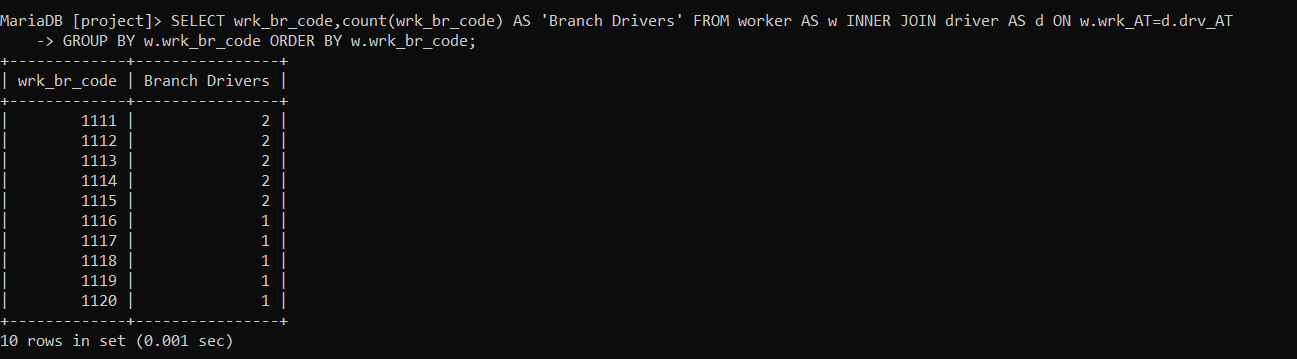
Α’ Μέρος

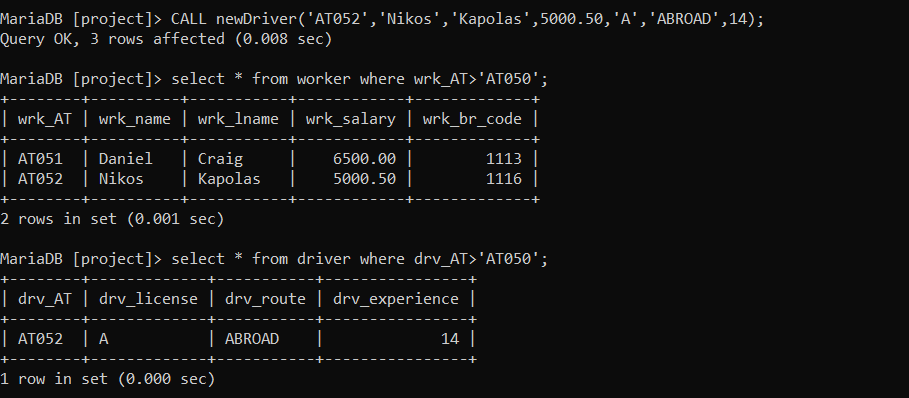
2.

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.

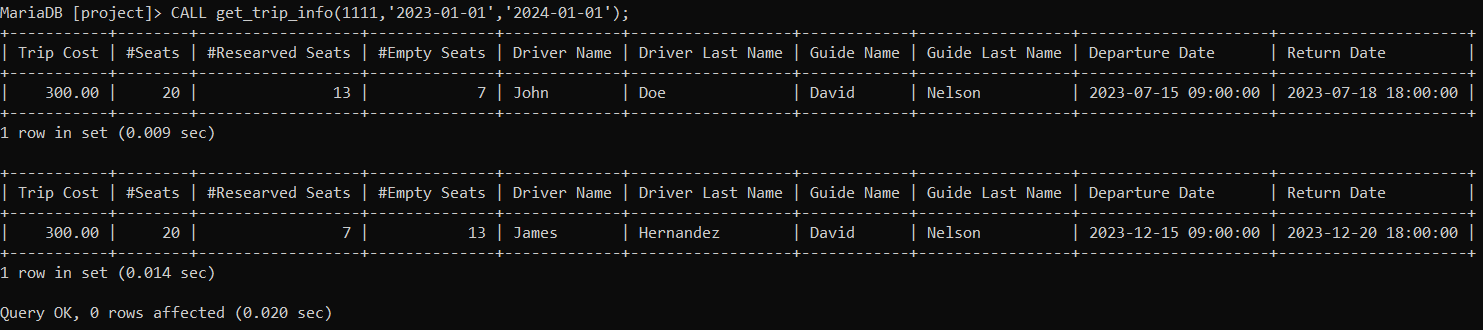


Καλούμε την procedure newDriver() για νέο οδηγό και βλέπουμε ότι εισάγει το νέο οδηγό στον πίνακα worker με wrk\_br\_code=1116, δηλαδή στο πρώτο κατάστημα με τους λιγότερους οδηγούς και στον πίνακα driver:



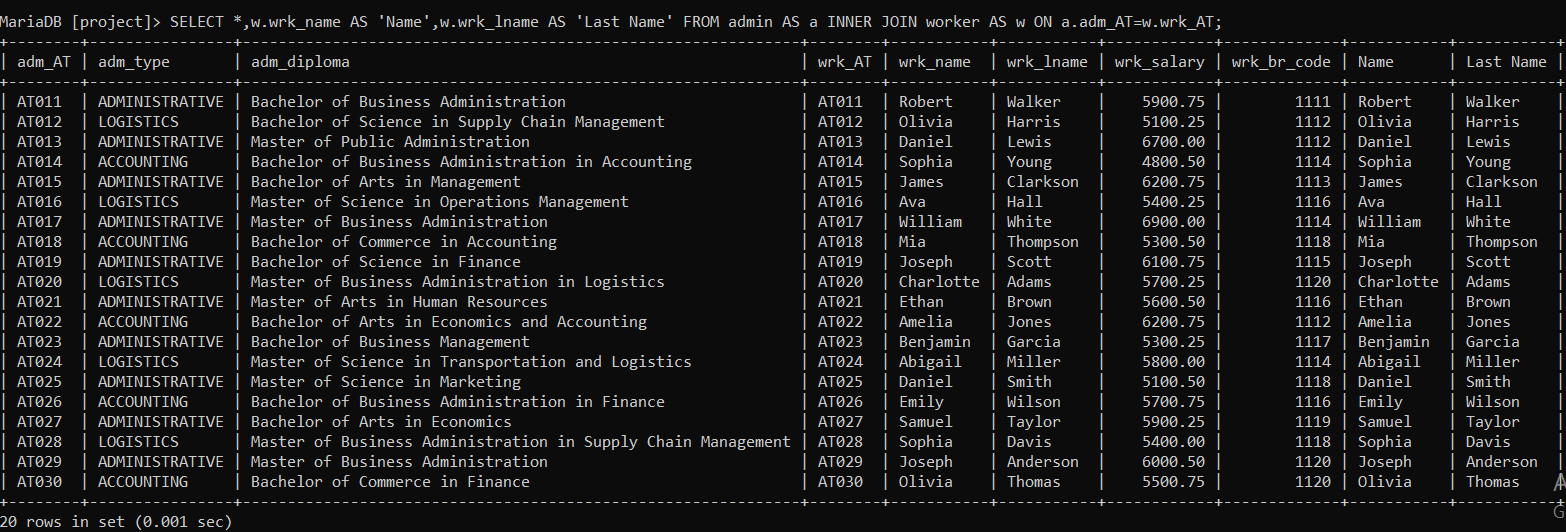
3.1.3.2. Η procedure get\_trip\_info() παίρνει ως ορίσματα έναν κωδικό καταστήματος και δύο ημερομηνίες και επιστρέφει πληροφορίες για τα ταξίδια που διοργανώνει το κατάστημα, με το δοσμένο κωδικό, ανάμεσα από τις δύο ημερομηνίες.

Καλούμε την procedure get\_trip\_info():

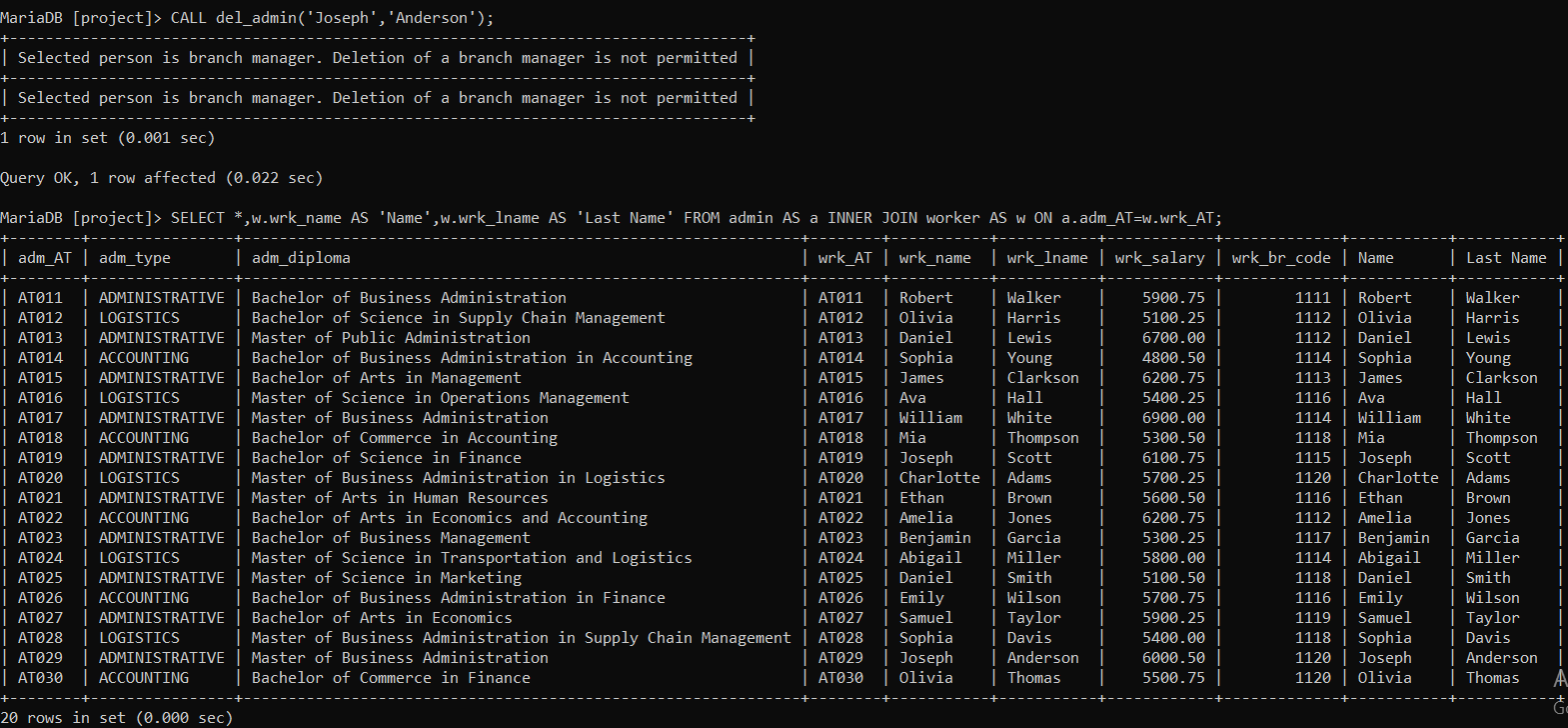


3.1.3.3. Η procedure del\_admin() παίρνει ως ορίσματα το όνομα και το επώνυμο ενός υπαλλήλου και αν ανήκει στο διοικητικό προσωπικό και δεν είναι διευθυντής κάποιου υποκαταστήματος, τον διαγράφει.

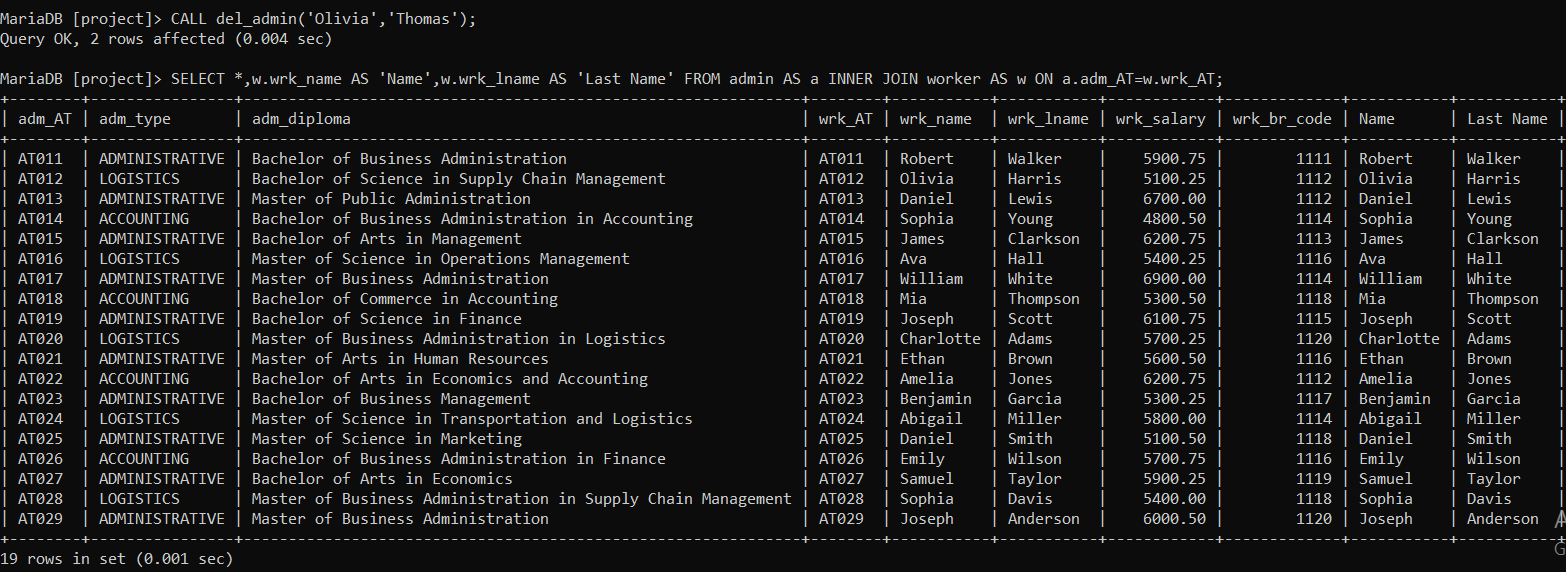
Οι εργαζόμενοι που ανήκουν στο διοικητικό προσωπικό είναι οι παρακάτω:



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 δείκτη 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

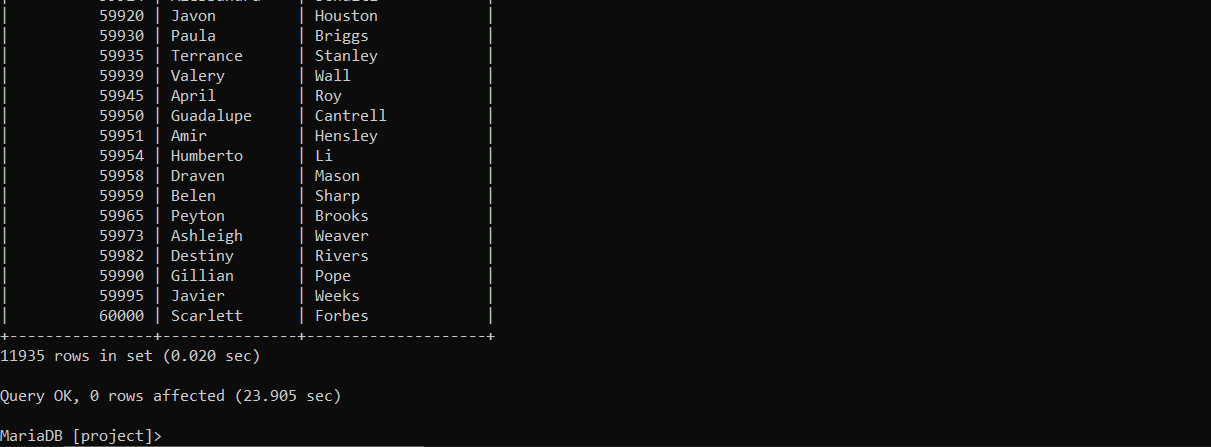
Παρακάτω, παραθέτουμε πρώτα τα αποτελέσματα από τις αναζητήσεις με/χωρίς indexes, και έπειτα τα προβλήματα που έπρεπε να επιλυθούν όταν κάναμε την αλλαγή από InnoDB σε Memory ENGINE στον πίνακα reservation\_offers\_memory.

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);

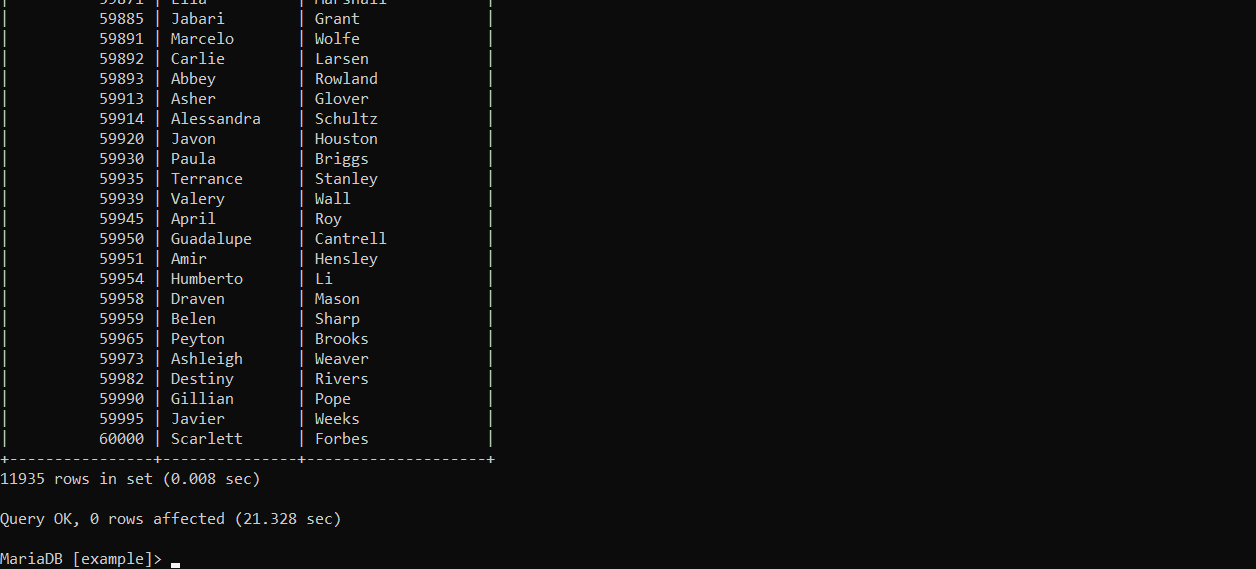
Έξοδος (χωρίς index, παρατίθεται μέρος της εξόδου, λόγω του μεγέθους της):



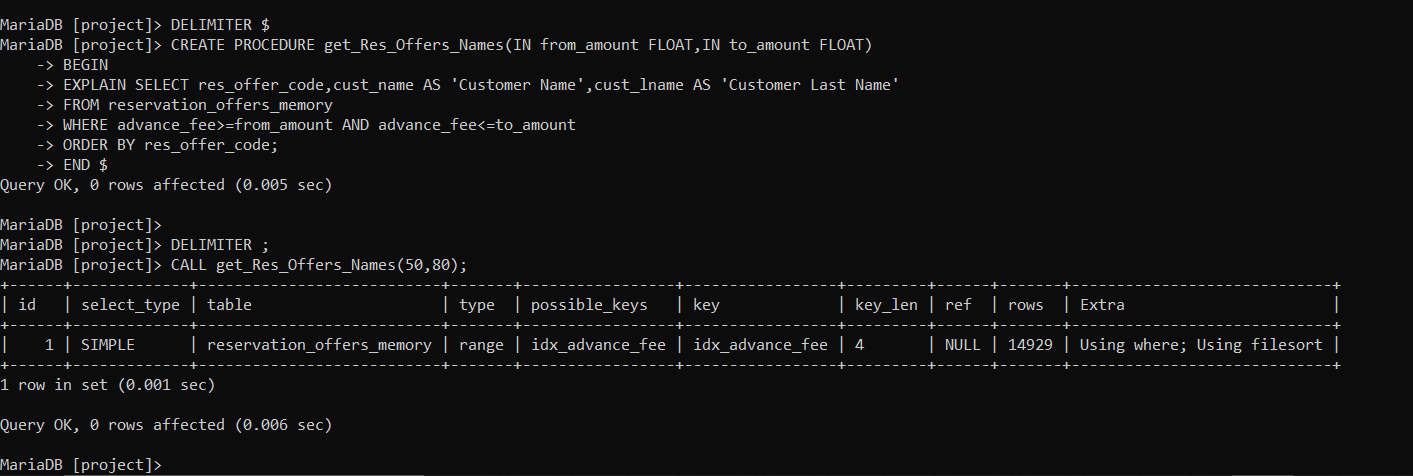
Δημιουργούμε τον index:

-> 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:

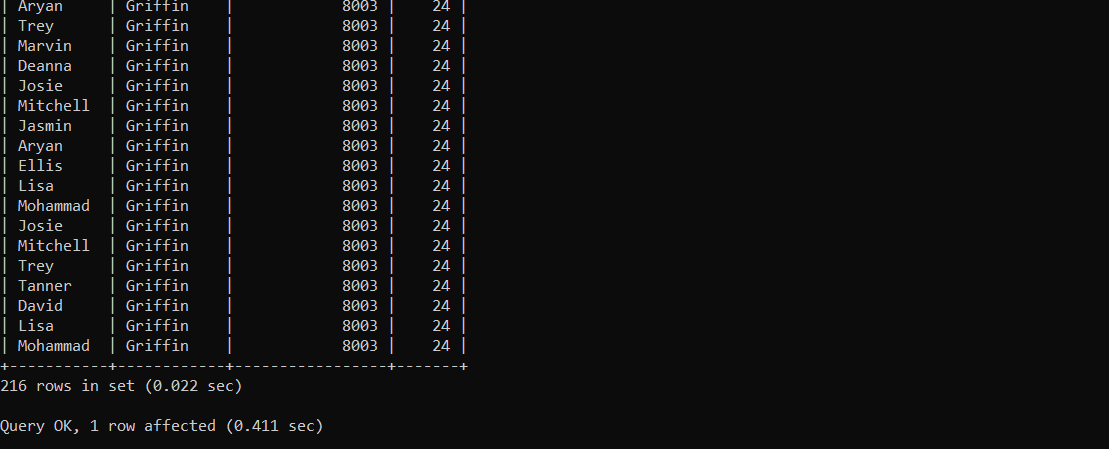


β)

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.

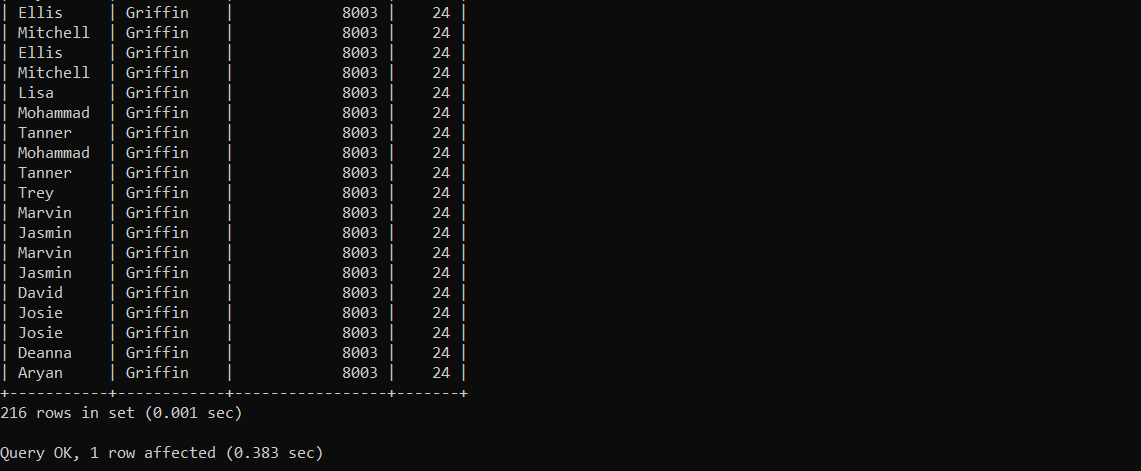
-> CALL get\_Offers\_Participation(‘Griffin’);

Έξοδος (χωρίς index):



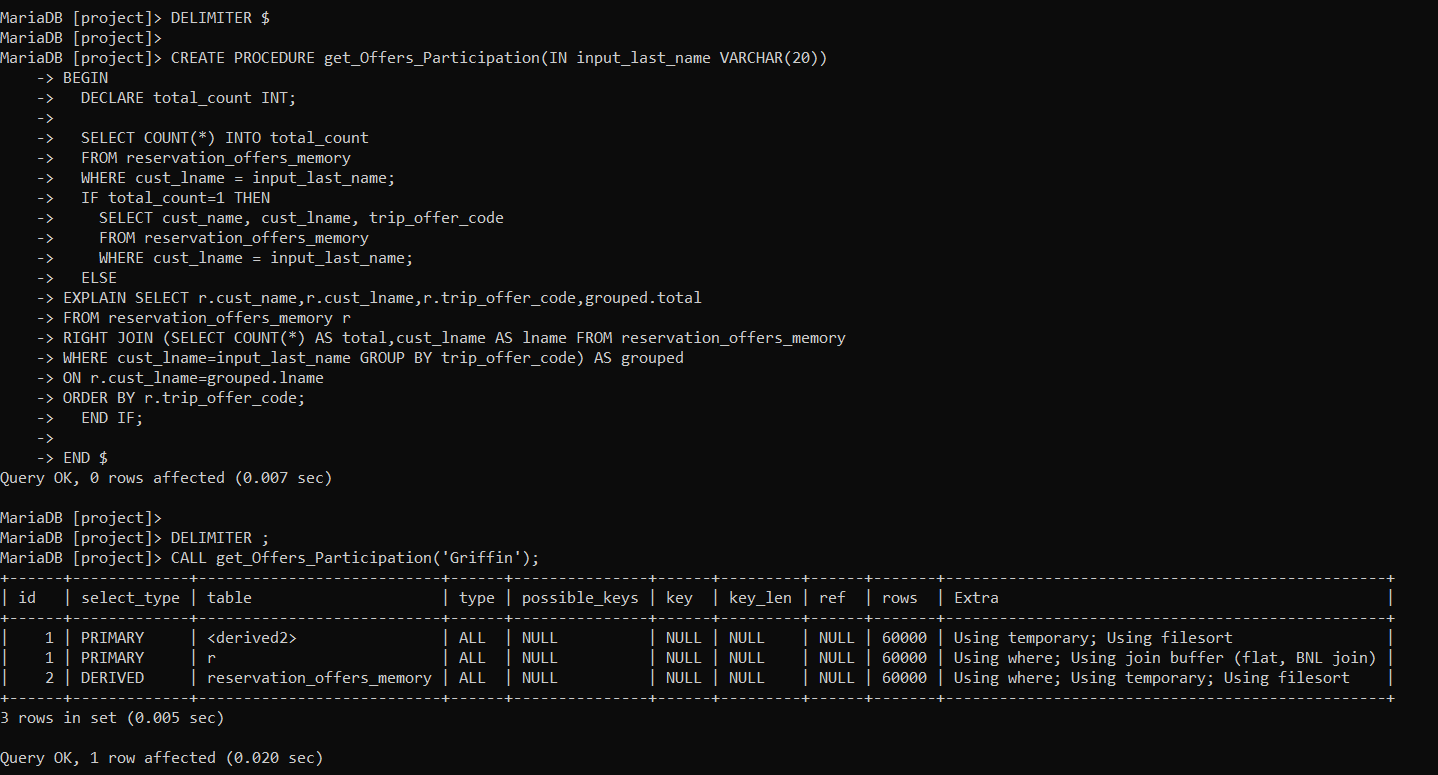
-> CREATE INDEX idx\_cust\_lname ON reservation\_offers\_memory(cust\_lname) USING HASH;

Έξοδος (με index):

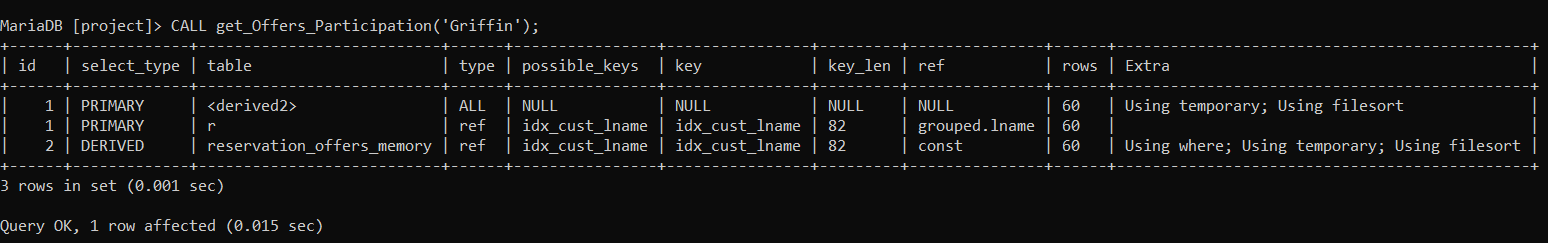


Εδώ παρατηρούμε αρκετά μεγάλη διαφορά στο χρόνο εκτέλεσης (από 0.022sec -> 0.001sec), κάτι που οφείλεται στο γεγονός ότι με χρήση HASH έχουμε ακριβή ταύτιση στην αναζήτηση, άρα δεν χάνουμε ίσως τόσο πολύ χρόνο σε ανισο-ισότητες όπως σε ένα BTREE. Βέβαια και πάλι ο χρόνος εκτέλεσης είναι αμελητέος (τάξης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

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:

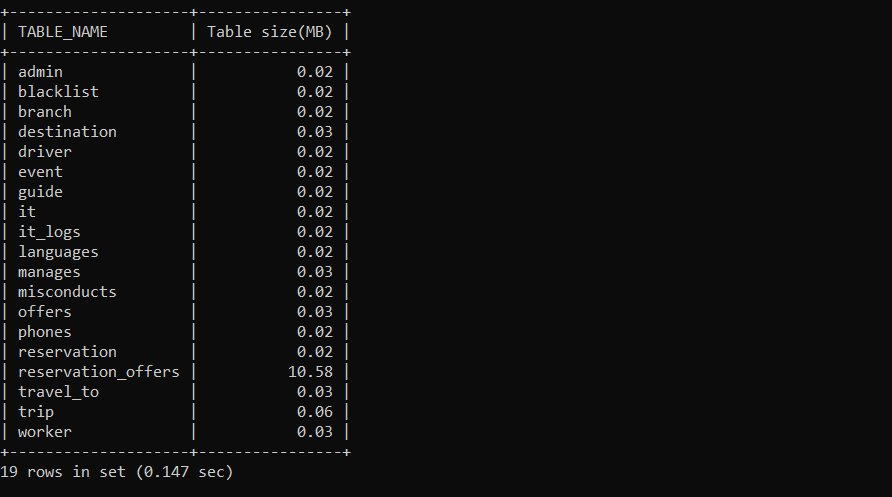
-> SELECT TABLE\_NAME,ROUND(SUM(DATA\_LENGTH + INDEX\_LENGTH) / 1024 / 1024, 2) AS 'Table Size (MB)'

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.