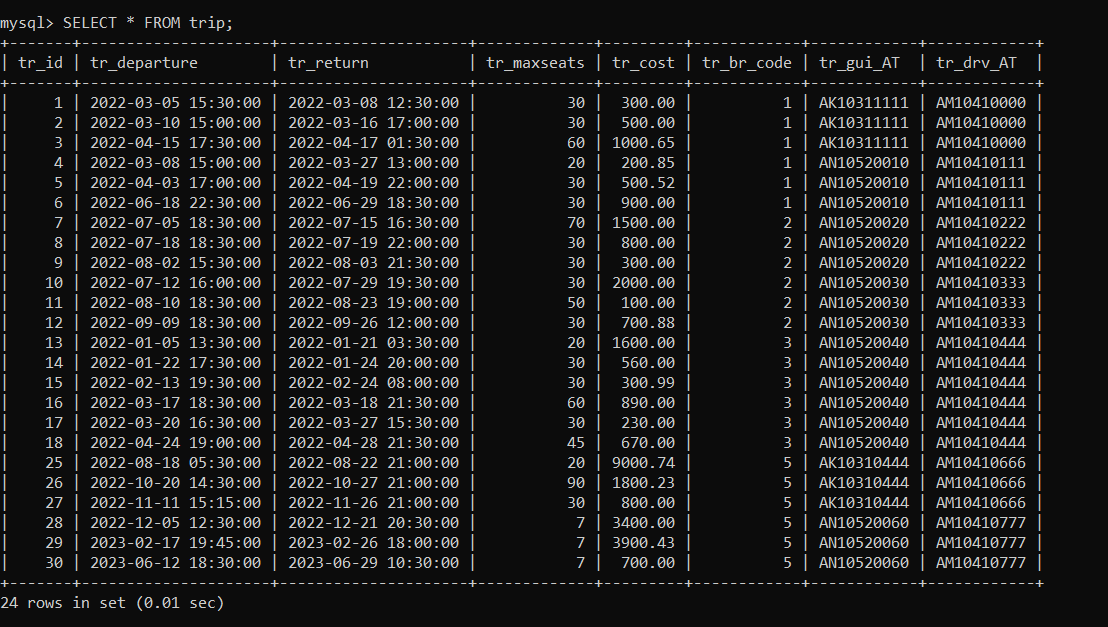
Κάνουμε τα παρακάτω τρία **select** για να ελέγξουμε ότι λειτουργεί σωστά το procedure.

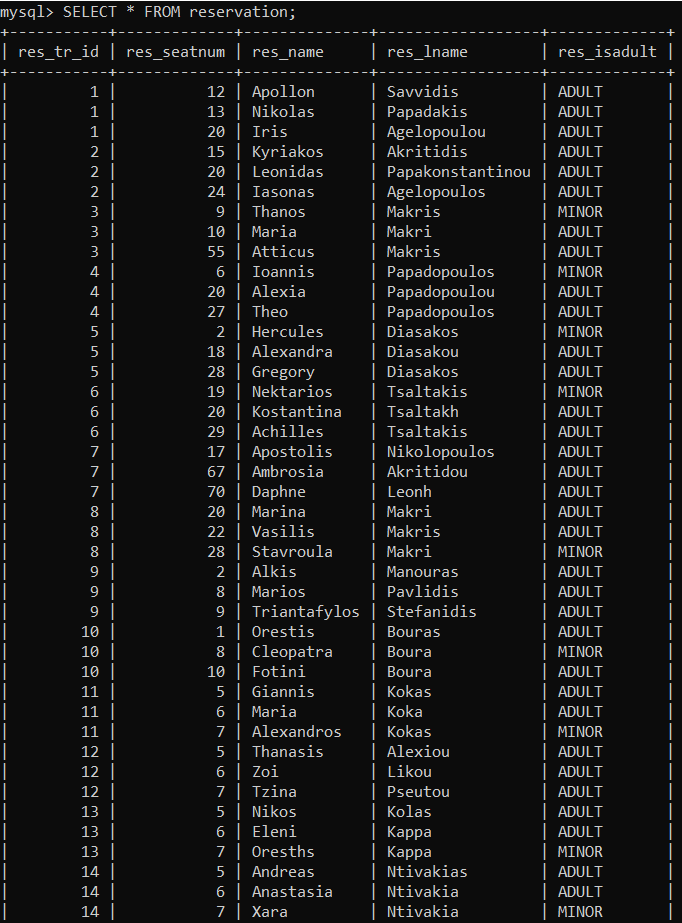
SELECT branch.\*,wrk\_name,wrk\_lname FROM admin INNER JOIN worker ON adm\_AT=wrk\_AT INNER JOIN branch ON wrk\_br\_code=br\_code WHERE adm\_type='ADMINISTRATIVE';

SELECT \* FROM trip;

SELECT \* FROM reservation;







Όπως φαίνεται το procedure λειτουργεί σωστά.

# Κεφάλαιο 3 - Triggers

## **3.1.4.1**

Δημιουργήσαμε triggers για την καταγραφή **εισαγωγής**, **ενημέρωσης** ή **διαγραφής** σε έναν από τους πίνακες **trip**, **reservation**, **event**, **travel\_to**, **destination**. Για κάθε πίνακα από αυτούς δημιουργούμε τρία triggers για όλες τις πιθανές δράσεις και μέσα σε αυτά εισάγουμε μια νέα εγγραφή στην οποία αποτυπώνεται η αλλαγή που κάναμε καθώς επίσης και ο χρήστης που την εκτέλεσε μαζί με την ώρα και ημερομηνία που την εκτέλεσε.

/\*triggers gia ton log\_trip\*/

DROP TRIGGER IF EXISTS log\_trip\_insert;

DELIMITER $

CREATE TRIGGER log\_trip\_insert AFTER INSERT ON trip

FOR EACH ROW

BEGIN

    INSERT INTO log\_trip VALUES

    ('INSERT',CURRENT\_USER(),NOW(),NEW.tr\_id,NEW.tr\_departure,NEW.tr\_return,

    NEW.tr\_maxseats,NEW.tr\_cost,NEW.tr\_br\_code,NEW.tr\_gui\_AT,NEW.tr\_drv\_AT);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_trip\_update;

DELIMITER $

CREATE TRIGGER log\_trip\_update AFTER UPDATE ON trip

FOR EACH ROW

BEGIN

    INSERT INTO log\_trip VALUES

    ('UPDATE',CURRENT\_USER(),NOW(),OLD.tr\_id,OLD.tr\_departure,OLD.tr\_return,

    OLD.tr\_maxseats,OLD.tr\_cost,OLD.tr\_br\_code,OLD.tr\_gui\_AT,OLD.tr\_drv\_AT);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_trip\_delete;

DELIMITER $

CREATE TRIGGER log\_trip\_delete AFTER DELETE ON trip

FOR EACH ROW

BEGIN

    INSERT INTO log\_trip VALUES

    ('DELETE',CURRENT\_USER(),NOW(),OLD.tr\_id,OLD.tr\_departure,OLD.tr\_return,

    OLD.tr\_maxseats,OLD.tr\_cost,OLD.tr\_br\_code,OLD.tr\_gui\_AT,OLD.tr\_drv\_AT);

END$

DELIMITER ;

UPDATE trip SET tr\_departure = '2069-03-05 15:30:06'

WHERE tr\_id = 33;

DELETE FROM trip where tr\_id = 33;

/\*triggers gia ton log\_reservation\*/

DROP TRIGGER IF EXISTS log\_reservation\_insert;

DELIMITER $

CREATE TRIGGER log\_reservation\_insert AFTER INSERT ON reservation

FOR EACH ROW

BEGIN

    INSERT INTO log\_reservation VALUES

    ('INSERT',CURRENT\_USER(),NOW(),NEW.res\_tr\_id,NEW.res\_seatnum,

    NEW.res\_name,NEW.res\_lname,NEW.res\_isadult);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_reservation\_update;

DELIMITER $

CREATE TRIGGER log\_reservation\_update AFTER UPDATE ON reservation

FOR EACH ROW

BEGIN

    INSERT INTO log\_reservation VALUES

    ('UPDATE',CURRENT\_USER(),NOW(),OLD.res\_tr\_id,OLD.res\_seatnum,

    OLD.res\_name,OLD.res\_lname,OLD.res\_isadult);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_reservation\_delete;

DELIMITER $

CREATE TRIGGER log\_reservation\_delete AFTER DELETE ON reservation

FOR EACH ROW

BEGIN

    INSERT INTO log\_reservation VALUES

    ('DELETE',CURRENT\_USER(),NOW(),OLD.res\_tr\_id,OLD.res\_seatnum,

    OLD.res\_name,OLD.res\_lname,OLD.res\_isadult);

END$

DELIMITER ;

/\*triggers gia ton log\_event\*/

DROP TRIGGER IF EXISTS log\_event\_insert;

DELIMITER $

CREATE TRIGGER log\_event\_insert AFTER INSERT ON event

FOR EACH ROW

BEGIN

    INSERT INTO log\_event VALUES

    ('INSERT',CURRENT\_USER(),NOW(),NEW.ev\_tr\_id,NEW.ev\_start,

    NEW.ev\_end,NEW.ev\_descr);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_event\_update;

DELIMITER $

CREATE TRIGGER log\_event\_update AFTER UPDATE ON event

FOR EACH ROW

BEGIN

    INSERT INTO log\_event VALUES

    ('UPDATE',CURRENT\_USER(),NOW(),OLD.ev\_tr\_id,OLD.ev\_start,

    OLD.ev\_end,OLD.ev\_descr);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_event\_delete;

DELIMITER $

CREATE TRIGGER log\_event\_delete AFTER DELETE ON event

FOR EACH ROW

BEGIN

    INSERT INTO log\_event VALUES

    ('DELETE',CURRENT\_USER(),NOW(),OLD.ev\_tr\_id,OLD.ev\_start,

    OLD.ev\_end,OLD.ev\_descr);

END$

DELIMITER ;

/\*triggers gia ton log\_travel\_to\*/

DROP TRIGGER IF EXISTS log\_travel\_to\_insert;

DELIMITER $

CREATE TRIGGER log\_travel\_to\_insert AFTER INSERT ON travel\_to

FOR EACH ROW

BEGIN

    INSERT INTO log\_travel\_to VALUES

    ('INSERT',CURRENT\_USER(),NOW(),NEW.to\_tr\_id,NEW.to\_dst\_id,

    NEW.to\_arrival,NEW.to\_departure);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_travel\_to\_update;

DELIMITER $

CREATE TRIGGER log\_travel\_to\_update AFTER UPDATE ON travel\_to

FOR EACH ROW

BEGIN

    INSERT INTO log\_travel\_to VALUES

    ('UPDATE',CURRENT\_USER(),NOW(),OLD.to\_tr\_id,OLD.to\_dst\_id,

    OLD.to\_arrival,OLD.to\_departure);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_travel\_to\_delete;

DELIMITER $

CREATE TRIGGER log\_travel\_to\_delete AFTER DELETE ON travel\_to

FOR EACH ROW

BEGIN

    INSERT INTO log\_travel\_to VALUES

    ('DELETE',CURRENT\_USER(),NOW(),OLD.to\_tr\_id,OLD.to\_dst\_id,

    OLD.to\_arrival,OLD.to\_departure);

END$

DELIMITER ;

/\*triggers gia ton log\_destination\*/

DROP TRIGGER IF EXISTS log\_destination\_insert;

DELIMITER $

CREATE TRIGGER log\_destination\_insert AFTER INSERT ON destination

FOR EACH ROW

BEGIN

    INSERT INTO log\_destination VALUES

    ('INSERT',CURRENT\_USER(),NOW(),NEW.dst\_id,NEW.dst\_name,NEW.dst\_dscr,

    NEW.dst\_rtype,NEW.dst\_language,NEW.dst\_location);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_destination\_update;

DELIMITER $

CREATE TRIGGER log\_destination\_update AFTER UPDATE ON destination

FOR EACH ROW

BEGIN

    INSERT INTO log\_destination VALUES

    ('UPDATE',CURRENT\_USER(),NOW(),OLD.dst\_id,OLD.dst\_name,OLD.dst\_dscr,

    OLD.dst\_rtype,OLD.dst\_language,OLD.dst\_location);

END$

DELIMITER ;

DROP TRIGGER IF EXISTS log\_destination\_delete;

DELIMITER $

CREATE TRIGGER log\_destination\_delete AFTER DELETE ON destination

FOR EACH ROW

BEGIN

    INSERT INTO log\_destination VALUES

    ('DELETE',CURRENT\_USER(),NOW(),OLD.dst\_id,OLD.dst\_name,OLD.dst\_dscr,

    OLD.dst\_rtype,OLD.dst\_language,OLD.dst\_location);

END$

DELIMITER ;

*Έλεγχος*

Θα ελέγξουμε την λειτουργία των triggers εκτελώντας ένα insert, update και delete στον πίνακα trips.

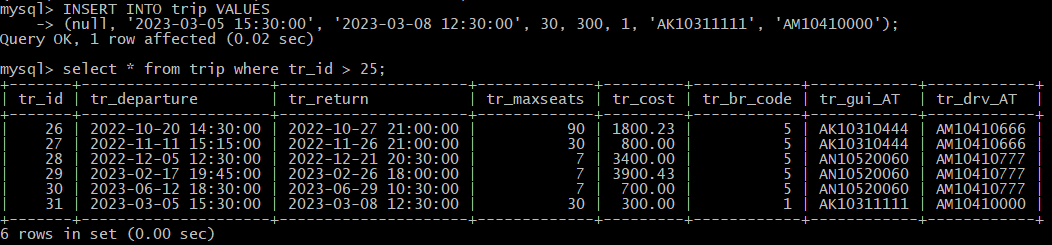
1. **Αρχικά εισάγουμε ένα νέο trip**

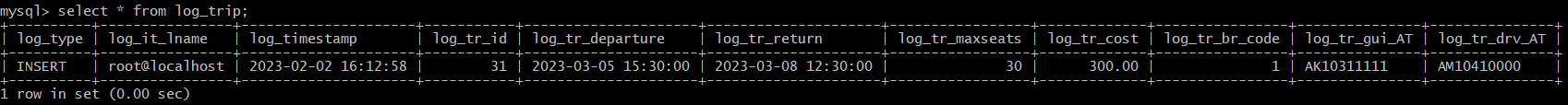
Ο πίνακας trip πριν την εισαγωγή:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Μετά την εισαγωγή:



Έλεγχος του πίνακα **log\_trip**:

Βλέπουμε πως στον πίνακα log\_trip αποθηκεύτηκε μια εγγραφή στην οποία καταγράφονται το είδος της ενέργειας (INSERT), το username του χρήστη που την εκτέλεσε (root) και τα στοιχεία της εγγραφής που εισάχθηκε στον πίνακα trip.

1. **Στην συνέχεια ενημερώνουμε ένα trip**

Έστω ότι αλλάζουμε την ημερομηνία αναχώρησης του καινούριου trip.

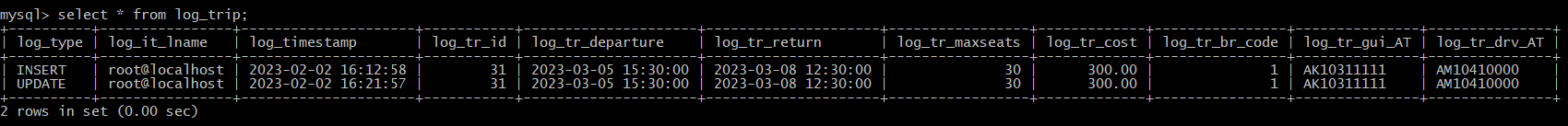
Εικόνα που περιέχει κείμενο, πορτοκαλί

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Βλέπουμε στην τελευταία γραμμή την ενημερωμένη ημερομηνία.

Έλεγχος του πίνακα **log\_trip**:

Βλέπουμε πως εκτελέστηκε ένα UPDATE και αποθηκεύτηκε η ημερομηνία αναχώρησης που είχε το ταξίδι πριν την ενημέρωση.

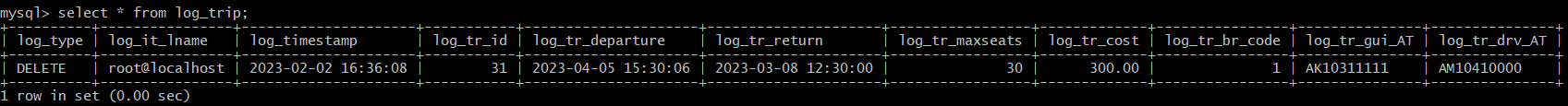
1. **Τέλος διαγράφουμε ένα trip**

****

**Εικόνα που περιέχει κείμενο, ηλεκτρονικές συσκευές, υπολογιστής, πίνακας αποτελεσμάτων

Περιγραφή που δημιουργήθηκε αυτόματα**

Διαγράψαμε το trip με id = 31.

Έλεγχος του πίνακα **log\_trip**:

Βλέπουμε πως εκτελέστηκε ένα DELETE και αποθηκεύτηκαν τα πεδία του ταξιδιού που διαγράφηκε.

Με αντίστοιχο τρόπο δουλεύουν και τα triggers των υπολοίπων πινάκων.

## **3.1.4.2**

DROP TRIGGER IF EXISTS check\_reservations;

DELIMITER $

CREATE TRIGGER check\_reservations BEFORE UPDATE ON trip

FOR EACH ROW

BEGIN

IF exists(SELECT \* FROM reservation WHERE res\_tr\_id=OLD.tr\_id)

THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT='This trip already has reservations therefore updates are restricted!';

END IF;

END$

DELIMITER ;

Το **trigger** αυτό αποτρέπει την αλλαγή της ημερομηνίας αναχώρησης,της ημερομηνίας επιστροφής και του κόστους του ταξιδιού, αν έχουν ήδη γίνει κρατήσεις γι’ αυτό.Ο έλεχγος γίνεται με ένα **if** που αν ισχύει τότε εκτυπώνεται κατάλληλο error.

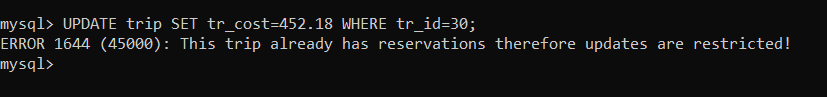
SELECT \* FROM reservation;

SELECT \* FROM trip;

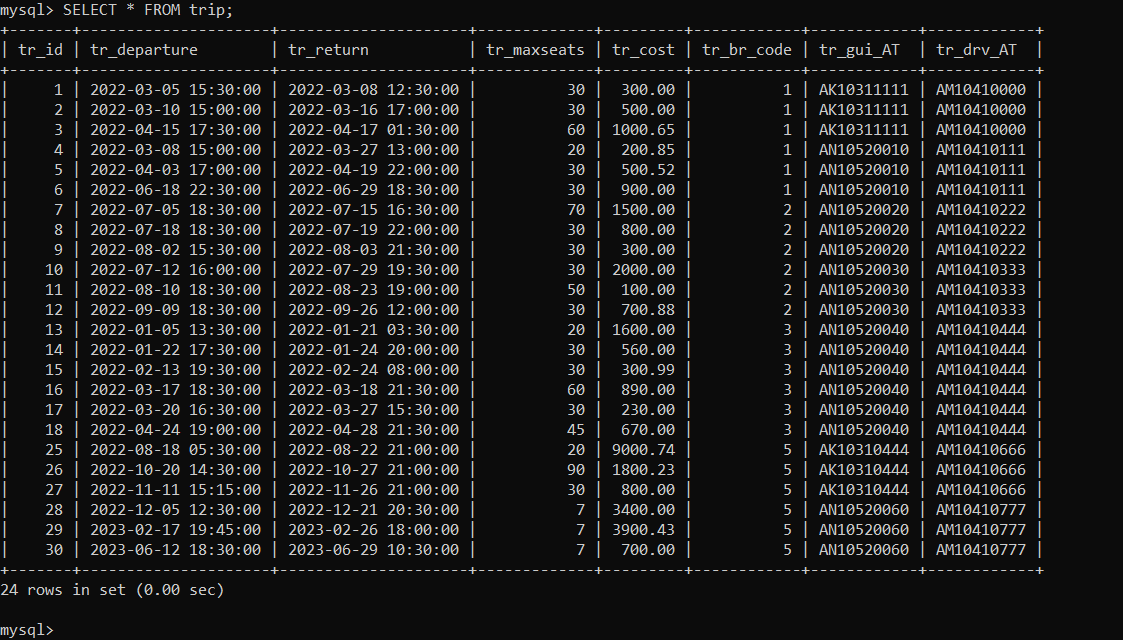
Εκτελούμε τα δύο παραπάνω **select** για να δούμε ποια **trips** έχουν κρατήσεις και ποια όχι.

UPDATE trip SET tr\_cost=452.18 WHERE tr\_id=30;

Στη συνέχεια Εκτελούμε το παραπάνω **update**.



Εκτελούμε το **select** για το **trip** και βλέπουμε αν έχει γίνει αλλαγή.



Όπως το περιμέναμε δεν έγινε κάποια αλλαγή.

## **3.1.4.3**

DROP TRIGGER IF EXISTS check\_salary;

DELIMITER $

CREATE TRIGGER check\_salary BEFORE UPDATE ON worker

FOR EACH ROW

BEGIN

IF OLD.wrk\_salary>=NEW.wrk\_salary

THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT='The new salary must be greater than the older!';

END IF;

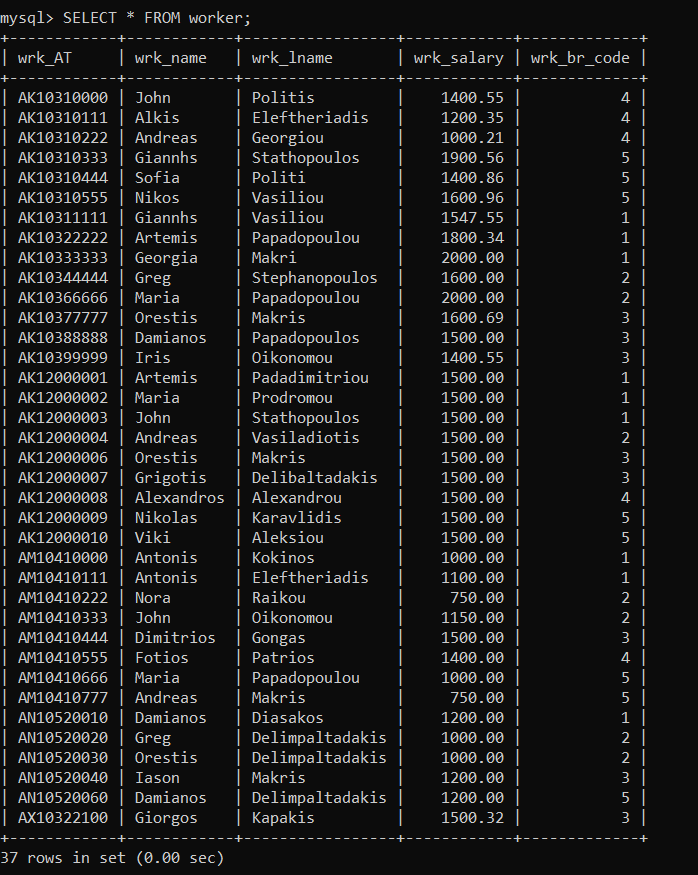
END$

DELIMITER ;

Το **trigger** αυτό δεν επιτρέπει τη μείωση του μισθού ενός υπαλλήλου.Με ένα if ελέχγουμε αν ο παλιός μισθός είναι μεγαλύτερος ή ίσος από τον καινούριο και είναι τότε εκτυπώνεται ένα error.

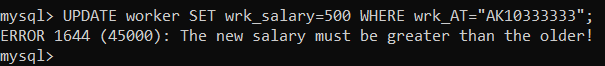
SELECT \* FROM worker;

Εκτελούμε πρώτα αυτό το **select** για να διαλέξουμε τον worker στον οποίο θα προσπαθήσουμε να αλλάξουμε τον μισθό.

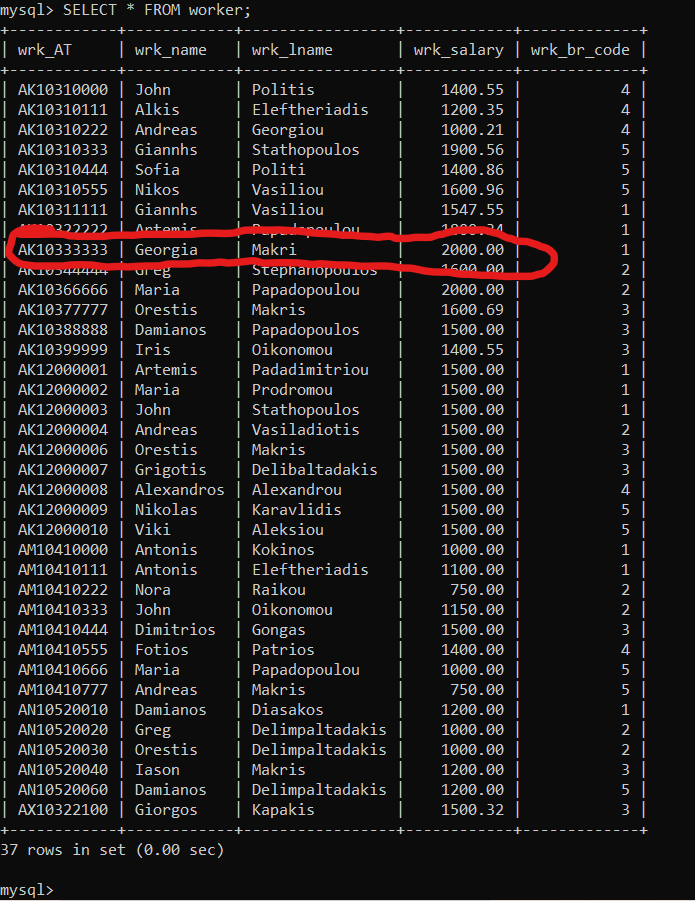


UPDATE worker SET wrk\_salary=500 WHERE wrk\_AT="AK10333333";

Εκτελούμε το παραπάνω **update**.



Εκτελούμε πάλι το **select** με το worker για να δούμε αν έχει γίνει αλλαγή.



Απ’ ότι φαίνεται δεν έχει γίνει αλλαγή.

# Μέρος Β: GUI

# Κεφάλαιο 4 - Java

**Κλάση BranchInfo:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **BranchInfo** παράθυρο θέλουμε να βάλουμε τις πληροφορίες σε ένα **table** για κάθε υποκατάστημα τα στοιχεία του, το ονοματεπώνυμο του διευθυντή του, το σύνολο κρατήσεων και το σύνολο εσόδων (κράτηση επί κόστος ταξιδιού) και σε ένα άλλο **table** για κάθε υποκατάστημα το όνομα, επώνυμο και μισθό όλων των υπάλληλων του και το συνολικό ποσό μισθών που πληρώνει.

Επομένως φτιάχνουμε τα δύο **tables** με δύο **buttons** δίπλα από το καθένα.Και τα δύο **tables** έχουν ένα **button** **‘’Clear’’** που ‘’καθαρίζει’’ τους πίνακες και ένα **button** **‘’Get branch info’’** και **‘’Get branch employees’’** αντίστοιχα που βάζουνε τις αντίστοιχες πληροφορίες που εξηγήσαμε παραπάνω.

Η μέθοδος **branch\_info()** εκτελεί στην βάση την εντολή **call branch\_info()** την οποία έχουμε εξηγήσει στα **Επιπλέον Procedures** και παίρνει τα αποτελέσματα της βάσης και τα βάζει στο ένα **table**.

Η μέθοδος **branch\_employees()** εκτελεί στην βάση την εντολή:

SELECT br\_code,wrk\_name,wrk\_lname,wrk\_salary,(SELECT SUM(wrk\_salary) from worker WHERE wrk\_br\_code=br\_code) AS 'Total salaries' FROM worker INNER JOIN branch ON br\_code=wrk\_br\_code;

Η παραπάνω εντολή εμφανίζει τις πληροφορίες που έχουμε εξηγήσει παραπάνω. Έτσι, παίρνουμε τα αποτελέσματα της βάσης και τα βάζουμε στο άλλο **table**.

**Κλάση Log\_Destination:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **Log\_Destination** παράθυρο θέλουμε να βλέπουμε όλες τις πληροφορίες που έχουν καταγραφεί στον πίνακα **log\_destination** της βάσης μας, ο οποίος περιέχει το είδος της ενέργειας, το επώνυμο **Υπευθύνου Πληροφορικής** που έκανε την ενέργεια, το timestamp και τις πληροφορίες της παλιάς ή καινούργιας γραμμής του πίνακα **destination** ανάλογα αν έχουμε κάνει **update** ή **delete** και **insert** αντίστοιχα.

Η μέθοδος **table\_load()** εκτελεί στην βάση την εντολή **SELECT \* FROM log\_destination** η οποία εμφανίζει όλες τις πληροφορίες του συγκεκριμένου πίνακα και τις βάζει σε ένα **table** του παραθύρου.

**Κλάση Log\_Event:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **Log\_Event** παράθυρο θέλουμε να βλέπουμε όλες τις πληροφορίες που έχουν καταγραφεί στον πίνακα **log\_event** της βάσης μας, ο οποίος περιέχει το είδος της ενέργειας, το επώνυμο **Υπευθύνου Πληροφορικής** που έκανε την ενέργεια, το timestamp και τις πληροφορίες της παλιάς ή καινούργιας γραμμής του πίνακα **event** ανάλογα αν έχουμε κάνει **update** ή **delete** και **insert** αντίστοιχα.

Η μέθοδος **table\_load()** εκτελεί στην βάση την εντολή **SELECT \* FROM log\_event** η οποία εμφανίζει όλες τις πληροφορίες του συγκεκριμένου πίνακα και τις βάζει σε ένα **table** του παραθύρου.

**Κλάση Log\_Reservation:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **Log\_Reservation** παράθυρο θέλουμε να βλέπουμε όλες τις πληροφορίες που έχουν καταγραφεί στον πίνακα **log\_reservation** της βάσης μας, ο οποίος περιέχει το είδος της ενέργειας, το επώνυμο **Υπευθύνου Πληροφορικής** που έκανε την ενέργεια, το timestamp και τις πληροφορίες της παλιάς ή καινούργιας γραμμής του πίνακα **reservation** ανάλογα αν έχουμε κάνει **update** ή **delete** και **insert** αντίστοιχα.

Η μέθοδος **table\_load()** εκτελεί στην βάση την εντολή **SELECT \* FROM log\_reservation** η οποία εμφανίζει όλες τις πληροφορίες του συγκεκριμένου πίνακα και τις βάζει σε ένα **table** του παραθύρου.

**Κλάση Log\_Travel\_To:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **Log\_Travel\_To** παράθυρο θέλουμε να βλέπουμε όλες τις πληροφορίες που έχουν καταγραφεί στον πίνακα **log\_travel\_to** της βάσης μας, ο οποίος περιέχει το είδος της ενέργειας, το επώνυμο **Υπευθύνου Πληροφορικής** που έκανε την ενέργεια, το timestamp και τις πληροφορίες της παλιάς ή καινούργιας γραμμής του πίνακα **travel\_to** ανάλογα αν έχουμε κάνει **update** ή **delete** και **insert** αντίστοιχα.

Η μέθοδος **table\_load()** εκτελεί στην βάση την εντολή:

SELECT log\_type , log\_it\_lname , log\_timestamp  , log\_to\_tr\_id ,log\_to\_dst\_id,log\_to\_arrival,log\_to\_departure FROM log\_travel\_to

Η παραπάνω εντολή εμφανίζει όλες τις πληροφορίες του συγκεκριμένου πίνακα και τις βάζει σε ένα **table** του παραθύρου.

**Κλάση Log\_Trip:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **Log\_Trip** παράθυρο θέλουμε να βλέπουμε όλες τις πληροφορίες που έχουν καταγραφεί στον πίνακα **log\_trip** της βάσης μας, ο οποίος περιέχει το είδος της ενέργειας, το επώνυμο **Υπευθύνου Πληροφορικής** που έκανε την ενέργεια, το timestamp και τις πληροφορίες της παλιάς ή καινούργιας γραμμής του πίνακα **trip** ανάλογα αν έχουμε κάνει **update** ή **delete** και **insert** αντίστοιχα.

Η μέθοδος **table\_load()** εκτελεί στην βάση την εντολή:

SELECT log\_type , log\_it\_lname , log\_timestamp  , log\_tr\_id ,log\_tr\_departure,log\_tr\_return,log\_tr\_maxseats,log\_tr\_cost,log\_tr\_br\_code, log\_tr\_gui\_AT ,log\_tr\_drv\_AT FROM log\_trip

Η παραπάνω εντολή εμφανίζει όλες τις πληροφορίες του συγκεκριμένου πίνακα και τις βάζει σε ένα **table** του παραθύρου.

**Κλάση ProcedureResutls:**

Έχει την ακόλουθη οπτική μορφή:

(Εικόνα)

Στο **ProcedureResutls** παράθυρο για κάθε υποκατάστημα και για ένα διάστημα που ορίζει ο **Υπεύθυνος Πληροφορικής** (**ΙΤ**), θέλουμε να βάλουμε το κόστος ταξιδιού, τις μέγιστες θέσεις ανά ταξίδι, το σύνολο κρατήσεων (**reservation**), τις διαθέσιμες κενές θέσεις, το επώνυμο και όνομα οδηγού και ξεναγού, την ημερομηνία αναχώρησης και επιστροφής σε ένα **table** και για ένα επώνυμο ενός πελάτη που ορίζει και πάλι ο ίδιος, θέλουμε να βάλουμε τα ονόματα και επώνυμα των πελατών με το επώνυμο αυτό και την προσφορά ταξιδιού στην οποία έχει γίνει εγγραφή σε ένα άλλο **table**. Σε περίπτωση που υπάρχουν πάνω από ένας επιστρέφει και το πλήθος των πελατών με το όνομα αυτό, ανά προσφορά ταξιδιού και χρησιμοποιεί και τα δύο **tables**.

Η μέθοδος **call\_search\_trip** εκτελεί στην βάση την εντολή **call search\_trip()** την οποία έχουμε εξηγήσει στο **Κεφάλαιο 2 - Stored Procedures** στο **3.1.3.2** και παίρνει τα αποτελέσματα της βάσης και τα βάζει στο ένα **table**.

Η μέθοδος **res\_names** εκτελεί στην βάση την εντολή **call res\_names()** την οποία έχουμε εξηγήσει στο **Κεφάλαιο 2 - Stored Procedures** στο **3.1.3.4** στο **β)** και παίρνει τα αποτελέσματα της βάσης και τα βάζει στο ένα **table** ή και στα δύο **tables** ανάλογα με την περίπτωση.

# Κεφάλαιο 5