Α’ Μέρος

1.

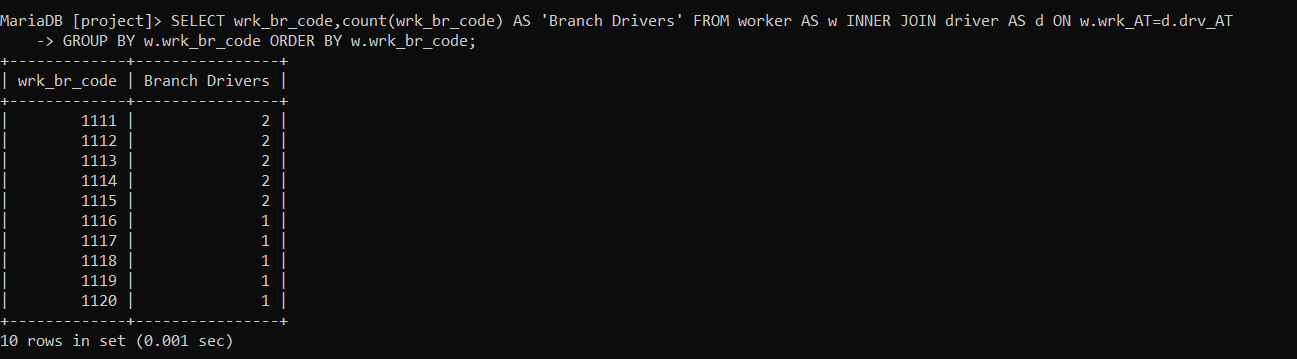
Παραδοχές:

* Στον πίνακα reservation κάθε άτομο μπορεί να κάνει μία κράτηση για τον ίδιο. Η μεταβλητή res\_seatnum αναφέρεται στον αριθμό της θέσης και όχι στον αριθμό των κρατήσεων που έχει γίνει από κάποιο άτομο. Έτσι, η μεταβλητή παίρνει αυξανόμενες τιμές από το 1 έως και το tr\_maxseats (αυτό ελέγχεται μέσω της διεπαφής).
* Στον πίνακα admin θεωρούμε ότι κάθε καταχωρημένος worker με type=’ADMINISTRATIVE’ θα διοικεί κάποιο κατάστημα, λογική με την οποία χειριζόμαστε και τα UPDATE statements στο GUI. Δηλαδή δεν μπορεί να αλλάξει type κάποιος manager, αφού αλλάζοντας τον τύπο του παύει να είναι και manager.
* Στην procedure α) του ερωτήματος 3.1.3.4, προσθέσαμε επίσης στην έξοδο και τον αριθμό κράτησης για καλύτερη αναγνωσιμότητα.
* Στα ζητούμενα της procedure β) του ερωτήματος 3.1.3.4 και στο 5 bullet του ερωτήματος 3.2.1, υπάρχει μια μικρή ασυνέπεια ως προς τη ζητούμενη έξοδο σε περίπτωση που υπάρχουν πάνω από ένα πελάτης με το δοσμένο επίθετο. Επιλέξαμε η έξοδος, στην περίπτωση που αναφέραμε, να ακολουθεί τη λογική του 5ου bullet, καθώς αυτό μας φάνηκε πιο λογικό και κατανοητό για τον χρήστη.
* Στον πίνακα it\_logs προσθέσαμε ως πρωτεύων κλειδί το γνώρισμα log\_id. Δεν καταγράφουμε το επώνυμο αυτού που έκανε την ενέργεια αλλά το ΑΤ του, καθώς μπορεί να υπάρχει κάποια συνωνυμία. Το γνώρισμα IT\_id (το ΑΤ δηλαδή) δεν αποτελεί ξένο κλειδί στον πίνακα worker ή IT, ώστε να μπορούμε να διατηρούμε τις καταγραφές ενεργειών ακόμα και για IT’s οι οποίοι διαγράφηκαν από το σύστημα. (Δοκιμάσαμε να το διατηρήσουμε ως ξένο κλειδί αλλά μας έβγαζε συνεχώς errors).
* Δημιουργήσαμε δύο πίνακες, έναν reservation\_offers και έναν reservation\_offer\_memory, όπου ο πρώτος λειτουργεί με ENGINE=InnoDB και ο δεύτερος με ENGINE=Memory. Η εξήγηση για αυτή την παραδοχή δίνεται στα προβλήματα που συναντήσαμε και πως τα επιλύσαμε για το ερώτημα 3.1.3.4.
* Δεν χρειάστηκε να αλλάξουμε τη δομή κάποιου από τους πίνακες που δίνονταν στο σχεσιακό μοντέλο της εκφώνησης και ο μόνος (επιπλέων) πίνακας που προστέθηκε για την υλοποίηση των βασικών ζητούμενων ήταν ο reservation\_offers\_memory. Για την υλοποίηση του ερωτήματος 3.2.3 προστέθηκαν δύο ακόμα πίνακες ο misconducts και blacklist. Σε αυτούς θα αναφερθούμε στο Β’ Μέρος της αναφοράς.
* Στο Β’ Μέρος, αναφέρεται πως ο IT θα μπορεί να εισάγει, διαγράφει και να τροποποιεί δεδομένα σε όλους τους πίνακες. Θεωρήσαμε λογικό, για τον πίνακα IT\_logs να έχει μόνο τη δυνατότητα να τον διαβάζει και να μην μπορεί να παρέμβει σε αυτόν με κανένα τρόπο, καθώς διαφορετικά δεν θα είχε και ιδιαίτερο νόημα η καταγραφή πράξεων που μπορούν να τροποποιηθούν.
* Στη δυνατότητα των ΙΤ να εισάγουν νέο υπεύθυνο πληροφορικής, δεν χρησιμοποιούμε dropdown λίστα για να πάρουμε τα ονόματα των worker’s που δεν έχουν κάποια ιδιότητα (π.χ. admin, driver,…), καθώς θέλουμε με την εισαγωγή του νέου ΙΤ να τον τοποθετήσουμε στο ίδιο κατάστημα με τον ΙΤ που τον εισάγει. Στη συνέχεια κάνουμε τους απαραίτητους ελέγχους και στο τέλος αν δεν υπάρχει κάποιο σφάλμα προσθέτουμε το νέο ΙΤ πρώτα στον πίνακα worker και έπειτα στον πίνακα IT. Αναλυτικότερη εξήγηση σχετικά με το πώς γίνονται αυτά δίνονται στο Β’ Μέρος της αναφοράς.
* Για το GUI, δεν ζητείται κάπου ξεκάθαρα η χρήση της procedure β) του ερωτήματος 3.1.3.4 και για αυτό έχει προστεθεί ως επιπλέον λειτουργικότητα στο ερώτημα 3.2.3.

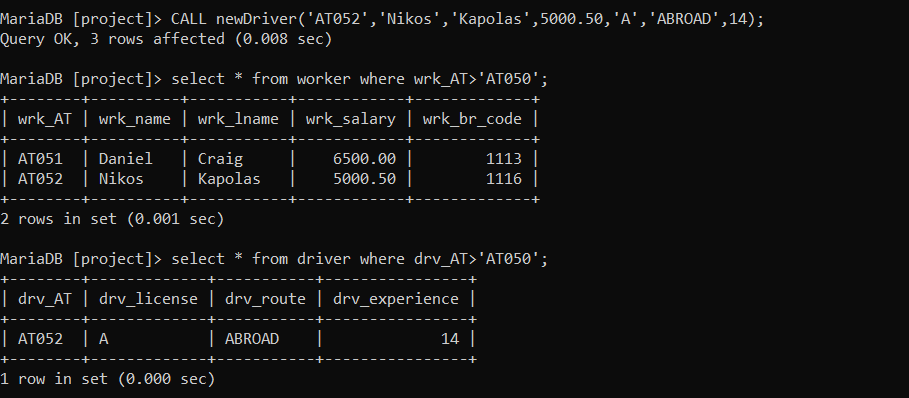
2.

3.1.3.1. Η procedure newDriver(), παίρνει ως ορίσματα τα στοιχεία του νέου οδηγού και εισάγει τα στοιχεία του νέου οδηγού στον πίνακα worker και driver, όπου το wrk\_br\_code του νέου οδηγού παίρνει την τιμή του καταστήματος με τους λιγότερους οδηγούς.

Με τα στοιχεία που έχουμε εισάγει ήδη, το πρώτο κατάστημα με τους λιγότερους οδηγούς είναι αυτό με br\_code=1116.

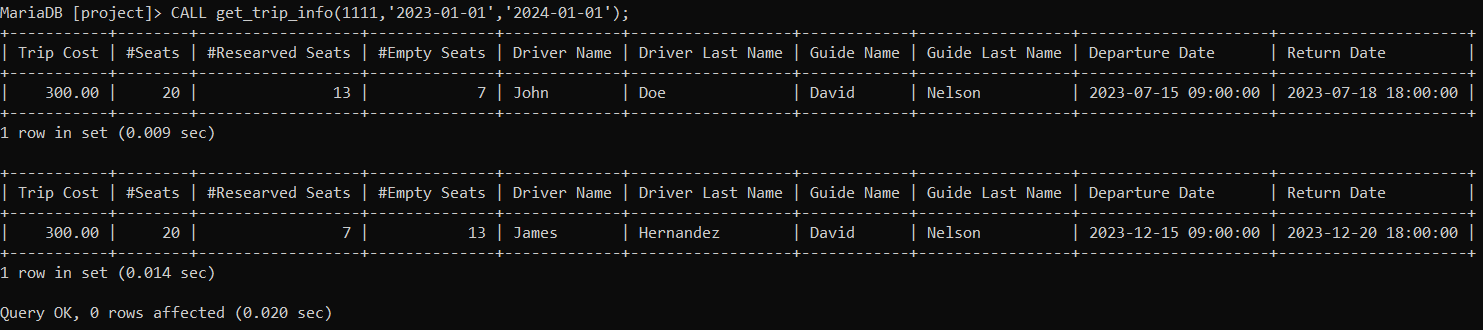


Καλούμε την procedure newDriver() για νέο οδηγό και βλέπουμε ότι εισάγει το νέο οδηγό στον πίνακα worker με wrk\_br\_code=1116, δηλαδή στο πρώτο κατάστημα με τους λιγότερους οδηγούς και στον πίνακα driver:



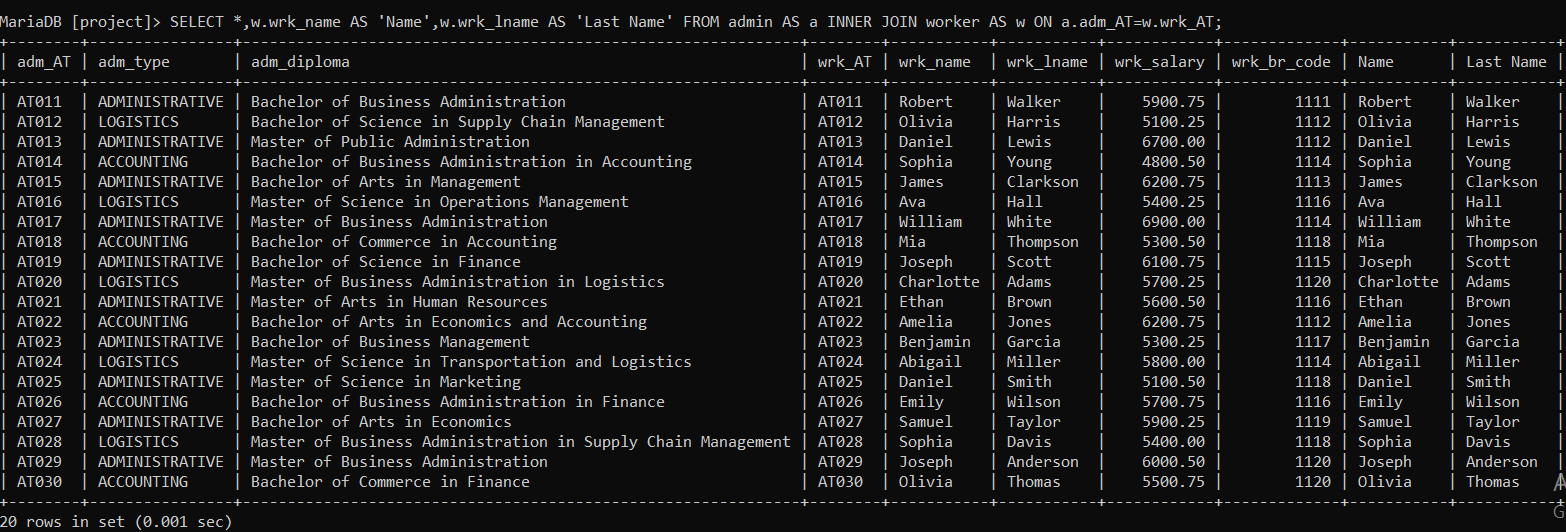
3.1.3.2. Η procedure get\_trip\_info() παίρνει ως ορίσματα έναν κωδικό καταστήματος και δύο ημερομηνίες και επιστρέφει πληροφορίες για τα ταξίδια που διοργανώνει το κατάστημα, με το δοσμένο κωδικό, ανάμεσα από τις δύο ημερομηνίες.

Καλούμε την procedure get\_trip\_info():

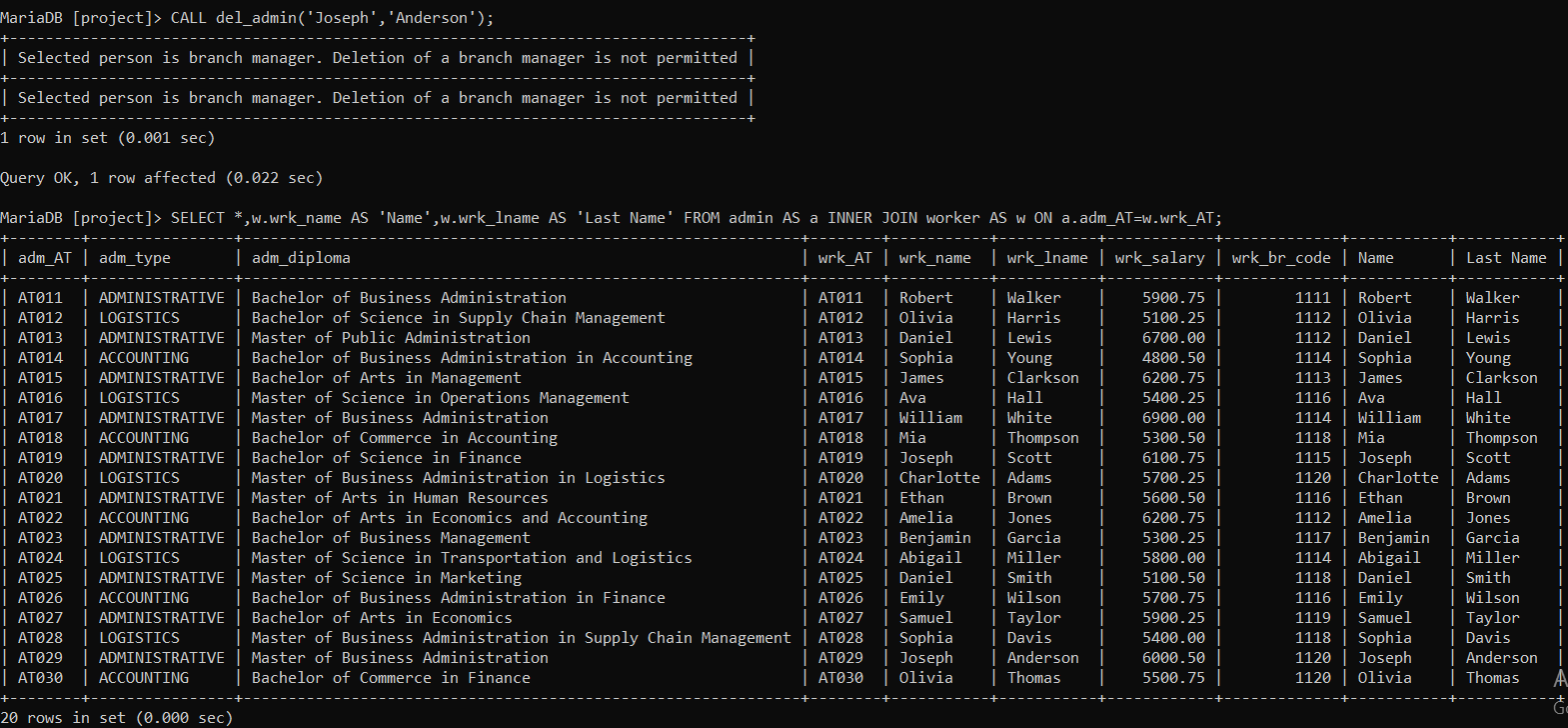


3.1.3.3. Η procedure del\_admin() παίρνει ως ορίσματα το όνομα και το επώνυμο ενός υπαλλήλου και αν ανήκει στο διοικητικό προσωπικό και δεν είναι διευθυντής κάποιου υποκαταστήματος, τον διαγράφει.

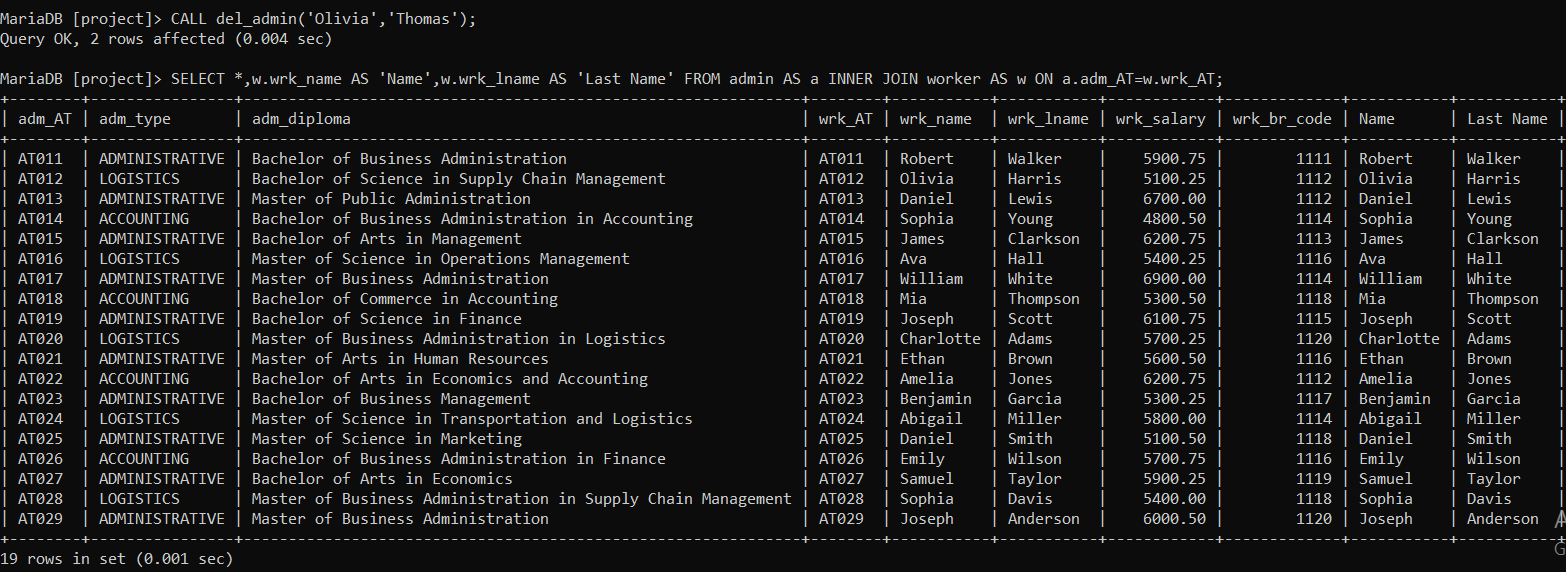
Οι εργαζόμενοι που ανήκουν στο διοικητικό προσωπικό είναι οι παρακάτω:



Καλούμε την del\_admin() για τον “Joseph Anderson” (προ τελευταίος στη λίστα), μας εμφανίζεται μήνυμα άρνησης διαγραφής διευθυντή καταστήματος και ο υπάλληλος δεν έχει διαγραφεί:



Καλούμε ξανά την del\_admin() για την “Olivia Thomas” (τελευταία στην λίστα) και αφού δεν είναι διευθύντρια κάποιου καταστήματος, διαγράφεται:



3.1.3.4

Στο ερώτημα 3.1.3.4, καλούμαστε να δημιουργήσουμε 2 stored procedures, οι οποίες θα εκτελούν ουσιαστικά μία αναζήτηση στον πίνακα reservation\_offers με βάση κάποιας(-ων) τιμών, είτε εύρους είτε ταιριάσματος (αλφαριθμητικού). Για να επιταχύνουμε την αναζήτηση των 2 αυτών procedures, θα πρέπει να δημιουργήσουμε από ένα κατάλληλο ευρετήριο για την καθεμία.

Για την πρώτη procedure, αφού έχουμε ερώτημα εύρους, επιλέγουμε να χρησιμοποιήσουμε BTREE index.

Για τη δεύτερη procedure, αφού έχουμε ερωτήματα ισότητας , επιλέγουμε να χρησιμοποιήσουμε HASH index, κάτι που έχει ως αποτέλεσμα να χρειαστεί να αλλάξουμε το ENGINE του πίνακα από το DEFAULT InnoDB σε Memory, καθώς το πρώτο δεν υποστηρίζει HASH type indexes. Επιλέξαμε να χρησιμοποιήσουμε δυο πίνακες, έναν με ονομασία reservation\_offers και ENGINE=InnoDB και ένα δεύτερο πίνακα με ονομασία reservation\_offers\_memory με ENGINE=Memory. Ο λόγος για τον οποίο το κάναμε αυτό θα αναφερθεί παρακάτω, μαζί με τα προβλήματα που αντιμετωπίσαμε για αυτό το μέρος

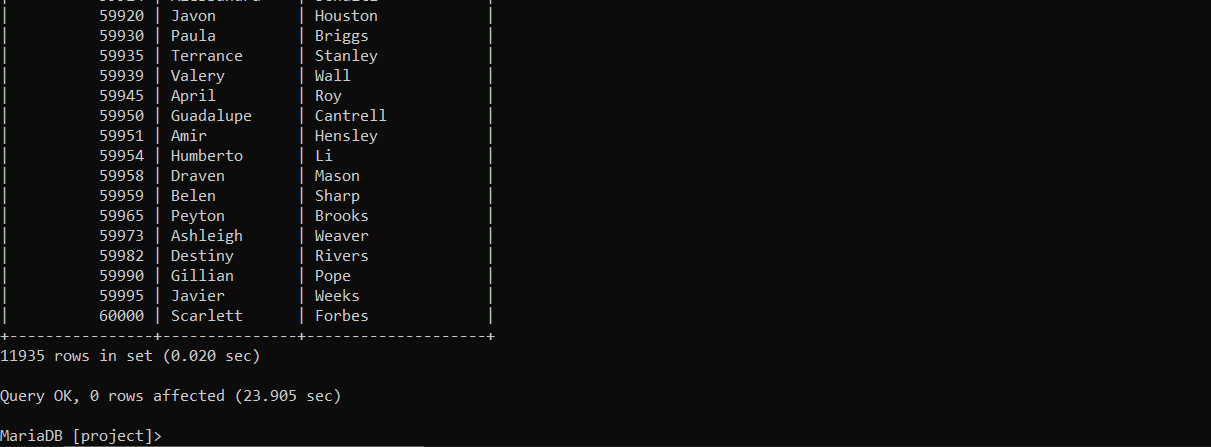
Παρακάτω, παραθέτουμε πρώτα τα αποτελέσματα από τις αναζητήσεις με/χωρίς indexes, και έπειτα τα προβλήματα που έπρεπε να επιλυθούν όταν κάναμε την αλλαγή από InnoDB σε Memory ENGINE στον πίνακα reservation\_offers\_memory.

α)

Για την πρώτη procedure, αφού η αναζήτηση που εκτελούμε μέσα σε αυτή, γίνεται με βάση το γνώρισμα advance\_fee, θα δημιουργήσουμε index για αυτό το γνώρισμα στον πίνακα reservation\_offers\_memory.

-> CALL get\_Res\_Offers\_Names(50,80);

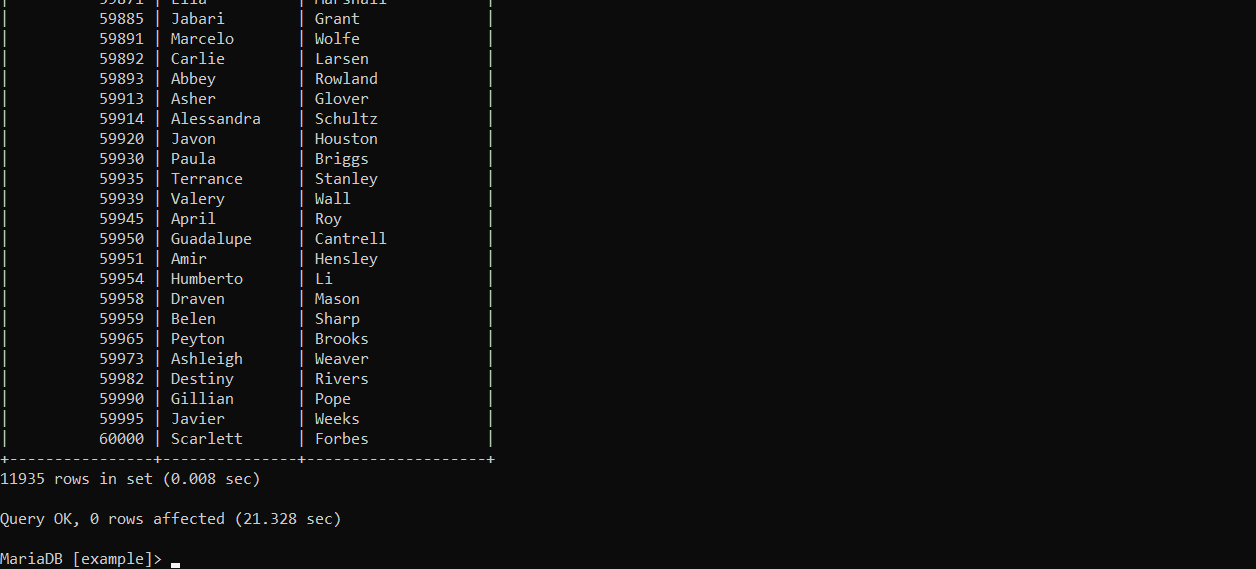
Έξοδος (χωρίς index, παρατίθεται μέρος της εξόδου, λόγω του μεγέθους της):



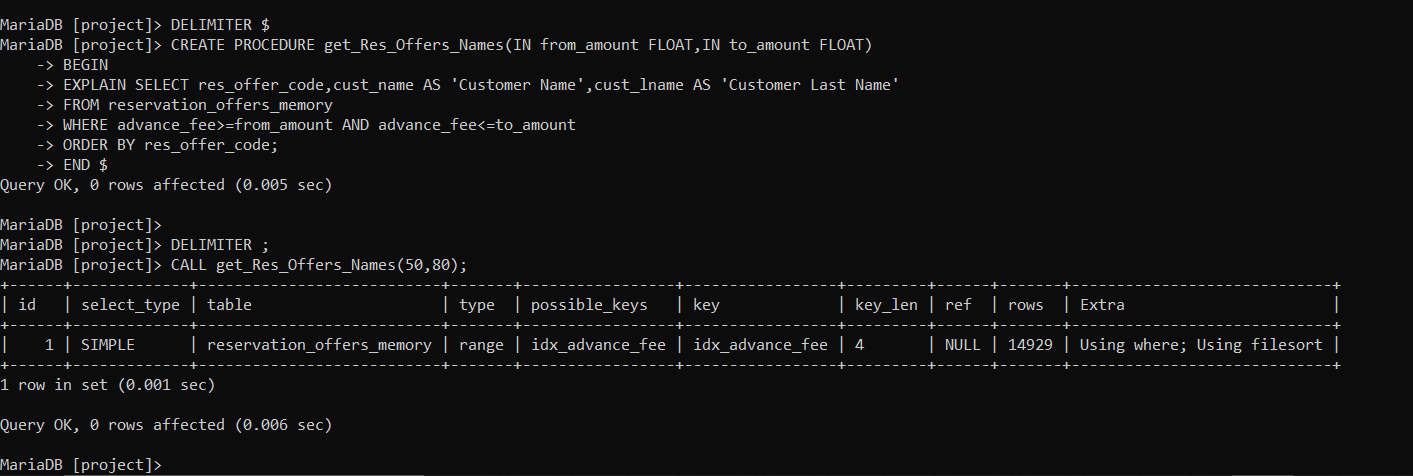
Δημιουργούμε τον index:

-> CREATE INDEX idx\_advance\_fee ON reservation\_offers\_memory(advance\_fee) USING BTREE;

Έξοδος (με index):



Παρατηρούμε μια μικρή μείωση του χρόνου εκτέλεσης αλλά όχι σημαντική (από 0.020sec -> 0.008sec), καθώς ίσως το πλήθος των δεδομένων είναι σχετικά μικρό και δεν μπορεί να φανεί και πολύ μεγάλη διαφορά με ή χωρίς index. Παρόλα αυτά, προσθέτοντας πριν τη ‘SELECT’ εντός της procedure την επιλογή ‘EXPLAIN’ μπορούμε να δούμε ότι κατά την εκτέλεση της εντολής, επιλέγεται να χρησιμοποιηθεί ο index που δηλώσαμε και όχι να προσπελαστεί ολόκληρος ο πίνακας:

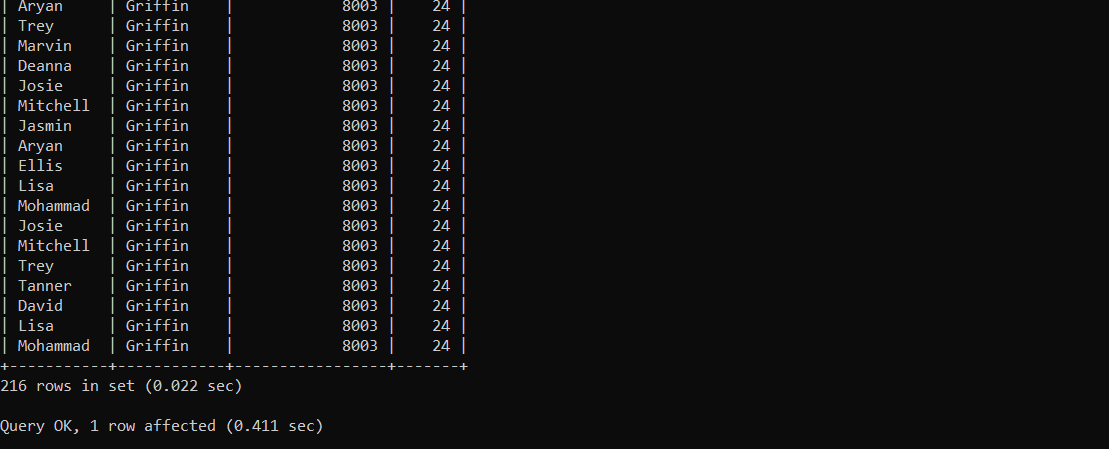


β)

Για την δεύτερη procedure, η αναζήτηση γίνεται με βάση το γνώρισμα cust\_lname, άρα για αυτό θα ορίσουμε τον index στον πίνακα reservation\_offers\_memory.

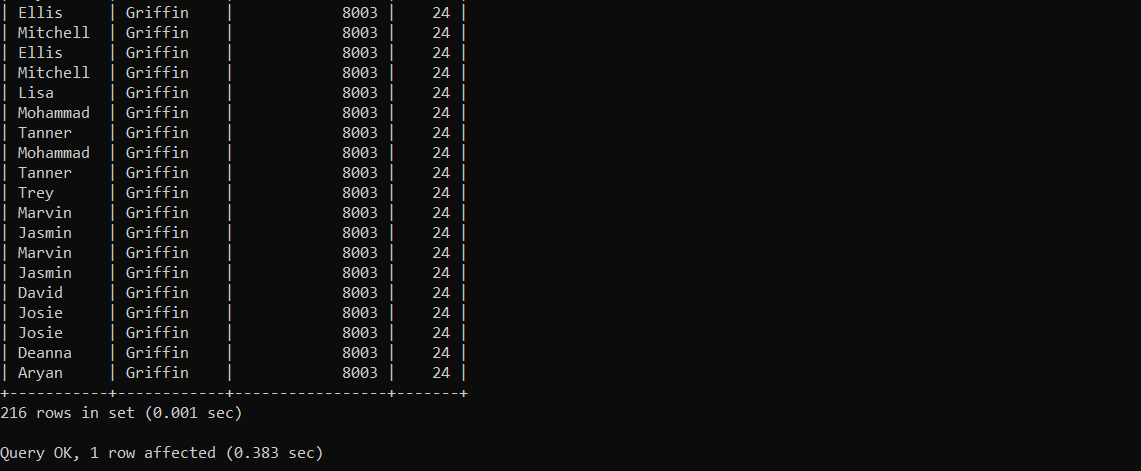
-> CALL get\_Offers\_Participation(‘Griffin’);

Έξοδος (χωρίς index):



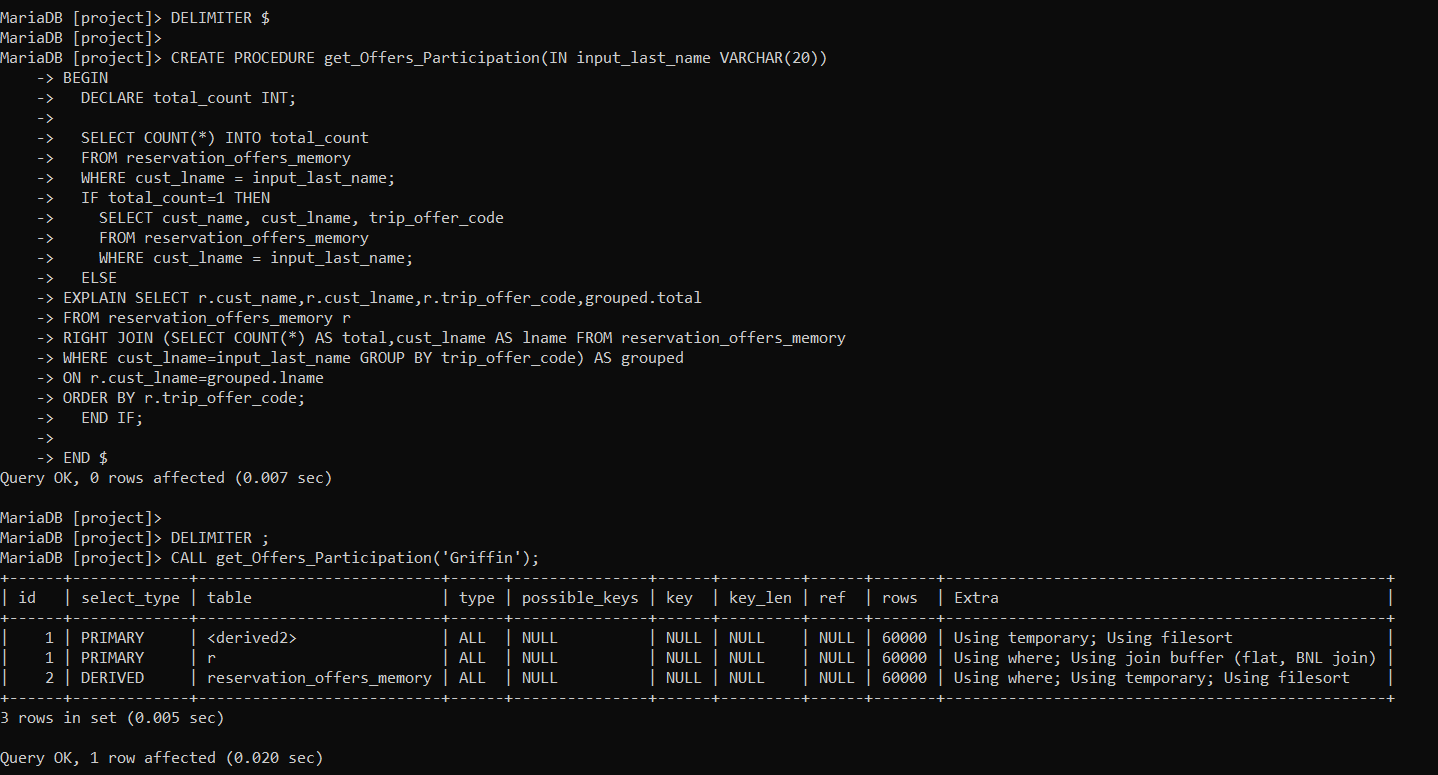
-> CREATE INDEX idx\_cust\_lname ON reservation\_offers\_memory(cust\_lname) USING HASH;

Έξοδος (με index):

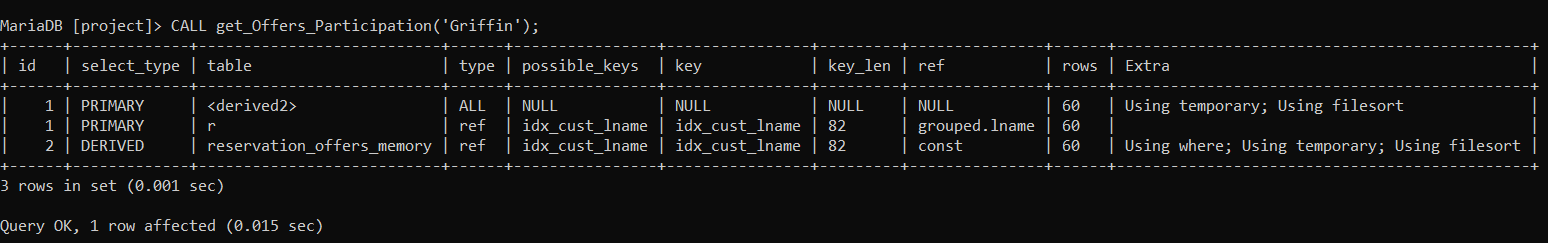


Εδώ παρατηρούμε αρκετά μεγάλη διαφορά στο χρόνο εκτέλεσης (από 0.022sec -> 0.001sec), κάτι που οφείλεται στο γεγονός ότι με χρήση HASH έχουμε ακριβή ταύτιση στην αναζήτηση, άρα δεν χάνουμε ίσως τόσο πολύ χρόνο σε ανισο-ισότητες όπως σε ένα BTREE. Βέβαια και πάλι ο χρόνος εκτέλεσης είναι αμελητέος (τάξης των msec) καθώς ίσως το μέγεθος των εγγραφών είναι σχετικά μικρό.

Και πάλι με χρήση της ‘EXPLAIN’ και χωρίς χρήση index, βλέπουμε ότι προσπελαύνεται ολόκληρος ο πίνακας κατά την αναζήτηση (στήλη rows: 60.000).



Με χρήση της ‘EXPLAIN’ και με χρήση index, βλέπουμε ότι χρησιμοποιείται, στην δεύτερη και τρίτη ‘SELECT’ ο index και ο αριθμός των γραμμών που πρόκειται να προσπελαστούν είναι κατά πολύ μικρότερος (στήλη rows: 60).



Ζητήματα που προέκυψαν με τη χρήση Memory ENGINE:

Αρχικά, καθώς πίνακες με Memory ENGINE χρησιμοποιούν την κύρια μνήμη και όχι το δίσκο για να αποθηκεύσουν τα δεδομένα τους, μετά από κάθε restart στη βάση ξαναδημιουργούνται, καθώς το αρχείο δήλωσής τους βρίσκεται στο δίσκο, αλλά είναι κενοί. Άρα, για να διατηρούμε τα δεδομένα του πίνακα, δημιουργήσαμε ένα πίνακα με InnoDB και ένα με Memory ENGINE, και κάθε φορά που μπαίνουμε στη βάση, απλά αντιγράφουμε τα περιεχόμενα του InnoDB πίνακα στον Memory πίνακα.

Το δεύτερο εμπόδιο που προέκυψε, όταν ορίσαμε το ENGINE = Memory στον πίνακα reservation\_offers\_memory, ήταν το σφάλμα: ERROR 1114 (HY000): The table ‘reservation\_offers’ is full.

Αυτό είχε ως αποτέλεσμα να μην μπορούμε να δημιουργήσουμε indexes για τον πίνακα, σε περίπτωση που είχαμε αποθηκεύσει τις εγγραφές πριν τη δημιουργία των indexes, ή να μην μπορεί να αποθηκεύσει όλες τις εγγραφές που θέλαμε (και τις 60.000).

Αφού είχαμε αρχικοποιήσει τους indexes για τον πίνακα και εισάγαμε σε αυτόν όσα records μπορούσε να πάρει, θέλαμε να δούμε τι χώρο καταλαμβάνουν τα δεδομένα μέχρι στιγμής, ώστε να έχουμε μια εικόνα για το πόση επέκταση θα κάνουμε στο χώρο του πίνακα. Για να δούμε το χώρο που καταλαμβάνει κάθε πίνακας στη βάση δεδομένων μας τρέξαμε την παρακάτω εντολή:

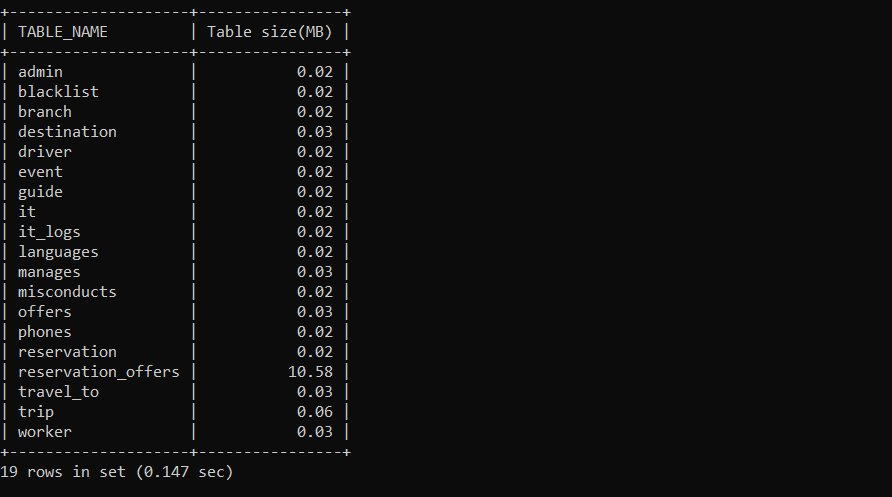
-> SELECT TABLE\_NAME,ROUND(SUM(DATA\_LENGTH + INDEX\_LENGTH) / 1024 / 1024, 2) AS 'Table Size (MB)'

FROM information\_schema.create\_tables

WHERE TABLE\_SCHEMA = 'project'

GROUP BY TABLE\_NAME ORDER BY 'Table Size (MB)' DESC;

Έξοδος:



Προκειμένου να αυξήσουμε τον επιτρεπτό χώρο που μπορεί να καταλαμβάνει κάθε πίνακας με ENGINE = Memory, κάναμε τα εξής βήματα:

1. Ανοίξαμε ένα τερματικό από XAMPP Control Panel και πήγαμε στο φάκελο mysql\bin:

# cd mysql\bin

2. Εκεί βρίσκεται το MariaDB configuration file 'my.ini'. Ανοίγουμε το αρχείο (start notepad my.ini) και προσθέτουμε/τροποποιούμε κάτω από το τμήμα [mysqld] τη μεταβλητή: max\_heap\_table\_size = 64M.

Ορίζουμε ως μέγιστο επιτρεπτό μέγεθος πίνακας δηλαδή τα 64Mb. Η επιλογή των 64Mb είναι λίγο υπερβολική, δεδομένου ότι το ERROR που προκύπτει, λογικά προκύπτει λόγω ανεπάρκειας μερικών Mb, αλλά από

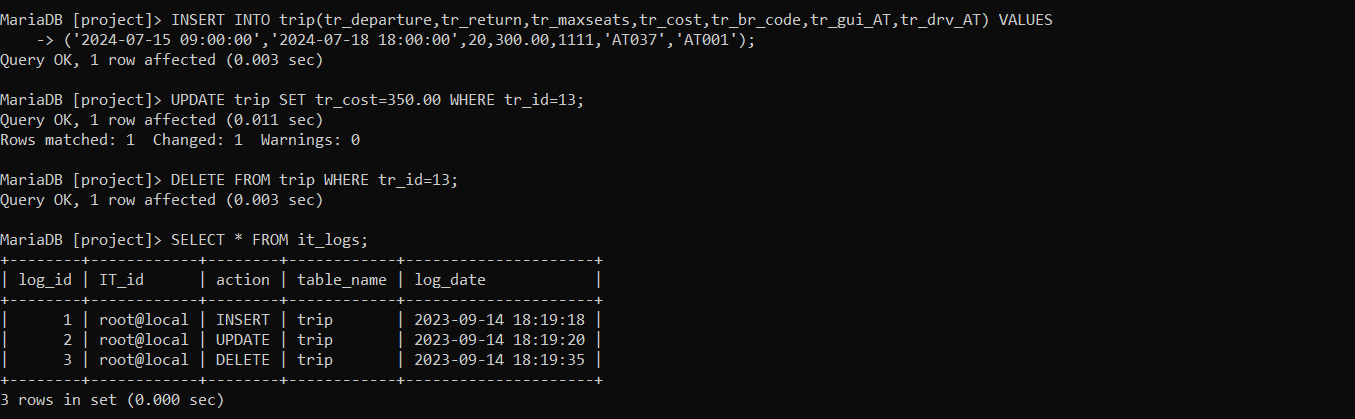
τη στιγμή που δεν υπάρχει κάποια άλλη αυστηρή απαίτηση σε χώρο μπορούμε να κάνουμε αυτή την παραχώρηση.

Μετά από αυτά τα βήματα, ο πίνακας δέχτηκε κανονικά όλες τις εγγραφές που εισάγαμε (και τις 60.000), καθώς και τα indexes.

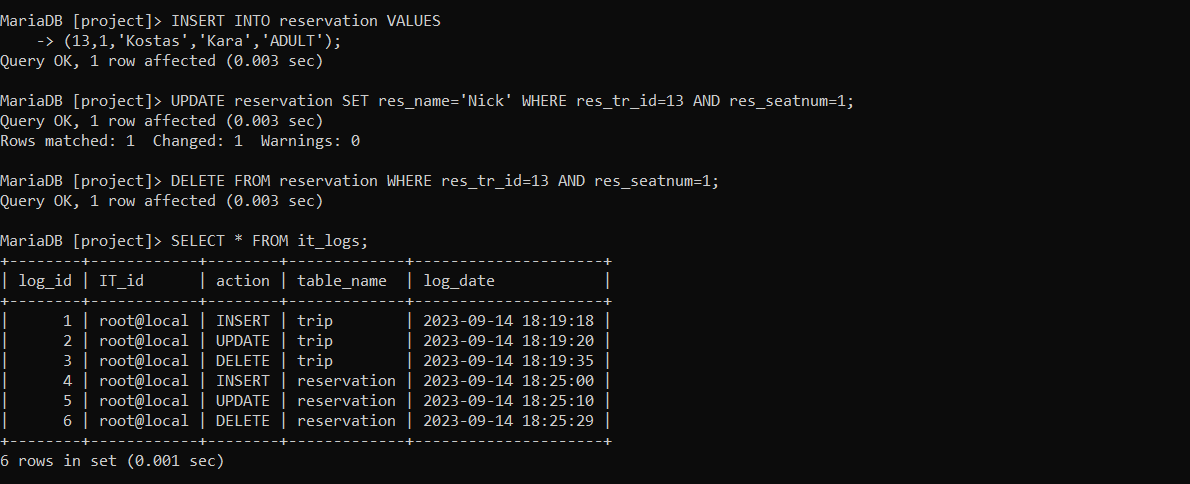
3.

3.1.4.1. Για την καταγραφή των ενεργειών του εκάστοτε χρήστη, όσο αφορά τους πίνακες trip, reservation, event, travel\_to και destination, δημιουργήσαμε 15 triggers, ένα για κάθε ενέργεια (insert, delete, update) για κάθε ένα από τους πίνακες. Όταν μπαίνουμε στη βάση μέσω της διεπαφής, αποθηκεύεται κάθε φορά ποιος χρήστης είναι συνδεδεμένος και καταγράφεται το ΑΤ του στο πεδίο “IT\_id”. Επειδή όμως μέσω της cmd, έχουμε μια default ταυτότητα του root, καταγραφόμαστε ως root@local στο πεδίο “IT\_id” και για αυτό θα φαίνεται αυτή η ταυτότητα στα παραδείγματα που θα δώσουμε.

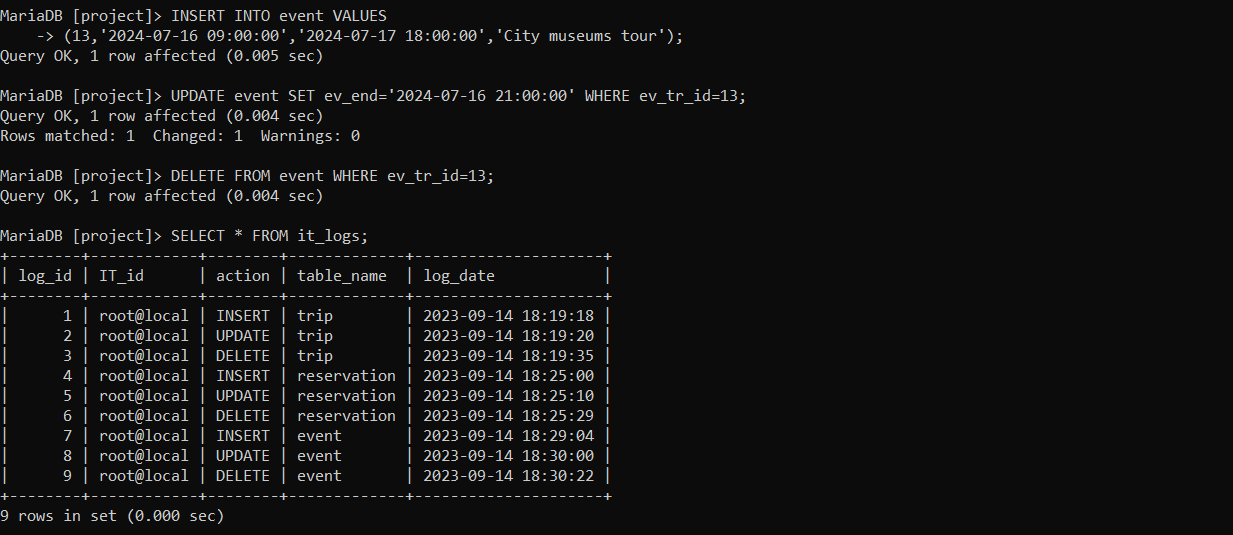
Παράδειγμα insert, update, delete στον πίνακα trip:



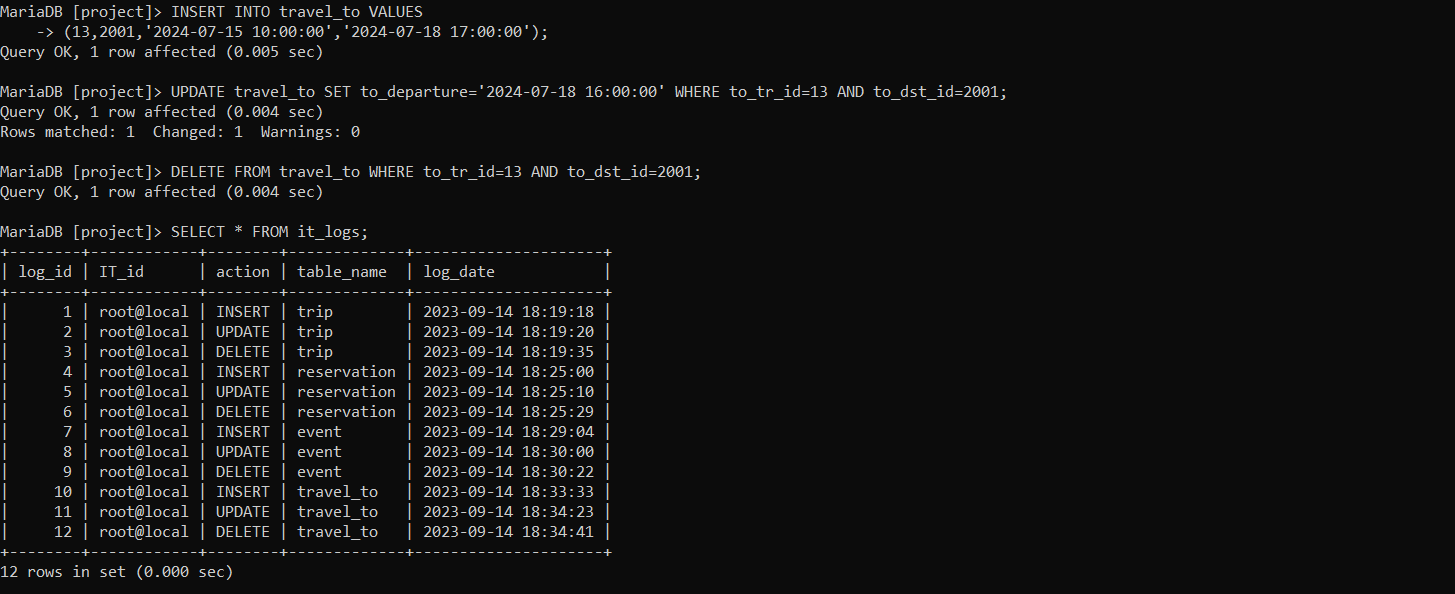
Παράδειγμα insert, update, delete στον πίνακα reservation:



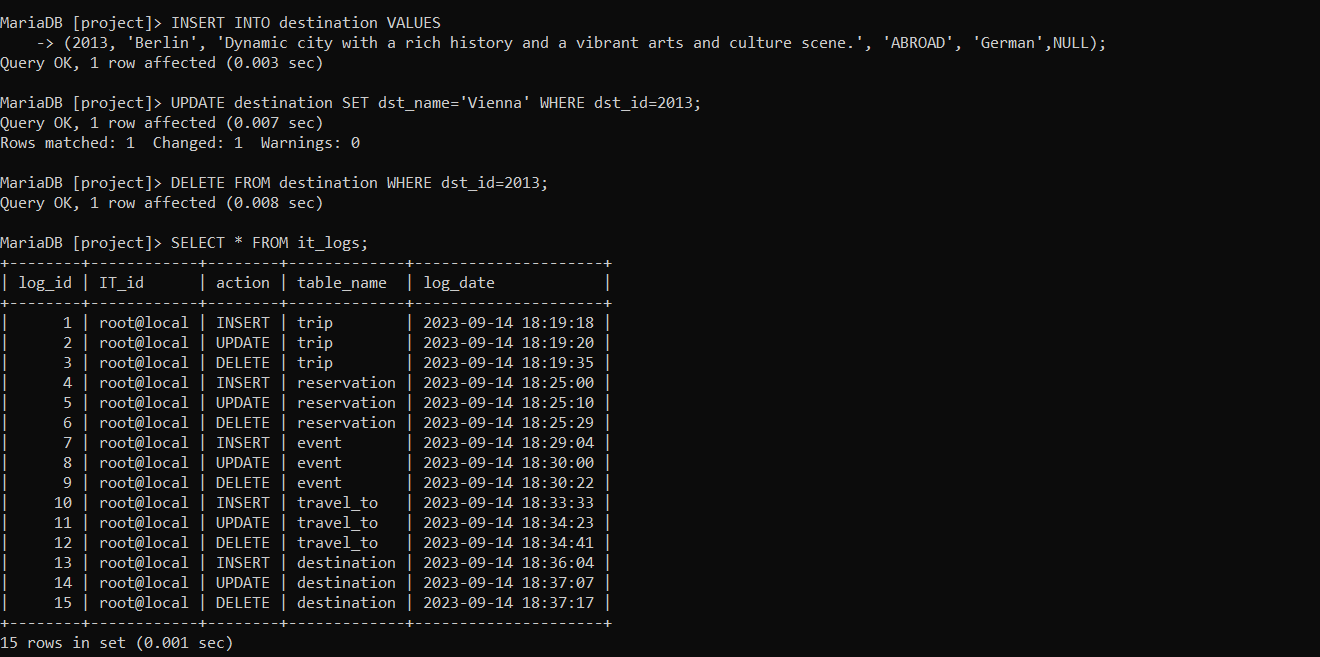
Παράδειγμα insert, update, delete στον πίνακα event:



Παράδειγμα insert, update, delete στον πίνακα travel\_to:



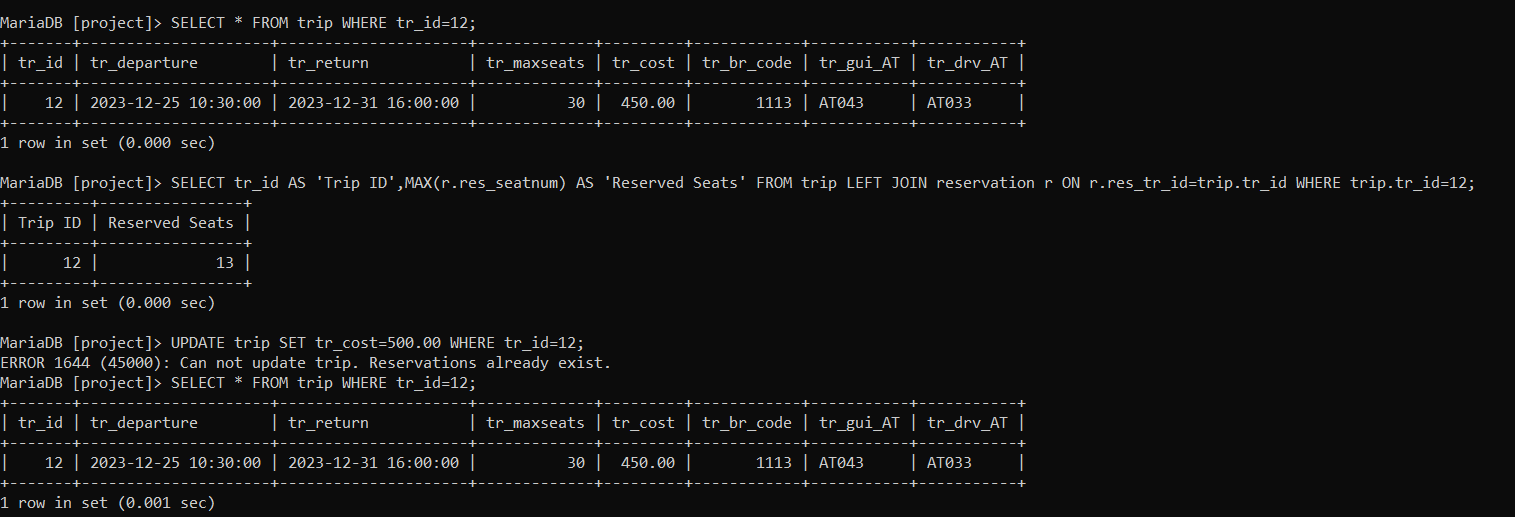
Παράδειγμα insert, update, delete στον πίνακα destination:



3.1.4.3

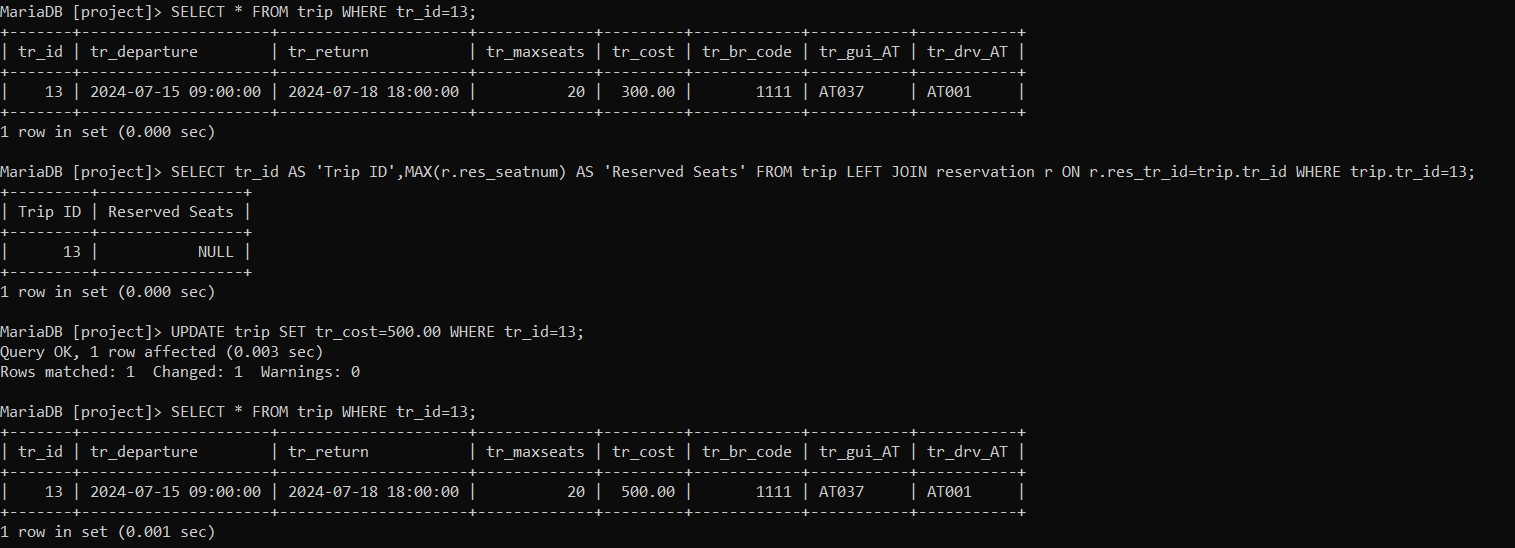
Ο trigger που δημιουργήσαμε ελέγχει αν πριν από μια εντολή update στον πίνακα trip, αν κάποιο από τα πεδία tr\_cost, tr\_departure ή tr\_return παίρνει καινούργια τιμή. Αν ναι, τότε ελέγχει αν υπάρχουν κρατήσεις για αυτό το ταξίδι και αν υπάρχουν αποτρέπει το update και στέλνει κατάλληλο μήνυμα.

Παράδειγμα για ταξίδι με κρατήσεις:



Βλέπουμε πως δεν έγινε κάποια αλλαγή στο record με tr\_id=12.

Παράδειγμα για ταξίδι χωρίς κρατήσεις:

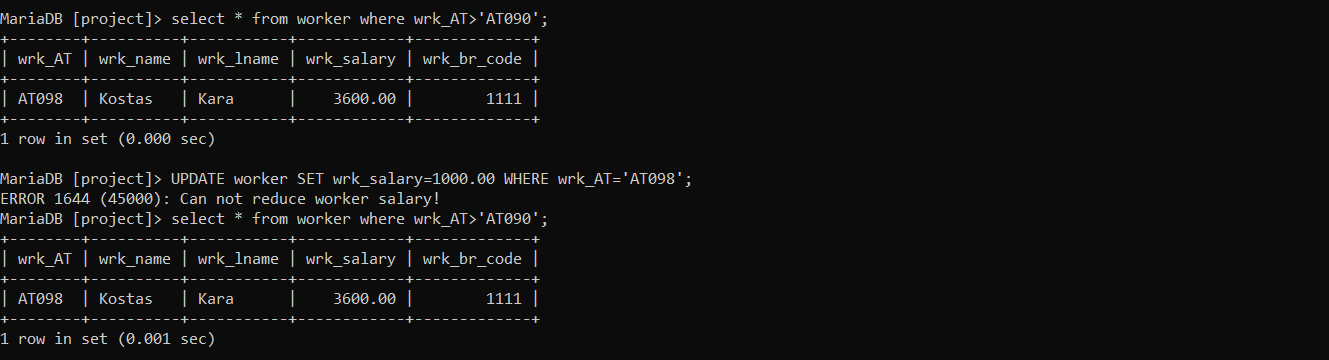


Βλέπουμε πως το κόστος του ταξιδιού με tr\_id=13 άλλαξε από 300.00 στα 500.00.

3.1.4.3

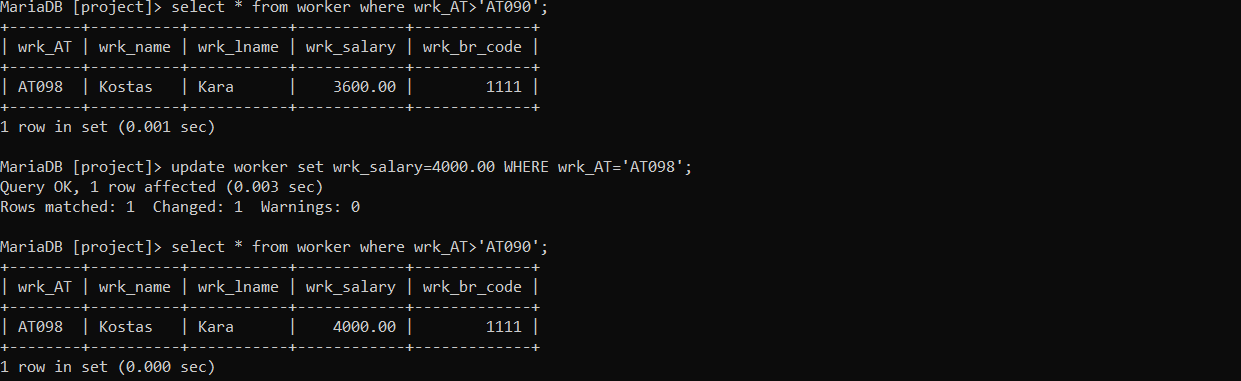
Ο trigger που δημιουργήσαμε ελέγχει αν ο παλιός μισθός, του υπαλλήλου που πρόκειται να γίνει το update, είναι μικρότερος από τον καινούριο. Αν ναι, αποτρέπει την αλλαγή και εμφανίζει κατάλληλο μήνυμα.

Παράδειγμα μείωσης μισθού:



Βλέπουμε ότι δεν άλλαξε ο μισθός του.

Παράδειγμα αύξησης μισθού:



Βλέπουμε ότι ο μισθός του αυξήθηκε.

Β’ Μέρος

3.2.1-3.2.2

Η διεπαφή ξεκινά από την οθόνη σύνδεσης όπου ο χρήστης εισάγει το επώνυμό του ως όνομα χρήστη και τον κωδικό του. Το σύστημα ελέγχει αν υπάρχει IT με αυτά τα στοιχεία στον πίνακα IT και αν ναι τον μεταφέρει στην κεντρική οθόνη, αλλιώς παραμένει στην οθόνη σύνδεσης.

Στην κεντρική οθόνη εμφανίζονται όλα τα κουμπιά με τις διαθέσιμες για το χρήστη λειτουργίες (12 στο σύνολο, 9 από τις βασικές απαιτήσεις και 3 επιπλέον που θα αναλυθούν στο ερώτημα 3.2.3).

Για τις λειτουργίες INSERT,UPDATE,DELETE σε κάθε μία από τις κλάσεις για τους πίνακες trip, reservation, event, travel\_to και destination έγινε χρήση των triggers που δημιουργήθηκαν στο ερώτημα 3.1.4.1.

Για τη λειτουργία INSERT, έχουμε δημιουργήσει μια κλάση για κάθε πίνακα, εκτός των IT και IT\_logs, καθώς για τον ΙΤ υπάρχει ξεχωριστό ερώτημα ( bullet ) και για τον IT\_logs αναφέρουμε στις παραδοχές τη λογική που ακολουθήσαμε. Αφού ο χρήστης επιλέξει σε ποιόν πίνακα θέλει να εισάγει στοιχεία, εμφανίζεται ένα παράθυρο με τα πεδία του αντίστοιχου πίνακα, ο χρήστης τα συμπληρώνει και επιλέγει “Insert”. Σε κάθε κλάση για τον αντίστοιχο πίνακα υπάρχει μια μέθοδος όπου συνδεόμαστε στη βάση δημιουργούμε το INSERT statement με τα στοιχεία από τα πεδία που συμπλήρωσε ο χρήστης παραπάνω και το εκτελούμε. Αν δεν προκύψει κάποιο σφάλμα επιστρέφεται μήνυμα επιτυχίας, αλλιώς το μήνυμα του σφάλματος. Στις κλάσεις όπου υπάρχει κάποιο ξένο κλειδί (π.χ. gui\_AT), υπάρχει και κατάλληλη συνάρτηση η οποία ανακτά από τη βάση τα AT των workers που είναι απλά καταγεγραμμένοι ως workers και τους εμφανίζει ως τις διαθέσιμες επιλογές του χρήστη για το αντίστοιχο πεδίο. Αντίστοιχα, προσθέσαμε dropdown λίστες για τα πεδία όπου απαιτούσαν να συμπληρωθούν με κάποια υπάρχουσα τιμή από τη βάση. Ακόμα, για την εισαγωγή Driver, χρησιμοποιείται η procedure newDriver() του ερωτήματος 3.1.3.1. Τέλος, για τους πίνακες trip, reservation, event, travel\_to και destination, μετά από κάθε INSERT αλλάζουμε το log\_id με το AT του IT που πραγματοποίησε την ενέργεια.

Για τη λειτουργία DELETE, επίσης δημιουργήσαμε μια κλάση για κάθε πίνακα ( εκτός του IT\_logs ). Και πάλι ο χρήστης επιλέγει τον πίνακα από τον οποίο θέλει να διαγράψει κάποιο στοιχείο(-α) και εμφανίζεται ένα παράθυρο με μια dropdown λίστα και το κουμπί διαγραφής. Η λίστα περιέχει το κλειδί(-α) του κάθε πίνακα με βάση το/τα οποίο(-α) θα γίνει η διαγραφή, καθώς και μερικά στοιχεία αυτού. Π.χ. για τη διαγραφή worker, εμφανίζεται το ΑΤ, το όνομα και το επώνυμο του worker και η διαγραφή γίνεται με βάση το ΑΤ. Υπάρχει η δυνατότητα ο χρήστης να αφήσει κενή τη μπάρα επιλογής προκειμένου να διαγράψει ολόκληρο τον πίνακα. Αν γίνει κάτι τέτοιο ο χρήστης ερωτάται πριν ολοκληρωθεί η ενέργεια και υπάρχει δυνατότητα ματαίωσής της. Στις κλάσεις DeleteAdmin και DeleteWorker γίνεται χρήση της procedure del\_admin(), του ερωτήματος 3.1.3.3. Τέλος, για τους πίνακες trip, reservation, event, travel\_to και destination, μετά από κάθε DELETE αλλάζουμε το log\_id με το AT του IT που πραγματοποίησε την ενέργεια.

Για τη λειτουργία UPDATE, επίσης δημιουργήσαμε μια κλάση για κάθε πίνακα ( εκτός του IT\_logs ). Αφού ο χρήστης επιλέξει σε ποιόν πίνακα θέλει να εισάγει στοιχεία, εμφανίζεται ένα παράθυρο με τα πεδία του αντίστοιχου πίνακα, ο χρήστης τα συμπληρώνει και επιλέγει “Update”. Στα πεδία που εμφανίζονται τα πρωτεύοντα κλειδιά(-ι) από κάθε πίνακα εμφανίζονται σε μορφή dropdown λίστας, όπως επίσης και τα πεδία που παίρνουν συγκεκριμένες τιμές ( π.χ. adm\_type ). Οι λίστες δημιουργούνται ανακτώντας τα απαραίτητα στοιχεία μέσα από κάθε πίνακα της βάσης. Στις κλάσεις UpdateTrip και UpdateWorker χρησιμοποιήθηκαν οι triggers privent\_trip\_update (ερώτημα 3.1.4.2) και prevent\_salary\_reduction (ερώτημα 3.1.4.3) αντίστοιχα. Τέλος, για τους πίνακες trip, reservation, event, travel\_to και destination, μετά από κάθε UPDATE αλλάζουμε το log\_id με το AT του IT που πραγματοποίησε την ενέργεια.

Για την δυνατότητα του IT να βλέπει για κάθε υποκατάστημα πληροφορίες για τα ταξίδια που διοργανώνει, εντός ορισμένου χρονικού διαστήματος (4ο bullet), δημιουργήσαμε την κλάση ChooseDatesWindow. Ο χρήστης επιλέγει, στο παράθυρο που εμφανίζεται, δύο ημερομηνίες των οποίων η χρονιά, ο μήνας και η ημέρα επιλέγονται από dropdown λίστες. Έπειτα καλείται μία μέθοδος, της ίδιας της κλάσης, η οποία αρχικά ανακτά όλους τους κωδικούς τον υποκαταστημάτων, και έπειτα για κάθε κωδικό και το ζεύγος των επιλεγμένων ημερομηνιών καλούμε την procedure get\_trip\_info() (ερώτημα 3.1.3.2) και τέλος εμφανίζουμε τα αποτελέσματα σε ένα νέο παράθυρο.

Για την δυνατότητα του ΙΤ να εισάγει το επώνυμο ενός πελάτη και να παίρνει μερικές πληροφορίες για αυτόν από τον πίνακα reservation\_offers (5ο bullet), δημιουργήσαμε την κλάση InsertCustNameWindow. Ο χρήστης εισάγει το επώνυμο με το οποίο θέλει να πραγματοποιήσει αναζήτηση και επιλέγει το κουμπί “Search”. Έπειτα, καλείται μία μέθοδος της ίδιας της κλάσης μέσα από την οποία καλούμε την procedure get\_Offers\_Participation() (ερώτημα 3.1.3.4 β) ). Αν υπάρχουν αποτελέσματα, τα εμφανίζουμε σε ένα νέο παράθυρο, αλλιώς εμφανίζουμε μήνυμα αποτυχίας ταύτισης για το επώνυμο.

Προκειμένου ο ΙΤ να μπορεί να βλέπει πληροφορίες για κάθε υποκατάστημα, όπως το ποιος το διοικεί, έσοδα κλπ. δημιουργήσαμε μία μέθοδο εντός της κλάσης MainAppWindow, η οποία ανακτά μέσω μίας “SELECT” και μερικών “LEFT JOIN”, παίρνουμε τις πληροφορίες για τα καταστήματα από τον πίνακα branch, από τους πίνακες manages, admin και worker εντοπίζουμε και ανακτούμε τις πληροφορίες του υπευθύνου κάθε καταστήματος και τέλος από τον πίνακα trip και ενός “LEFT JOIN” με τον πίνακα max\_seatnums που δημιουργούμε μέσω ενός sub-query, για να πάρουμε τον αριθμό των κρατημένων θέσεων ανά ταξίδι, παίρνουμε τις πληροφορίες που θέλουμε για το σύνολο των κρατήσεων και των εσόδων. Αποθηκεύει σε μία λίστα τις πληροφορίες και έπειτα τις εμφανίζουμε σε ένα νέο παράθυρο.

Για να μπορεί να δει ο ΙΤ πληροφορίες για τους εργαζόμενους κάθε καταστήματος, δημιουργήσαμε ακόμα μία μέθοδο εντός της κλάσης MainAppWindow, η οποία ανακτά μέσω μίας “SELECT” ανακτά τα ονόματα και τους μισθούς των υπαλλήλων κάθε καταστήματος από τον πίνακα worker και μετά με ένα “LEFT JOIN” με τον πίνακα branch\_total\_salary, τον οποίο δημιουργούμε με ένα sub-query, ο οποίος περιέχει το συνολικό μισθό των υπαλλήλων ανά υποκατάστημα, παίρνουμε και την τελευταία απαιτούμενη πληροφορία. Τέλος, εμφανίζουμε τα αποτελέσματα σε ένα νέο παράθυρο.

Για την εισαγωγή ενός νέου ΙΤ από τον συνδεδεμένο IT, δημιουργήσαμε μία νέα κλάση, την InsertNewITWindow, όπου ο χρήστης εισάγει τα ΑΤ, όνομα, επώνυμο και μισθό του νέου IT που θέλει να εισάγει. Έπειτα καλείται μία μέθοδος της ίδιας της κλάσης όπου ελέγχουμε αν υπάρχει κάποιος worker με ίδιο ΑΤ, αν υπάρχει εμφανίζεται αντίστοιχο μήνυμα. Αν δεν υπάρχει, ανακτούμε από τη βάση τον αριθμό του branch στο οποίο ανήκει ο ΙΤ που κάνει την εισαγωγή, εισάγουμε πρώτα τον νέο ΙΤ στον πίνακα worker και έπειτα στον πίνακα ΙΤ με το AT του, το default password και ημερομηνία έναρξης καθηκόντων την ημερομηνία εισαγωγής των στοιχείων του.

Για να μπορεί να δει ο ΙΤ όλες τις ενέργειες που έχουν καταγραφεί στον πίνακα IT\_logs, δημιουργήσαμε ακόμα μία μέθοδο εντός της κλάσης MainAppWindow, η οποία μέσω ενός “SELECT” query, ανακτά από τον πίνακα it\_logs τις πληροφορίες σχετικά με τους πίνακες, τις ενέργειες που έγιναν σε αυτούς, την ημερομηνία που έγιναν, καθώς και με ένα “INNER JOIN” με τον πίνακα worker, τα επώνυμα των IT’s που πραγματοποίησαν αυτές τις ενέργειες. Τέλος, εμφανίζουμε τα αποτελέσματα σε ένα νέο παράθυρο.

Σημειώσεις:

* Περιγράφοντας τις λειτουργίες που υλοποιήσαμε, αναφέρουμε π.χ. δημιουργήσαμε την X κλάση και ο χρήστης εισάγει/επιλέγει στο παράθυρό που εμφανίζεται… . Η εμφάνιση των παραθύρων γίνεται μέσω μεθόδων σε κάθε κλάση, οι οποίες καλούνται από τα αντικείμενα ή τα στιγμιότυπα των κλάσεων που δημιουργούμε κατά απαίτηση, απλά για συντομία παραλείπεται όλο αυτό στην περιγραφή.
* Σε μερικά πεδία απαιτείται η συμπλήρωση ημερομηνιών. Η ημερομηνίες δημιουργούνται με τις ίδιες μεθόδους κάθε φορά, μέσω απλών for loops, και έχουν τη μορφή array.
* Όπου αναφέρουμε ότι εμφανίζουμε τα αποτελέσματα σε νέο παράθυρο, αυτό γίνεται μέσω μίας μεθόδου της κλάσης ResultScreen, όπου η μέθοδος δέχεται σαν όρισμα τη λίστα αλφαριθμητικών που θέλουμε να εκτυπώσουμε, δημιουργεί ένα νέο παράθυρο και εμφανίζει το περιεχόμενο της λίστας διαπερνόντας την με μία απλή for loop.

3.2.3

Όπως αναφέρουμε και στις παραδοχές, καθώς η procedure α) τους ερωτήματος 3.1.3.4 δεν ζητείται να χρησιμοποιηθεί στο GUI, την προσθέσαμε σαν επιπλέον λειτουργικότητα. Συγκεκριμένα, δημιουργήσαμε την κλάση getCustomersByAdvance, στο παράθυρο της οποίας ο χρήστης συμπληρώνει δυο τιμές, το εύρος δηλαδή της προκαταβολής για την αναζήτηση και επιλέγει “Search”. Έπειτα, καλείται μια μέθοδος εντός της ίδιας της κλάσης η οποία καλή την Procedure get\_Res\_Offers\_Names() και επιστρέφει τα αποτελέσματα της procedure σε μορφή λίστας. Τέλος, τα αποτελέσματα εμφανίζονται σε νέο παράθυρο.

Επίσης, προσθέσαμε τη δυνατότητα προβολής του δημοφιλέστερου, μέχρι στιγμής, προορισμού. Αυτό έγινε με τη δημιουργία μίας μεθόδου μέσα στην κλάση MainAppWindow, όπου (ξεκινάμε από μέσα προς τα έξω):

(SELECT t.to\_tr\_id AS trip,t.to\_dst\_id AS dest\_id,SUM(res.participation) AS sum\_part FROM travel\_to t

LEFT JOIN (SELECT res\_tr\_id,COUNT(\*) AS participation FROM reservation GROUP BY res\_tr\_id)

AS res ON res.res\_tr\_id=t.to\_tr\_id

GROUP BY t.to\_dst\_id) : με εσωτερικό το sub-query παίρνουμε το σύνολο των κρατήσεων ανά ταξίδι και με το “LEFT JOIN” του εξωτερικού sub-query με το εσωτερικό και την “GROUP BY” παίρνουμε το σύνολο των κρατήσεων ανά προορισμό.

(SELECT k.dest\_id AS destination,k.sum\_part AS booked FROM trip tr INNER JOIN

(SELECT t.to\_tr\_id AS trip,t.to\_dst\_id AS dest\_id,SUM(res.participation) AS sum\_part FROM travel\_to t

LEFT JOIN (SELECT res\_tr\_id,COUNT(\*) AS participation FROM reservation GROUP BY res\_tr\_id)

AS res ON res.res\_tr\_id=t.to\_tr\_id

GROUP BY t.to\_dst\_id) k ON tr.tr\_id=k.trip WHERE tr.tr\_return<NOW()) AS trips : παίρνουμε τους προορισμούς και τους επισκέπτες του από τα ταξίδια που έχουν γίνει “WHERE tr.tr\_return<NOW()”

SELECT d.dst\_id,d.dst\_name,trips.booked FROM destination d INNER JOIN

(SELECT k.dest\_id AS destination,k.sum\_part AS booked FROM trip tr INNER JOIN

(SELECT t.to\_tr\_id AS trip,t.to\_dst\_id AS dest\_id,SUM(res.participation) AS sum\_part FROM travel\_to t

LEFT JOIN (SELECT res\_tr\_id,COUNT(\*) AS participation FROM reservation GROUP BY res\_tr\_id)

AS res ON res.res\_tr\_id=t.to\_tr\_id

GROUP BY t.to\_dst\_id) k ON tr.tr\_id=k.trip WHERE tr.tr\_return<NOW()) AS trips

ON trips.destination=d.dst\_id

ORDER BY d.dst\_id; : παίρνουμε τις πληροφορίες του προηγούμενου συν τα ονόματα αυτών των προορισμών.

Έπειτα, ανακτούμε τα αποτελέσματα και ταυτόχρονα βρίσκουμε τον προορισμό με τους περισσότερους επισκέπτες. Τέλος, εμφανίζουμε σε νέο παράθυρο τους προορισμούς και το πλήθος των επισκεπτών τους καθώς και τον πιο δημοφιλή προορισμό.