**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

**ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ**

**ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ**

**ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ**

Χρήστος Ηλιακόπουλος el20233

Παναγιώτης Τσακλάνος el18937

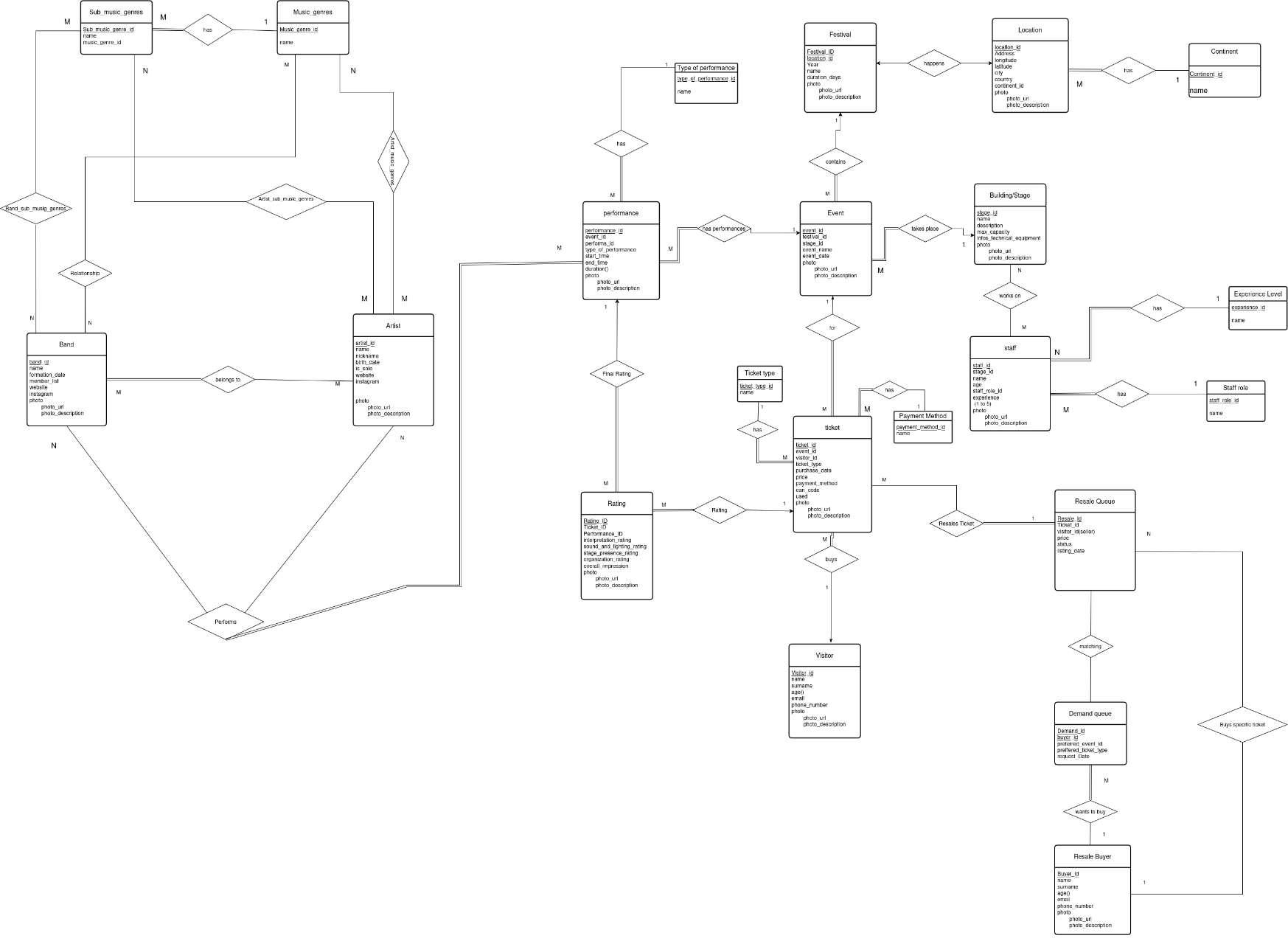
Δημήτρης Τζέλλος el21623

**ΕΙΣΑΓΩΓΉ**

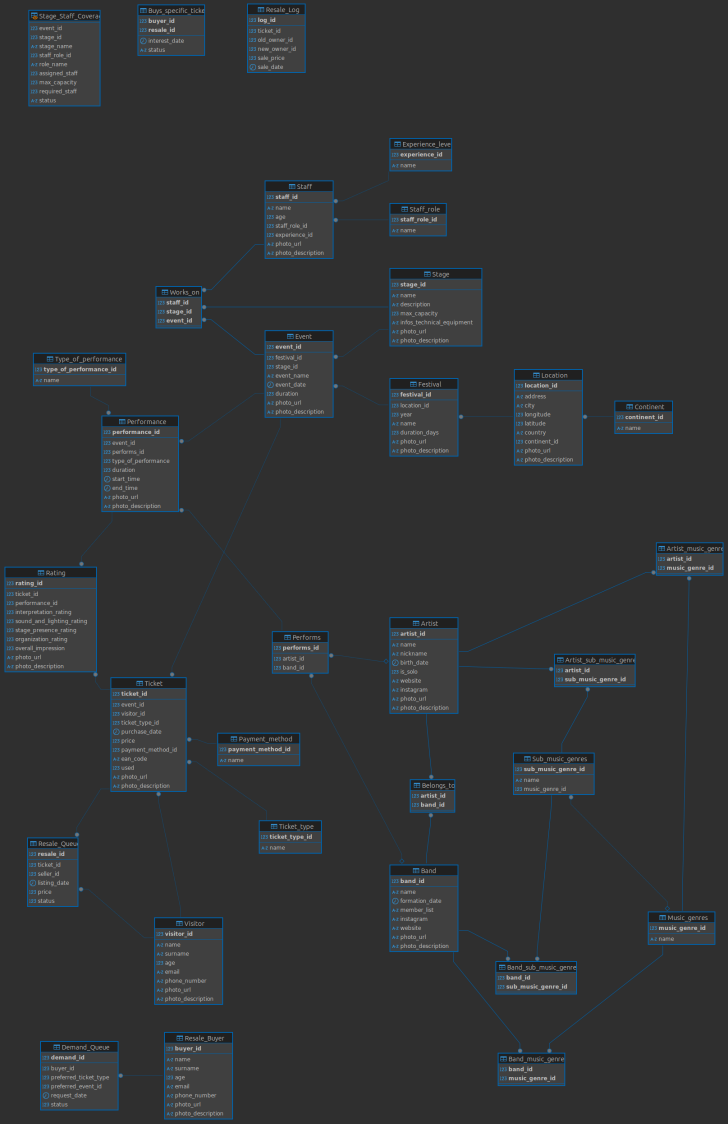
Η παρούσα εργασία παρουσιάζει τη σχεδίαση και την υλοποίηση μιας βάσης δεδομένων για το διεθνές φεστιβάλ Pulse University. Μας ζητήθηκε ο σχεδιασμός και η υλοποίοηση ενός συστήματος αποθήκευσης και διαχείρισης πληροφοριών που απαιτούνται για την οργάνωση και τη διεξαγωγή του.

**ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ**

**2.1)** Παρουσιάζεται το διάγραμμα Οντοτήτων-Συσχετίσεων (Entity – Relationship Diagram) το οποίο δημιουργήθηκε με βάση τις απαιτούμενες προδιαγραφές της εκφώνησης (παράχθηκε με το draw.io)



**2.2)** Με βάση το παραπάνω διάγραμμα Οντοτήτων – Συσχετίσεων προκύπτει το σχεσιακό διάγραμμα ως εξής (παράχθηκε με το Dbeaver):



Στο σχεσιακό διάγραμμα εμφανίζονται όλες οι οντότητες που παρατηρούνται στο ER διάγραμμα μαζί με τις αντίστοιχες σχέσεις τους.

Η οντότητα Festival αναφέρει όλα τα στοιχεία που πρέπει να έχει κάθε φεστιβάλ όπως το όνομα, την χρονιά που πραγματοποιείται, πόσες μέρες διαρκεί, φωτογραφία και περιγραφεί για το ίδιο καθώς και σύνδεση με την τοποθεσία που πραγματοποιείτε μέσω του location\_id ως foreign key. Τέλος κάθε φεστιβάλ έχει το δικό του μοναδικό αναγνωριστικό festival\_id.

Η οντότητα Location αναφέρει τα αναγκαία χαρακτηριστικά που πρέπει να έχει μία τοποθεσία, όπως είναι η διεύθυνση, η πόλη, η χώρα, το γεωγραφικό μήκος και πλάτος, φωτοφραφία της μαζί με περιγραφή για αυτή. Ακόμη, έχει ένα μοναδικό αναγνωριστικό για να ξεχωρίζουν οι τοποθεσίες μεταξύ τους, το location\_id. Η ήπειρος εισάγεται ως foreign key από τον lookup table με το όνομα Continent.

Η οντότητα Continent είναι ένας απλος look up table που βοηθάει στην καλύτερη εισαγωγή και ταξινόμηση των ηπείρων, δίνοντας σε κάθε ήπειρο μοναδικό key με το continent\_id

Η οντότητα Event περιέχει πληροφορίες για το όνομα, την ημέρα που διεξάγεται, την διάρκεια του Event αλλά και αντίστοιχες φωτογραφίες μαζί με περιγραφή για κάθε ένα Event. Ταυτόχρονα συνδέεται μέσω foreign keys με το φεστιβάλ στο οποίο θα πραγματοποιηθεί το εκάστοτε Event αλλά και τον χώρο που του αναλογεί, χρησιμοποιώντας αντίστοιχα festival\_ids και stage\_ids. Κάθε Event ξεχωρίζει με το δικό του μοναδικό αναγνωριστικό event\_id

Η οντότητα Stage που αναφέρθηκε προηγουμένως περιγράφει τους χώρους που θα πραγματοποιούνται τα Events. Περιέχει πληροφορίες για το όνομα, την περιγραφή του χώρου, την μέγιστη δυνατή χωριτικότητα, τις πληροφορίες για τον τεχνικό εξοπλισμό κάθε χώρου και φωτογραφία για αυτόν μαζί με περιγραφή. Κάθε χώρος έχει το δικό του μοναδικό αναγνωριστικό stage\_id

Η οντότητα Staff αναφέρεται στο προσωπικό που θα εργάζεται σε κάθε χώρο, και περιγράφεται από το όνομα, την ηλικία, τον ρόλο στον οποίο δουλεύει και την αντίστοιχη εμπειρία που κατέχει, μία φωτογραφία μαζί με την αντίστοιχη περιγραφή και φυσικα μοναδικό αναγνωριστικό staff\_id.

Οι οντότητες Experience\_Level και Staff\_role είναι και οι δύο look up tables που δημιουργήθηκαν για τα αντίστοιχα attributes στο Staff table. Σκοπός τους είναι η καλύτερη οργάνωση και διαχείριση των συγκεκριμένων δεδομένων, περιέχοντας όνομα για κάθε κατηγορία (εμπειρίας και ρόλου εργασίας) και τα αντίστοιχα μοναδικά αναγνωριστικά exprerience\_id και staff\_role\_id αντίστοιχα

Για την σύνδεση των οντοτήτων Stage, Event και Staff δημιουργήσαμε τη σχέση Works On, η οποία είναι υπεύθυνη για την ανάθεση και αποθήκευση τον αρχείων προσωπικού σε αντίστοιχα stages για αντίστοιχα events.

Η οντότητα Performance περιγράφει μία εμφάνιση που γίνεται σε ένα συγκεκριμένο Event. Περιγράφεται το είδος του performance, από τη διάρκεια του, την ώρα έναρξης και ώρα λήξης, μαζί με την αντίστοιχη φωτογραφία και περιγραφή της. Περιέχει ένα μοναδικό αναγνωριστικό, το performance\_id, ενώ συνδέεται με ένα μοναδικό event μέσω του event\_id ως foreign key. Τέλος περιέχει και ένα ακόμα foreign key, το performs\_id το οποίο θα δούμε στη συνέχεια.

Η οντότητα Artist αναφέρει τα απαραίτητα χαρακτηριστικά που πρέπει να έχει ένας καλλιτέχνης, όπως όνομα, nickname, ημερομηνία γέννησης, αν είναι σόλο ή ανήκει σε κάποια μπάντα, αν έχει προσωπική ισοστελίδα και προφίλ στο instagram. Ακόμη περιέχει και αντίστοιχη φωτογραφία του καλλιτέχνη με περιγραφή της, ενώ κάθε καλλιτέχνης έχει μοναδικό αναγνωριστικό το artist\_id. Η σύνδεση με τα μουσικά του είδη και υποδείδη του κάθε καλλιτέχνη θα αναλυθεί παρακάτω

Η οντότητα Band αναφαίρει τα απαραίτητα χαρακτηριστικά που πρέπει να έχει μία μπάντα, όπως όνομα, ημερομηνία δημιουργίας, λίστα με τα μέλη της, προφίλ στο instagram και ιστοσελίδα της μπάντας. Ακόμη περιέχει φωτογραφία της μπάντας μαζί με αντίστοιχη περιγραφή της, ενώ κάθε μπάντα έχει μοναδικό αναγνωριστικό το band\_id. Αντίστοιχα με τους καλλιτέχνες τα είδη και υποείδη μουσικής της μπάντας θα αναλυθούν προσεχώς

Για τα είδη και υποείδη μουσικής δημιουργήσαμε 2 look up tables (Music\_genres και Sub\_music\_genres). Αρχικά για τα κύρια είδη μουσικής έχουμε ένα μοναδικό αναγνωριστικό music\_genre\_id και ένα όνομα που αναφέρει τι είδος είναι. Στη συνέχεια τα υποείδη έχουν αντίστοιχα και αυτά ένα μοναδικό αναγνωριστικό sub\_music\_genre\_id και όνομα για να πει τι είδος είναι. Όμως εδώ αυτός ο πίνακας έχει και ένα foreign key το music\_genre\_id (του Music\_genres πίνακα) που συνδέει το υποείδος με το κύριο είδος στο οποίο ανήκει.

Για να συνδεθούν καλλιτέχνες και μπάντες με είδη και υποείδη αντίστοιχα δημιουργήσαμε τις σχέσεις Artist\_music\_genres και Artist\_sub\_music\_genres για τους καλλιτέχνες, και Band\_music\_genres και Band\_sub\_music\_genres για τις μπάντες αντίστοιχα. Η κάθε σχέση περιέχει το αντίστοιχο αναγνωριστικό καλλιτέχνη ή μπάντας και το αντίστοιχο χαρακτηριστικό είδους και υποείδους μουσικής δημιουργώντας τη σύνδεση μεταξύ τους.

Για τη σύνδεση των καλλιτεχνών με τις αντίστοιχες μπάντες έχουμε τη σχέση Belongs\_to που ταιριάζει καλλιτέχνες σε αντίστοιχες μπάντες, χρησιμοποιώντας τα artist\_id και band\_id

Για να συνδέσουμε τους καλλιτέχνες και τις μπάντες με αντίστοιχα Performances που πραγματοποιούνται έχουμε δημιουργήσει τη σχέση Performs, η οποία περιέχει ένα μοναδικό αναγνωριστικό performs\_id και μετά είτε artist\_id είτε band\_id με περιορισμό να μην γίνει ποτέ εισαγωγή σε αυτή με είτε κανένα από τα δύο ή και τα δύο.

Η οντότητα Visitor περιέχει τα απαραίτητα στοιχεία κάθε επισκέπτει όπως όνομα, επίθετο, ηλικία, email, τηλέφωνο και αντίστοιχη φωτογραφία με την δική της περιγραφή. Ακόμη, κάθε visitor έχει το δικό του μοναδικό αναγνωριστικό visitor\_id

Η οντότητα Ticket περιέχει τα απαραίτητα χαρακτηριστικά κάθε εισιτηρίου, όπως τον τύπο του εισιτηρίου, την ημερομηνία αγοράς, την τιμή του, τον τρόπο πληρωμής κατά την αγορά του, κωδικό EAN, αν είναι χρησιμοποιημένο, αντίστοιχη φωτογραφία και περιγραφή της. Ακόμη περιέχει το αναγνωριστικό του event στο οποίο ανήκει (event\_id), αναγνωριστικό του ιδιοκτήτη του (visitor\_id) και μοναδικό αναγνωριστικο ticket\_id

Για το είδος της πληρωμής κατά την αγορά του εισιτηρίου αλλά και το είδος του εισιτηρίου έχουμε δημιουργήσει δύο look up tables, το Payment\_method και το Ticket\_type. Το κάθε ένα περιέχει αντίστοιχο αναγνωριστικό για την εκάστοτε κατηγορία και όνομα που το περιγράφει.

Η οντότητα Rating περιέχει τα απαραίτητα χαρακτηριστικά για κάθε βαθμολογία, όπως αξιολόγηση ερμηνείας καλλιτεχνών, αξιολόγηση ήχου και φωτισμού, αξιολόγηση σκηνικής παρουσίας, αξιολόγηση οργάνωσης, συνολική εντύπωση και φωτογραφία μαζί με την αντίστοιχη περιγραφή της. Ακόμη, για να υπάρξει rating εισαγωγή (θα δούμε στη συνέχεια πως υλοποιείται ο περιορισμός) πρέπει να υπάρχει και χρησιμοποιημένο εισιτήριο αλλά και αντίστοιχο performance όπου γίνεται το rate. Επομένως κάθε rating συνδεέται με τα εισιτήρια χρησιμοποιώντας ως foreign key το ticket\_id και με τα performances χρησιμοποιώντας το performance\_id. Τέλος, κάθε αξιολόγηση έχει το δικό της μοναδικό αναγνωριστικό rating\_id

Για την ουρά μεταπώλησης υλοποιήθηκε το παρακάτω:

Αρχικά μόλις γίνει ένα evet sold out ένα εισιτήριο μπορεί να τοποθετηθεί στην Resale Queue σε περίπτωση που θελήσει κάποιος αγοραστής να το αγοράσει. Ταυτόχρονα υπάρχει και η Demand Queue, η οποία έχει καταχωρήσεις επιθυμιών από αγοραστές για συγκεκριμένα εισιτήρια με βάση τον τύπο του εισιτηρίου και το event στο οποίο ανήκει. Η ανταλλαγή γίνεται αυτόματα μεταξύ των δύο όπως θα δούμε στη συνέχεια

Η οντότητα Resale Queue περιέχει το αναγνωριστικό του εισιτηρίου που τοποθετείτε, το αναγνωριστικό του πωλητή, την ημερομηνία που γίνεται η καταχώρηση, την τιμή και το status της εξέλιξης της συγκεκριμένης εισαγωγής (ικανοποιήθηκε ή όχι). Κάθε εισαγωγή έχει μοναδικό αναγνωριστικό resale\_id

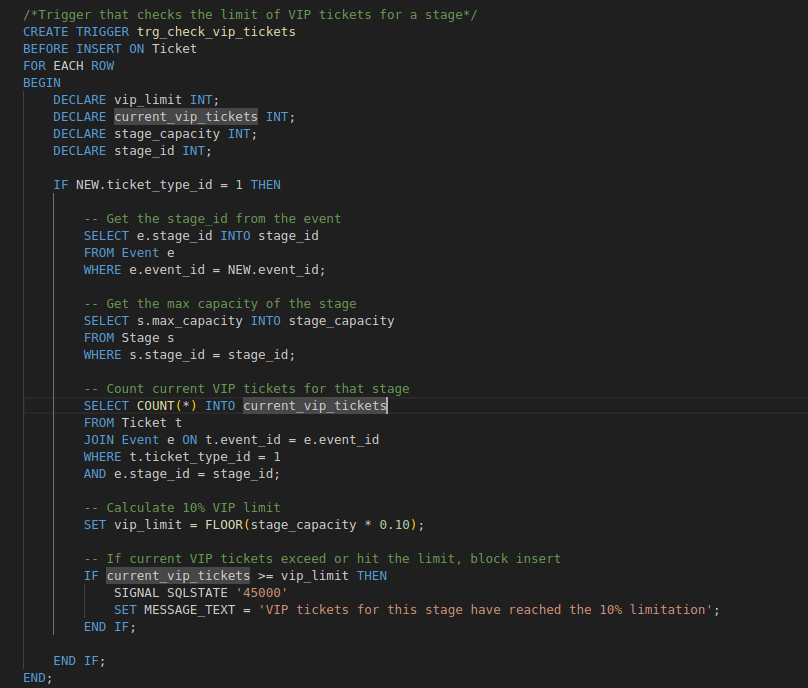
Η οντότητα Demand Queue περιέχει το αναγνωριστικο του αγοραστή, τον προτιμόμενο τύπο εισιτηρίου και το προτιμόμενο event, την ημερομηνία καταχώρησης του log, το status για το αν έχει ικανοποιηθεί και φυσικά μοναδικό αναγνωριστικό demand\_id.

Η οντότητα Resale Buyer περιέχει το όνομα του αγοραστή, το επίθετό του, την ηλικία του, το email του, το τηλέφωνό του και φωτογραφία με αντίστοιχη περιγραφή. Ακόμη, κάθε αγοραστής έχει μοναδικό αναγνωριστικό buyer\_id

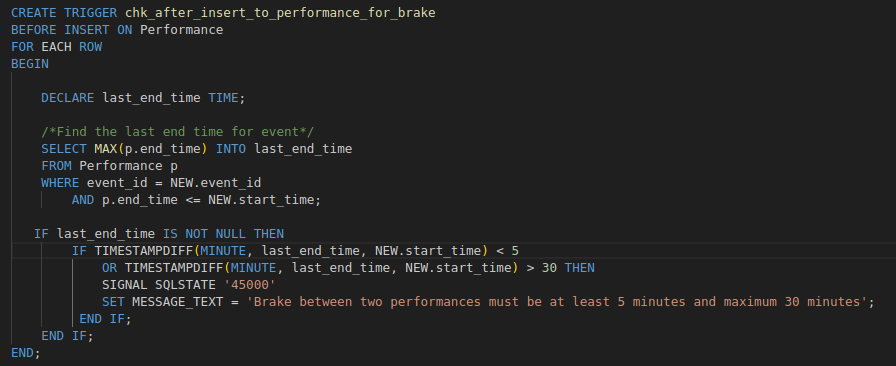
Στο σχεσιακό εμφανίζονται και μερικές ακόμα οντότητες όπως η Buy\_specific\_ticket, Resale\_Log που επιτελούν ρόλο καταγραφής δεδομένων

***Στη συνέχεια θα αναλύσουμε τα triggers που έχουμε δημιουργήσει:***

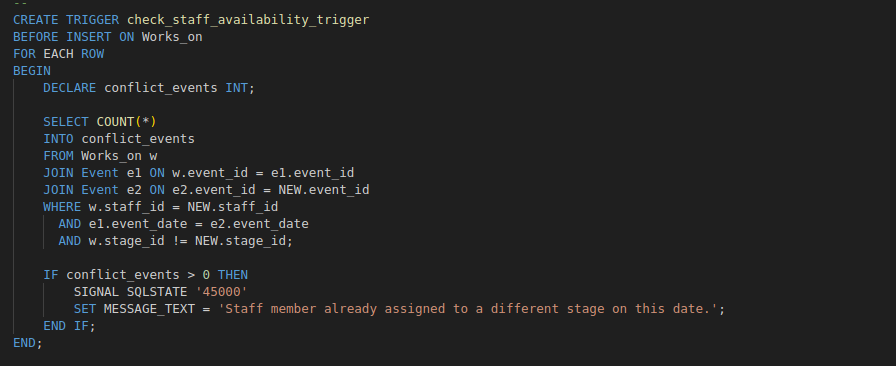
Το πρώτο μας trigger, με όνομα trg\_check\_vip\_tickets, έχει σκοπό να διασφαλίσει ότι ο αριθμός των VIP εισιτηρίων για κάθε σκηνή (stage) δεν ξεπερνά το όριο του 10% της μέγιστης χωρητικότητάς της, όπως ορίζεται στην εκφώνηση της εργασίας. Εκτελείται πριν από κάθε εισαγωγή νέου εισιτηρίου (BEFORE INSERT) στον πίνακα Ticket και ενεργοποιείται μόνο εάν το νέο εισιτήριο είναι τύπου VIP (ticket\_type\_id = 1). Ο μηχανισμός του trigger αντλεί το stage\_id μέσω της συσχέτισης με το αντίστοιχο event, υπολογίζει τη μέγιστη επιτρεπτή ποσότητα VIP εισιτηρίων (10% της χωρητικότητας της σκηνής) και ελέγχει πόσα τέτοια εισιτήρια έχουν ήδη καταχωρηθεί. Εάν το όριο έχει ήδη καλυφθεί ή ξεπεραστεί, η εισαγωγή απορρίπτεται και επιστρέφεται σχετικό μήνυμα σφάλματος.



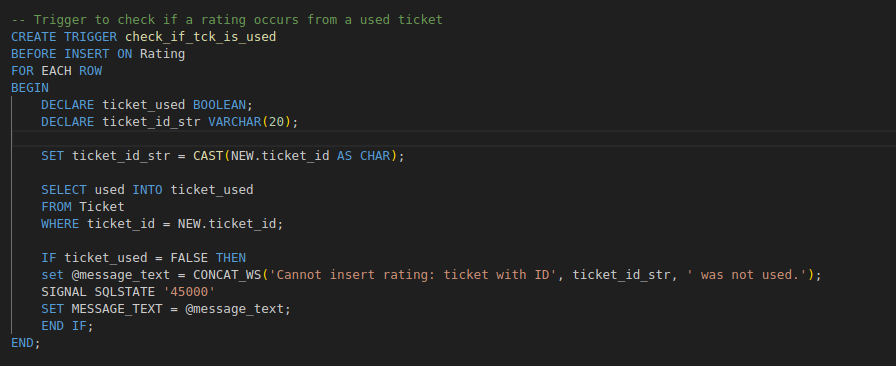
Το δεύτερο trigger με όνομα trigger\_chk\_after\_insert\_to\_performance\_for\_brake έχει στόχο να διασφαλίσει ότι τηρούνται τα χρονικά περιθώρια διαλείμματος ανάμεσα σε διαδοχικές εμφανίσεις (performances) στο ίδιο event. Εκτελείται πριν από την εισαγωγή (BEFORE INSERT) μιας νέας εμφάνισης στον πίνακα Performance και ελέγχει αν υπάρχει προηγούμενη εμφάνιση στο ίδιο event της οποίας η ώρα λήξης (end\_time) προηγείται της ώρας έναρξης (start\_time) της νέας εμφάνισης. Εφόσον υπάρχει τέτοια προηγούμενη εμφάνιση, υπολογίζει τη χρονική απόσταση μεταξύ των δύο εμφανίσεων και απορρίπτει την εισαγωγή εάν αυτή είναι μικρότερη από 5 λεπτά ή μεγαλύτερη από 30 λεπτά.



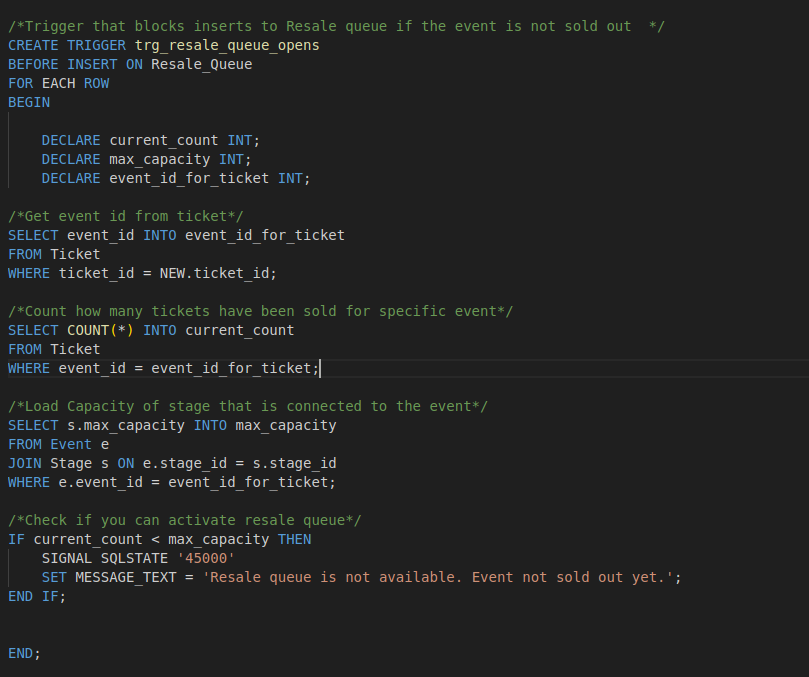
Το τρίτο trigger με όνομα trigger check\_staff\_availability\_trigger αποσκοπεί στον έλεγχο διαθεσιμότητας του προσωπικού κατά την εισαγωγή εγγραφών στον πίνακα Works\_on, που καταγράφει ποιο προσωπικό εργάζεται σε ποια σκηνή και εκδήλωση. Εκτελείται πριν την εισαγωγή (BEFORE INSERT) και επεμβαίνει για να αποτρέψει περιπτώσεις όπου το ίδιο άτομο έχει ήδη ανατεθεί σε διαφορετική σκηνή για την ίδια ημερομηνία. Συγκεκριμένα, ελέγχει αν υπάρχει ήδη ανάθεση του ίδιου μέλους προσωπικού (staff\_id) σε event με ίδια ημερομηνία (event\_date), αλλά διαφορετική σκηνή (stage\_id). Αν εντοπιστεί τέτοια σύγκρουση, η εισαγωγή απορρίπτεται και εμφανίζεται κατάλληλο μήνυμα σφάλματος.



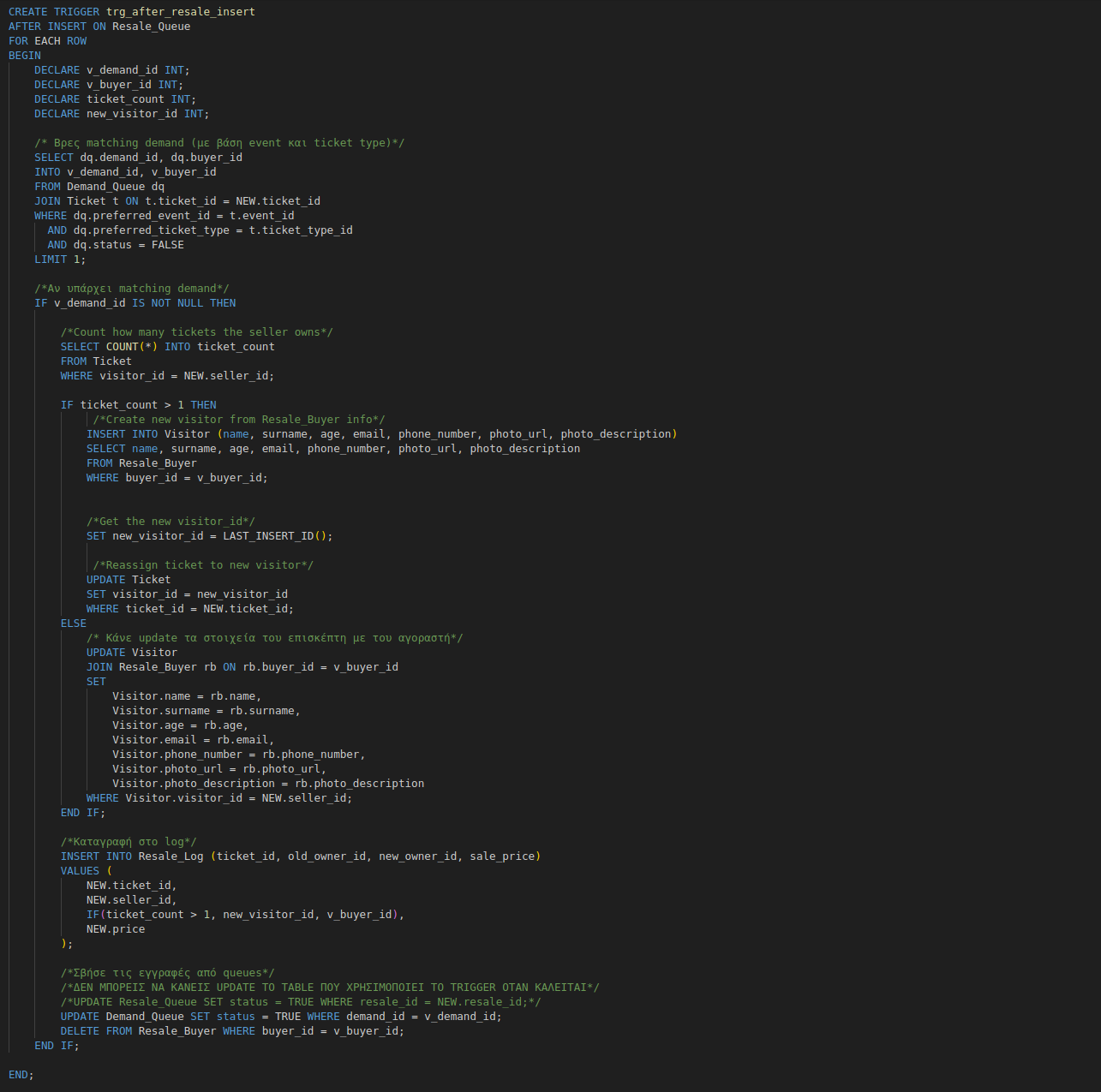
Το trigger check\_if\_tck\_is\_used εξασφαλίζει ότι δεν μπορεί να καταχωρηθεί αξιολόγηση (rating) από εισιτήριο που δεν έχει χρησιμοποιηθεί. Εκτελείται πριν την εισαγωγή (BEFORE INSERT) μιας νέας εγγραφής στον πίνακα Rating και ελέγχει την τιμή του πεδίου used του εισιτηρίου (Ticket) που σχετίζεται με την αξιολόγηση. Αν το εισιτήριο δεν έχει χρησιμοποιηθεί ακόμα (δηλαδή δεν έχει ενεργοποιηθεί κατά την είσοδο στο φεστιβάλ), τότε η εισαγωγή απορρίπτεται και εμφανίζεται προσαρμοσμένο μήνυμα σφάλματος που περιλαμβάνει το ID του εισιτηρίου.



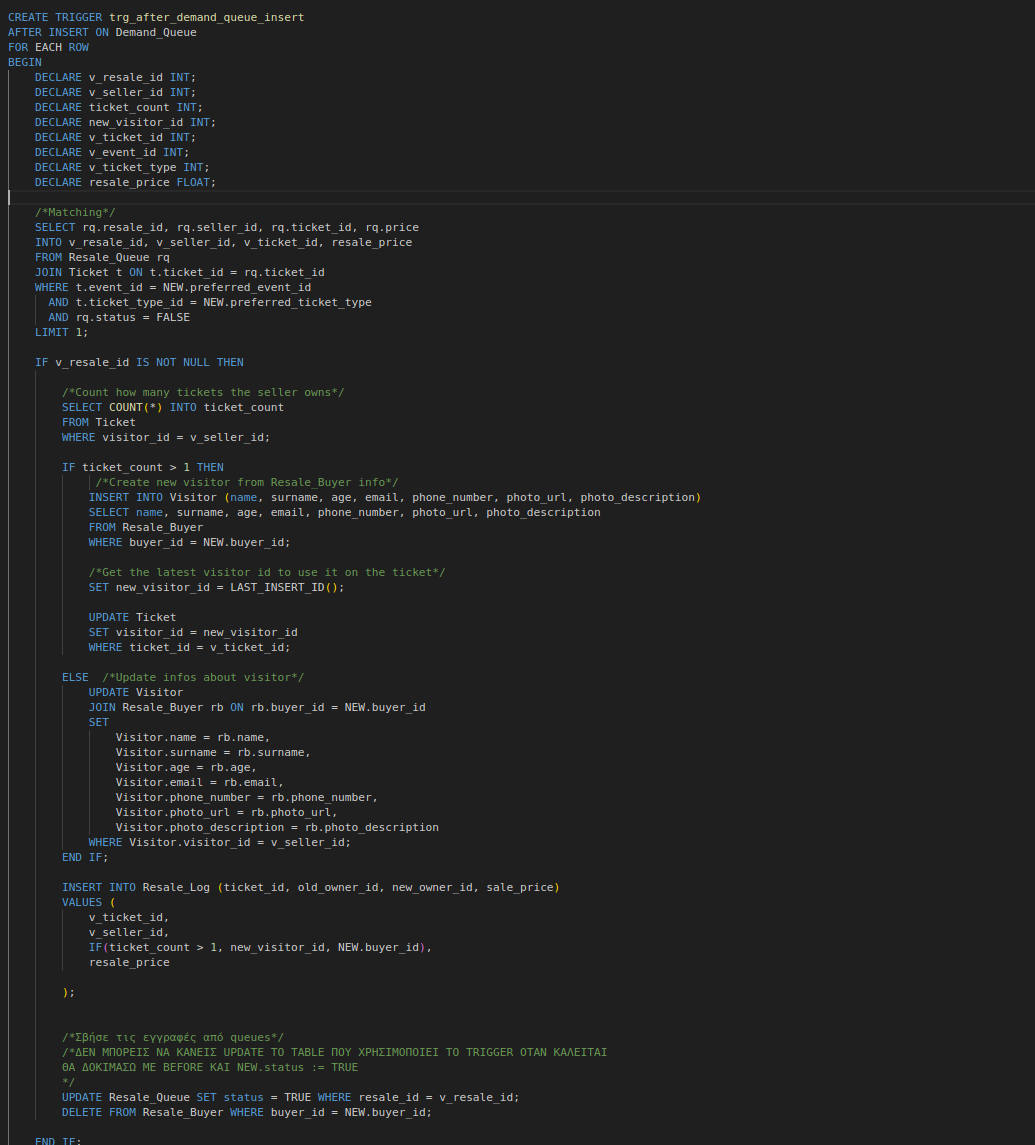
Το trigger trg\_resale\_queue\_opens εκτελείται πριν από την εισαγωγή (BEFORE INSERT) μιας νέας εγγραφής στον πίνακα Resale\_Queue και έχει σκοπό να διασφαλίσει ότι η μεταπώληση εισιτηρίων ενεργοποιείται μόνο όταν η παράσταση (event) είναι εξαντλημένη. Συγκεκριμένα, το trigger εντοπίζει ποιο event αφορά το εισιτήριο που τίθεται προς πώληση και ελέγχει πόσα εισιτήρια έχουν ήδη πουληθεί γι' αυτό. Στη συνέχεια, αντλεί τη μέγιστη χωρητικότητα της σκηνής που φιλοξενεί το event. Αν ο αριθμός των πωληθέντων εισιτηρίων είναι μικρότερος από τη μέγιστη χωρητικότητα της σκηνής, τότε η εισαγωγή απορρίπτεται με σχετικό μήνυμα σφάλματος.



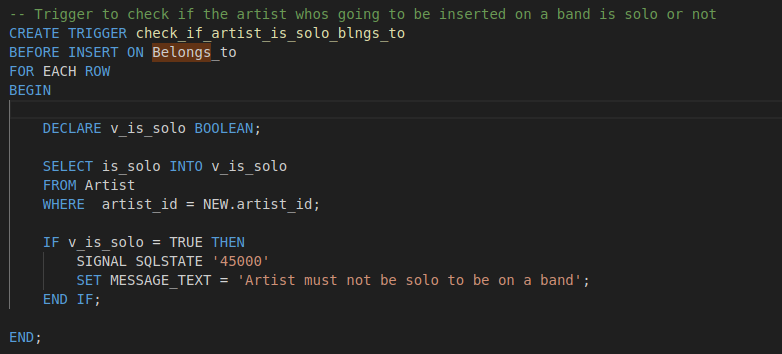
Το trigger trg\_after\_resale\_insert ενεργοποιείται μετά την εισαγωγή (AFTER INSERT) μιας εγγραφής στον πίνακα Resale\_Queue και έχει στόχο να υλοποιήσει αυτόματα τη μεταπώληση ενός εισιτηρίου, εφόσον υπάρχει αντίστοιχη ζήτηση από τον πίνακα Demand\_Queue. Συγκεκριμένα, αναζητά μία ενεργή ζήτηση για το ίδιο event και τύπο εισιτηρίου, και αν βρεθεί, προχωρά είτε σε μεταφορά του εισιτηρίου σε νέο επισκέπτη είτε σε ενημέρωση των προσωπικών στοιχείων του υπάρχοντος επισκέπτη, ανάλογα με το αν ο πωλητής διαθέτει περισσότερα από ένα εισιτήρια. Στην πρώτη περίπτωση δημιουργείται νέα εγγραφή στον πίνακα Visitor, ενώ στη δεύτερη γίνεται απευθείας ενημέρωση των στοιχείων του. Επιπλέον, καταγράφεται η πράξη στον πίνακα Resale\_Log και απενεργοποιείται η σχετική ζήτηση (status = TRUE) στον πίνακα Demand\_Queue. Τέλος, η εγγραφή του αγοραστή διαγράφεται από τον πίνακα Resale\_Buyer.



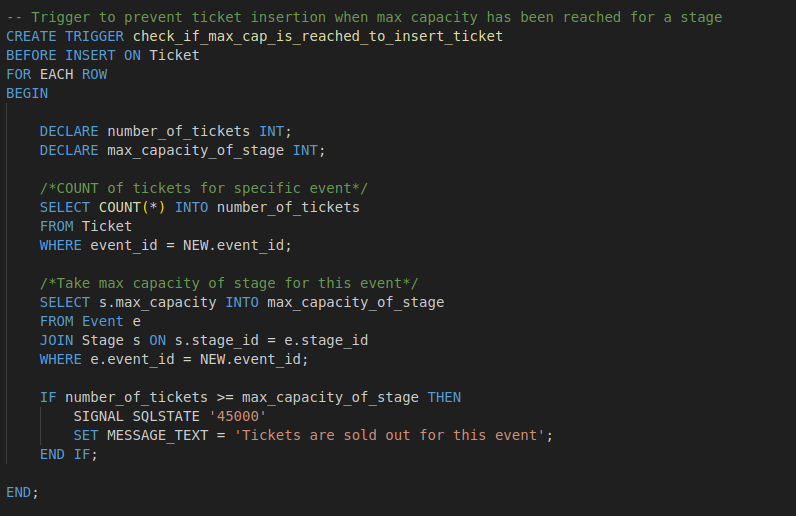
Το trigger trg\_after\_demand\_queue\_insert ενεργοποιείται μετά την εισαγωγή (AFTER INSERT) μιας νέας εγγραφής στον πίνακα Demand\_Queue και έχει ως στόχο να διευκολύνει την αυτόματη εκτέλεση μιας μεταπώλησης εισιτηρίου, εφόσον υπάρχει διαθέσιμο αντίστοιχο εισιτήριο στον πίνακα Resale\_Queue. Συγκεκριμένα, αναζητεί ένα διαθέσιμο προς πώληση εισιτήριο που να ταιριάζει με τις προτιμήσεις του αγοραστή (σε event και τύπο εισιτηρίου). Αν εντοπιστεί τέτοιο εισιτήριο, ελέγχεται αν ο πωλητής έχει περισσότερα από ένα εισιτήρια: στην περίπτωση αυτή δημιουργείται νέα εγγραφή στον πίνακα Visitor για τον αγοραστή και γίνεται ανάθεση του εισιτηρίου στον νέο επισκέπτη. Εάν ο πωλητής είχε μόνο το συγκεκριμένο εισιτήριο, τότε ενημερώνονται τα στοιχεία του επισκέπτη με εκείνα του αγοραστή. Παράλληλα, καταγράφεται η συναλλαγή στο Resale\_Log, ενώ ενημερώνεται η κατάσταση της αντίστοιχης εγγραφής στο Resale\_Queue ως ολοκληρωμένη (status = TRUE) και διαγράφονται τα στοιχεία του αγοραστή από τον πίνακα Resale\_Buyer.



Το trigger check\_if\_artist\_is\_solo\_blngs\_to ενεργοποιείται πριν την εισαγωγή (BEFORE INSERT) μιας νέας εγγραφής στον πίνακα Belongs\_to και έχει ως στόχο να εξασφαλίσει ότι μόνο καλλιτέχνες που δεν είναι δηλωμένοι ως solo μπορούν να ενταχθούν σε κάποιο συγκρότημα (band). Συγκεκριμένα, κατά την εισαγωγή, ελέγχεται η τιμή του πεδίου is\_solo από τον πίνακα Artist για τον καλλιτέχνη που επιχειρείται να συσχετιστεί. Αν διαπιστωθεί ότι ο καλλιτέχνης είναι solo (is\_solo = TRUE), η εισαγωγή ακυρώνεται μέσω κατάλληλου μηνύματος σφάλματος, αποτρέποντας έτσι τη δημιουργία μιας μη έγκυρης συσχέτισης.



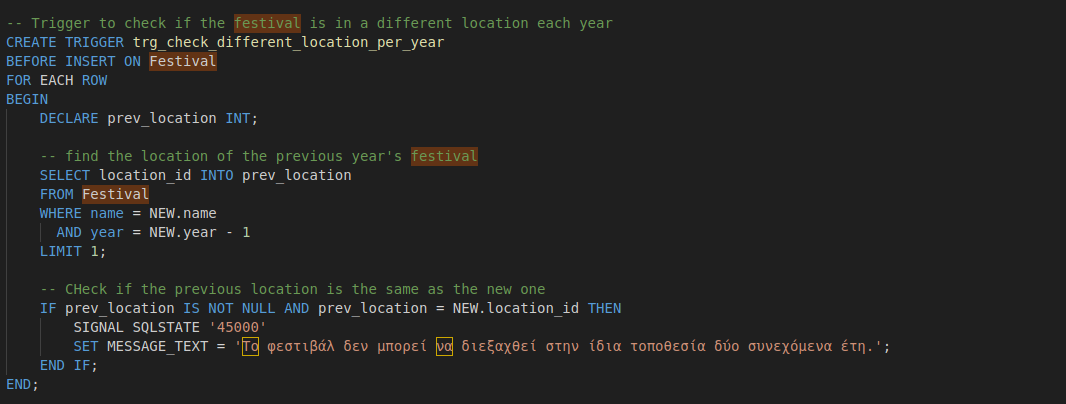
Το trigger check\_if\_max\_cap\_is\_reached\_to\_insert\_ticket ενεργοποιείται πριν την εισαγωγή (BEFORE INSERT) μιας νέας εγγραφής στον πίνακα Ticket και έχει ως στόχο να αποτρέψει την έκδοση εισιτηρίων πέραν της καθορισμένης χωρητικότητας της σκηνής όπου πραγματοποιείται το αντίστοιχο event. Συγκεκριμένα, κατά την εισαγωγή, το trigger υπολογίζει τον συνολικό αριθμό των εισιτηρίων που έχουν ήδη εκδοθεί για το συγκεκριμένο event και αντλεί τη μέγιστη χωρητικότητα της σκηνής μέσω συσχέτισης με τον πίνακα Stage. Εφόσον διαπιστωθεί ότι ο αριθμός των υφιστάμενων εισιτηρίων είναι ίσος ή μεγαλύτερος από τη χωρητικότητα της σκηνής, το trigger ακυρώνει την εισαγωγή με κατάλληλο μήνυμα σφάλματος, αποτρέποντας έτσι την υπέρβαση του ορίου συμμετοχής και διασφαλίζοντας τη συμμόρφωση με τους περιορισμούς της υποδομής του φεστιβάλ.



Το trigger check\_performer\_availability\_on\_insert ενεργοποιείται πριν από την εισαγωγή (BEFORE INSERT) μιας νέας εγγραφής στον πίνακα Performance και έχει ως σκοπό να διασφαλίσει τη χρονική διαθεσιμότητα του καλλιτέχνη ή του συγκροτήματος τόσο σε επίπεδο ημερήσιου προγράμματος όσο και σε επίπεδο ετήσιας συμμετοχής. Συγκεκριμένα, κατά την εισαγωγή μιας νέας εμφάνισης, το trigger παίρνειτο artist\_id ή band\_id από τον πίνακα Performs και ελέγχει εάν το συγκεκριμένο entity έχει ήδη κάποια άλλη εμφάνιση την ίδια ημέρα σε επικαλυπτόμενες ώρες – σε τέτοια περίπτωση, η εισαγωγή απορρίπτεται με σχετικό μήνυμα σφάλματος. Επιπλέον, προκειμένου να αποφευχθεί η υπερβολική συμμετοχή ενός καλλιτέχνη ή συγκροτήματος στο πρόγραμμα του φεστιβαλ, το trigger ελέγχει αν έχει προηγηθεί συμμετοχή στα τρία αμέσως προηγούμενα έτη (όπως ορίζεται και από εκφώκαι, εφόσον ισχύει, απαγορεύει την εισαγωγή τέταρτης συνεχόμενης εμφάνισης.



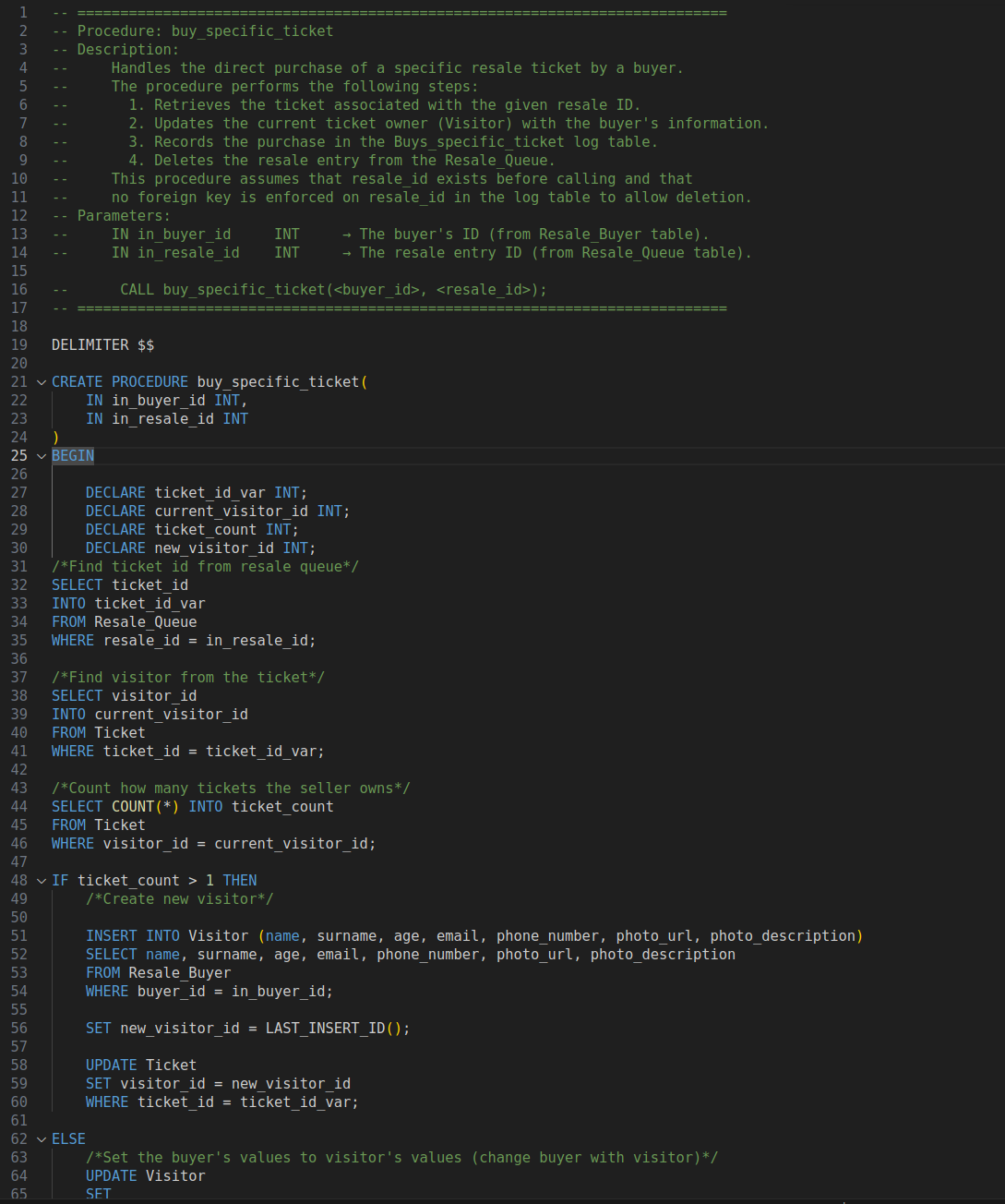
Το trigger trg\_check\_different\_location\_per\_year ενεργοποιείται πριν από την εισαγωγή (BEFORE INSERT) μιας νέας εγγραφής στον πίνακα Festival και έχει ως σκοπό να διασφαλίσει ότι το φεστιβάλ δεν θα διεξαχθεί στην ίδια τοποθεσία για δύο συνεχόμενα έτη. Συγκεκριμένα, κατά την εισαγωγή μιας νέας εγγραφής φεστιβάλ, το trigger αναζητά —με βάση το όνομα (name) και το έτος (year - 1)— την τοποθεσία (location\_id) του φεστιβάλ της προηγούμενης χρονιάς. Εφόσον διαπιστωθεί ότι η τοποθεσία της νέας εγγραφής είναι ίδια με εκείνη του αμέσως προηγούμενου έτους, η εισαγωγή απορρίπτεται με αντίστοιχο μήνυμα σφάλματος. Με τον τρόπο αυτό, ενισχύεται η απαίτηση της εκφώνησης που ορίζει ότι το φεστιβάλ πρέπει να φιλοξενείται κάθε χρόνο σε διαφορετική τοποθεσία.

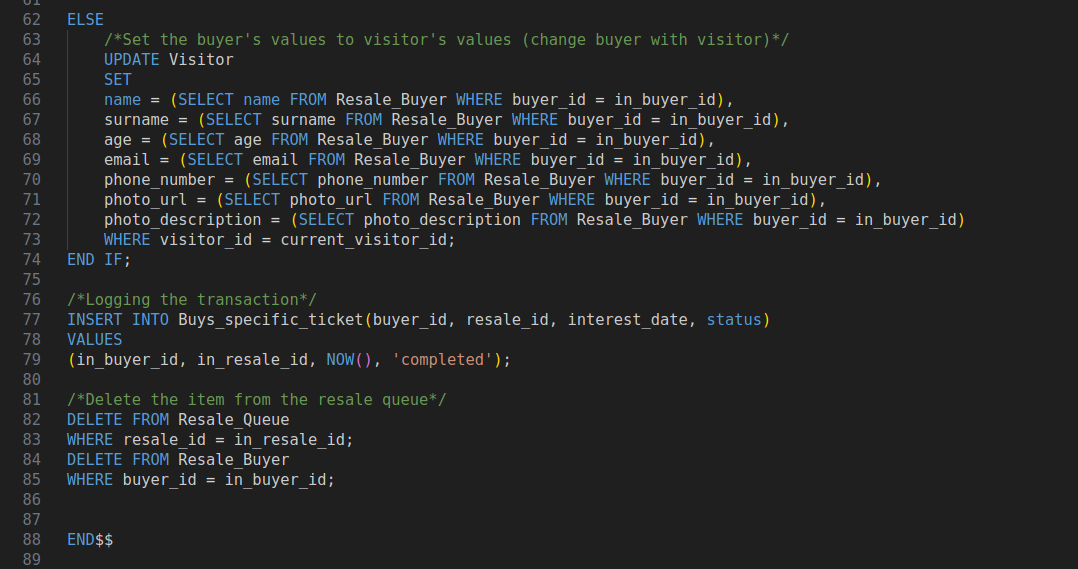


**Στη συνέχεια θα παρουσιαστούν τα procedures που υλοποιήθηκαν για την παρούσα εργασία:**

Η stored procedure buy\_specific\_ticket υλοποιεί τη στοχευμένη αγορά ενός συγκεκριμένου εισιτηρίου που έχει διατεθεί προς μεταπώληση από επισκέπτη του φεστιβάλ. Δέχεται ως παραμέτρους το buyer\_id (από τον πίνακα Resale\_Buyer) και το resale\_id (από τον πίνακα Resale\_Queue) και εκτελεί μία πλήρως αυτοματοποιημένη διαδικασία μεταβίβασης του εισιτηρίου στον νέο κάτοχο. Πιο συγκεκριμένα:

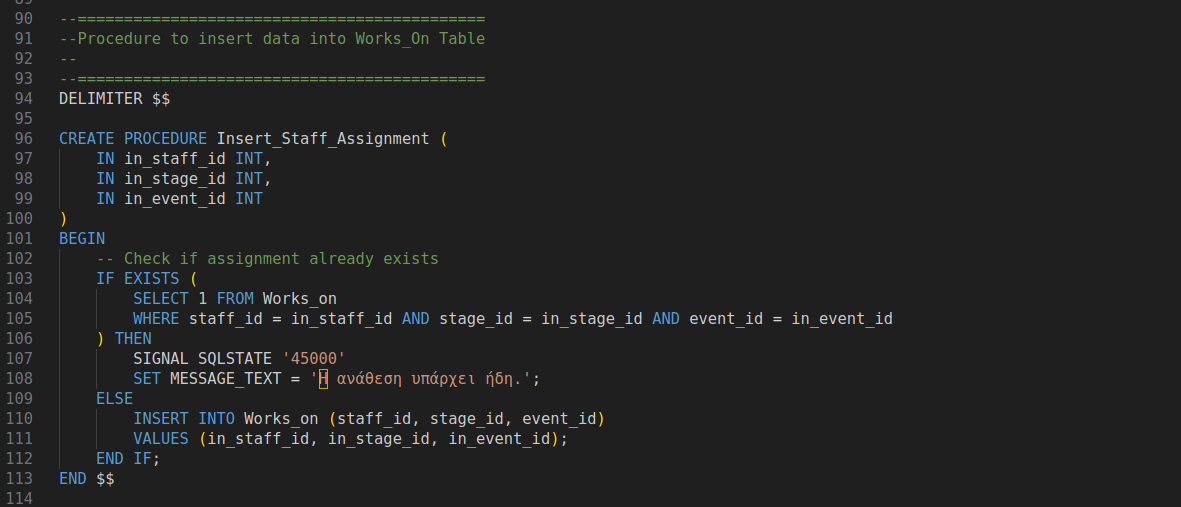
1. Εντοπίζει το ticket\_id του εισιτηρίου που συνδέεται με την αντίστοιχη εγγραφή στο Resale\_Queue.
2. Εντοπίζει τον τρέχοντα κάτοχο του εισιτηρίου και μετρά πόσα εισιτήρια κατέχει.
3. Αν ο πωλητής έχει πάνω από ένα εισιτήρια, δημιουργείται νέος επισκέπτης (Visitor) με βάση τα στοιχεία του αγοραστή και το εισιτήριο μεταβιβάζεται σε αυτόν.
4. Αν ο πωλητής κατέχει μόνο αυτό το εισιτήριο, τότε ενημερώνονται τα προσωπικά στοιχεία του υπάρχοντος Visitor με τα στοιχεία του αγοραστή (ταυτότητα παραμένει).
5. Καταγράφεται η συναλλαγή στον πίνακα Buys\_specific\_ticket ως ολοκληρωμένη (status = 'completed').
6. Διαγράφεται η εγγραφή από το Resale\_Queue και τα στοιχεία του αγοραστή από τον πίνακα Resale\_Buyer.





Η stored procedure Insert\_Staff\_Assignment έχει ως στόχο την ασφαλή και μοναδική καταχώριση μιας ανάθεσης προσωπικού στον πίνακα Works\_on, ο οποίος καταγράφει ποια μέλη προσωπικού εργάζονται σε ποια σκηνή και εκδήλωση. Η διαδικασία δέχεται ως παραμέτρους το αναγνωριστικό του εργαζομένου (staff\_id), της σκηνής (stage\_id) και του event (event\_id) και ακολουθεί τα εξής βήματα:

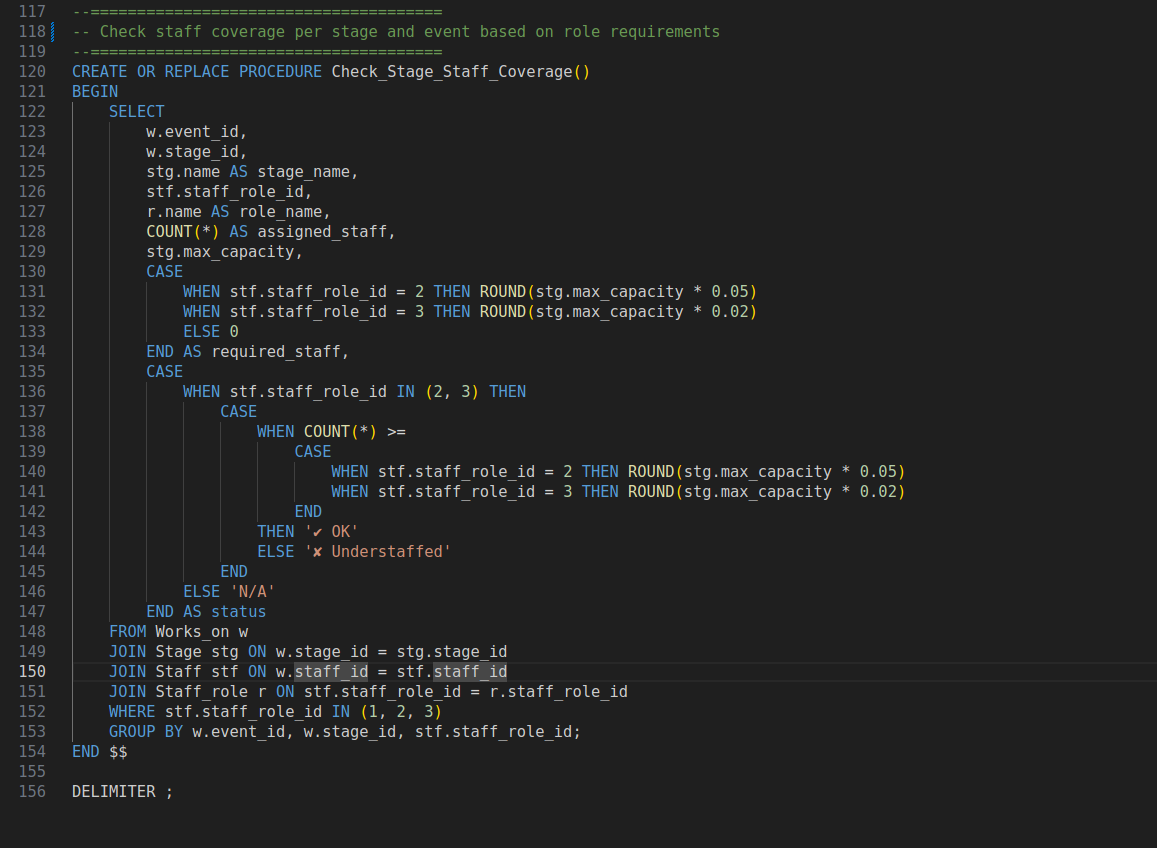
1. Έλεγχος ύπαρξης: Πριν από την εισαγωγή, η procedure ελέγχει αν υπάρχει ήδη εγγραφή με τα ίδια στοιχεία στο Works\_on. Αν υπάρχει, διακόπτει την εκτέλεση και επιστρέφει μήνυμα σφάλματος: "Η ανάθεση υπάρχει ήδη."
2. Εισαγωγή: Αν δεν υπάρχει τέτοια εγγραφή, πραγματοποιείται εισαγωγή της νέας ανάθεσης στον πίνακα.



Η stored procedure Check\_Stage\_Staff\_Coverage έχει ως στόχο τον έλεγχο κάλυψης προσωπικού ανά σκηνή και εκδήλωση, σύμφωνα με τις ελάχιστες απαιτήσεις που ορίζονται στην εκφώνηση της εργασίας. Η procedure συγκεντρώνει στοιχεία από τις αναθέσεις του πίνακα Works\_on και παρέχει αναλυτική εικόνα για το αν έχουν καλυφθεί επαρκώς οι απαιτούμενοι ρόλοι, με βάση τη χωρητικότητα της κάθε σκηνής.

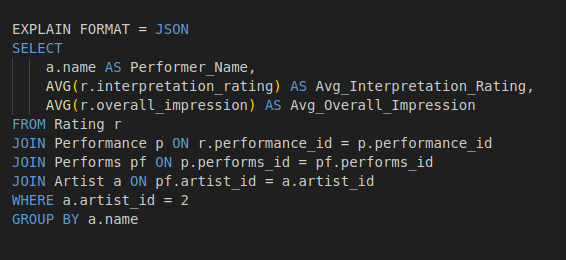
Συγκεκριμένα, η διαδικασία:

1. Ομαδοποιεί τις αναθέσεις κατά event\_id, stage\_id και staff\_role\_id.
2. Υπολογίζει πόσα μέλη προσωπικού έχουν ανατεθεί σε κάθε ρόλο.
3. Αντλεί τη μέγιστη χωρητικότητα της σκηνής (max\_capacity) και, για ρόλους ασφαλείας (staff\_role\_id = 2) και βοηθητικού προσωπικού (staff\_role\_id = 3), υπολογίζει τον απαιτούμενο αριθμό εργαζομένων (αντίστοιχα 5% και 2% της χωρητικότητας).
4. Εμφανίζει για κάθε περίπτωση αν η σκηνή είναι επαρκώς στελεχωμένη (✔ OK) ή υποστελεχωμένη (✘ Understaffed).
5. Για άλλους ρόλους (π.χ. τεχνικούς, staff\_role\_id = 1), η απαίτηση δεν αξιολογείται (καθώς δεν υπήρχε περιορισμός στην εκφώνηση) και δηλώνεται ως N/A

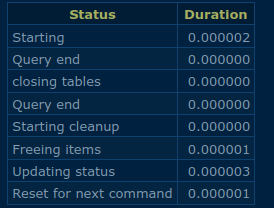


***Query 4***

Για το query 4 πραγματοποιήσαμε την παρακάτω υλοποίηση



Από αυτή λαμβάνουμε χρησιμοποιώντας EXPLAIN FORMAT = JSON και SET profiling = 1; με SHOW PROFILE τα παρακάτω αποτελέσματα, τα οποία μας εξηγούν τις επιλογές που έκανε ο optimizer της mariadb με βάση τα indexes που έχουμε δημιουργήσει και τα primaries. Στη συνέχεια θα επιλέξουμε να χρησιμοποιήσουμε διαφορετικό query plan και να κάνουμε force indexes εμείς οι ίδιοι.

Figure 1: Query 4 without force index traces

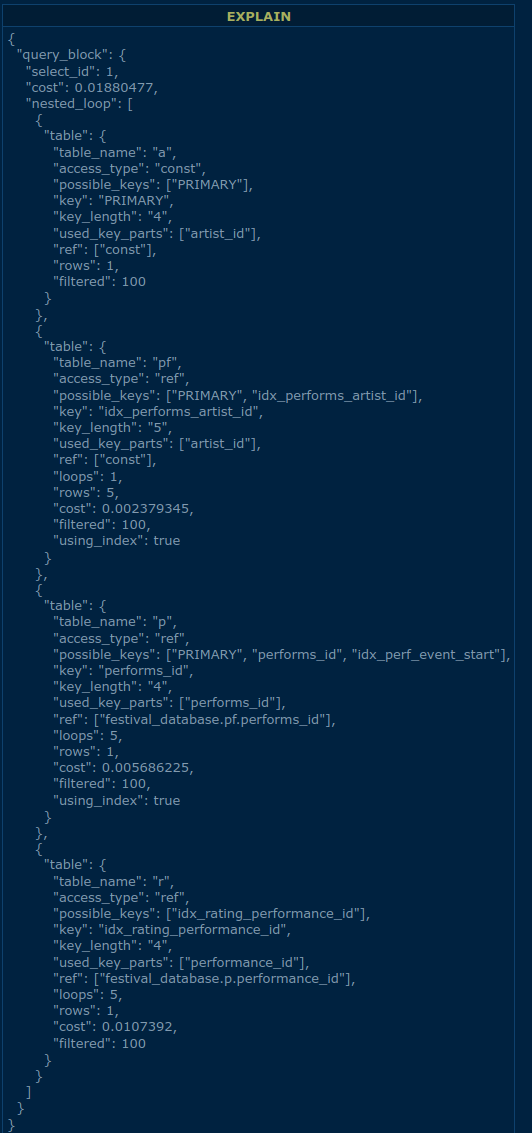
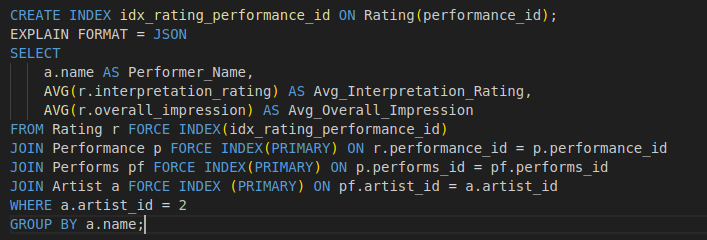
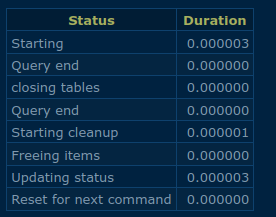


Figure 2: Query 4 without force index EXPLAIN

Figure 3: Query 4 new query plan with force indexes

Με το νέο query plan λαμβάνουμε τα παρακάτω αποτελέσματα. Βλέπουμε ότι οι επιλογές που επιβάλαμε εμείς στον optimizer είχανε αρνητικό αντίκτυπο με το συνολικό χρόνο να αυξάνεται και το κόστος για την εκτέλεση του query να γίνεται μεγαλύτερο κατά 8 φορές. Από αυτό προκύπτει εύκολα ότι ο optimizer της mariadb κάνει αρκετά καλή δουλειά όσον αφορά το planning για την εκτέλεση ενός query και θέλει αρκετή προσοχή όταν ορίζουμε εμείς οι ίδιοι ένα συγκεκριμένο μονοπάτι για εκτέλεση.

Figure : Query 4 force index traces

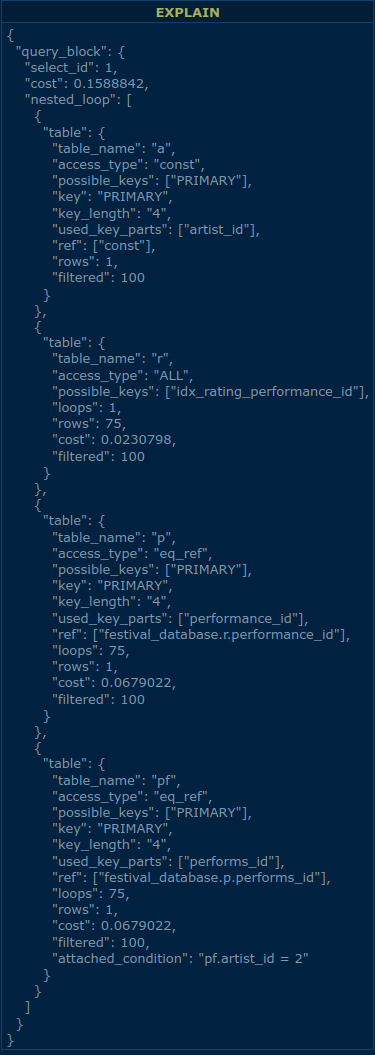
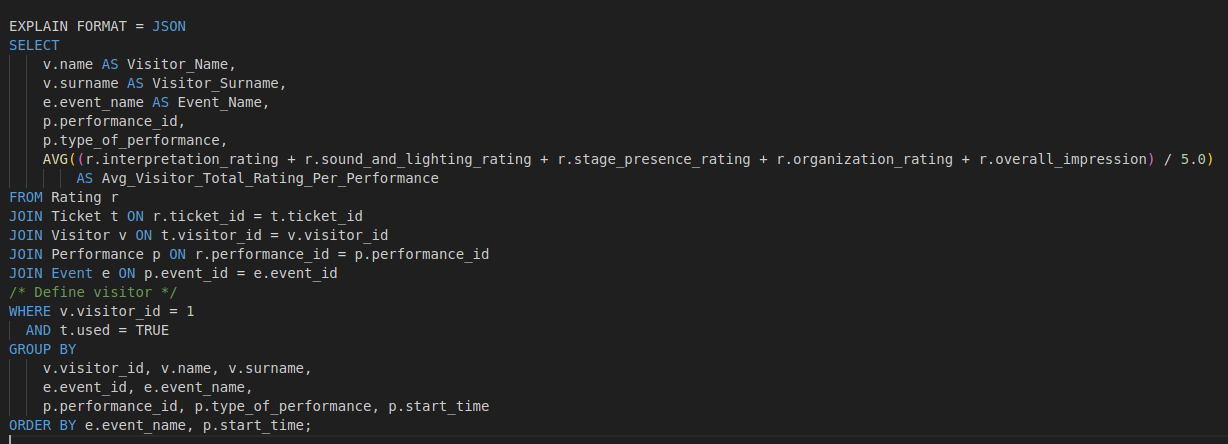


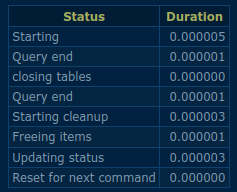
Figure 5: Query 4 with force index EXPLAIN

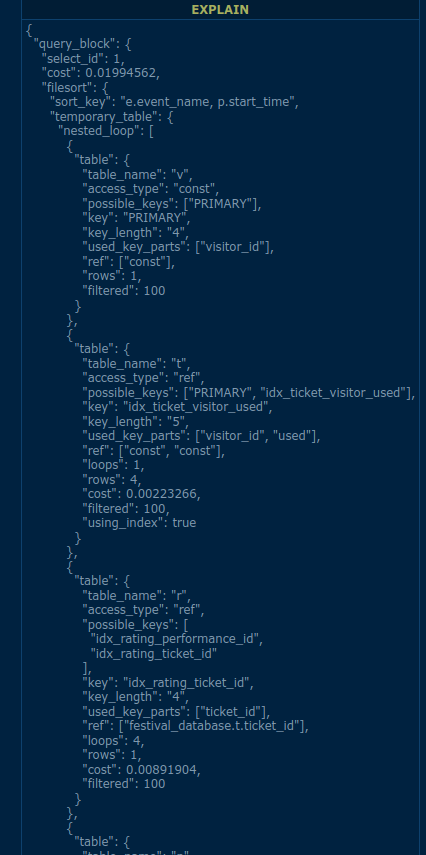
***Query 6***

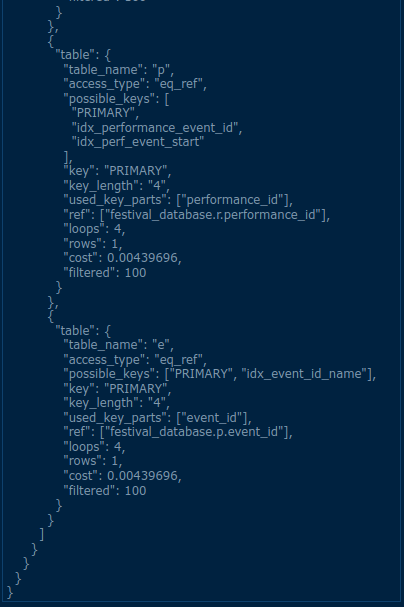
Για το query 6 πραγματοποιήσαμε την εξής υλοποίηση

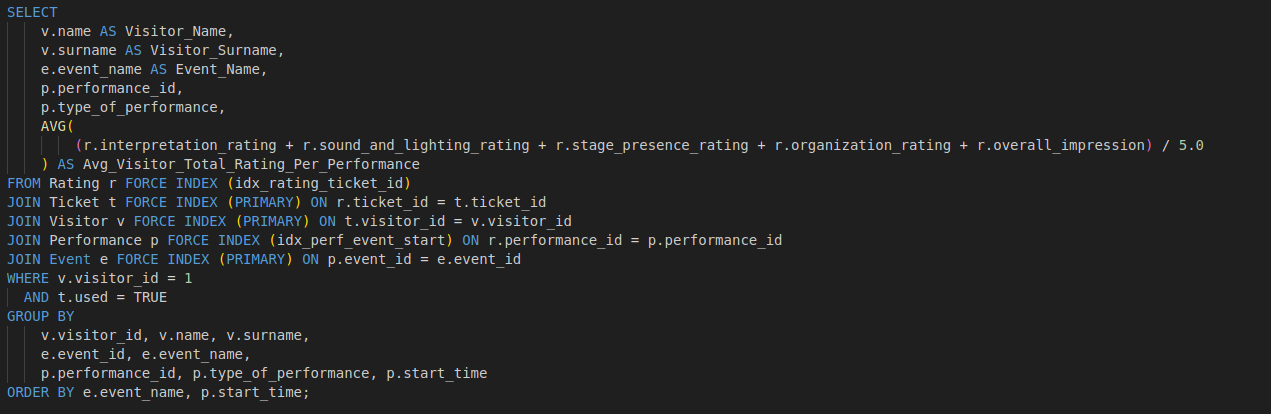
Figure 6: Query 6 implementation without force index

Από αυτή λαμβάνουμε χρησιμοποιώντας EXPLAIN FORMAT = JSON και SET profiling = 1; με SHOW PROFILE τα παρακάτω αποτελέσματα, τα οποία μας εξηγούν τις επιλογές που έκανε ο optimizer της mariadb με βάση τα indexes που έχουμε δημιουργήσει και τα primaries. Στη συνέχεια θα επιλέξουμε να χρησιμοποιήσουμε διαφορετικό query plan και να κάνουμε force indexes εμείς οι ίδιοι.

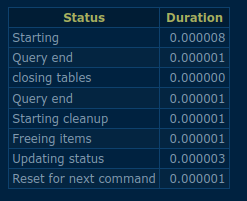
Figure 7: Query 6 without force index traces

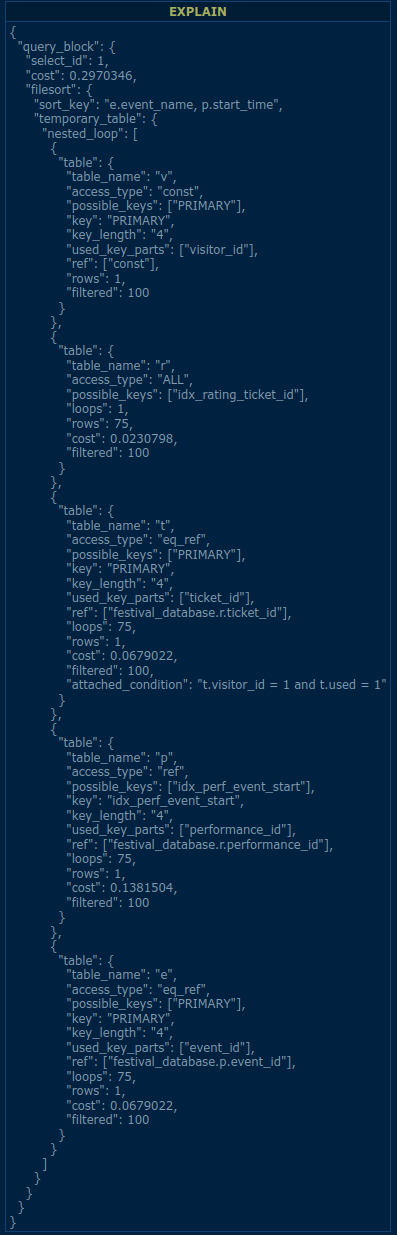


Figure 8: Query 6 without force index EXPLAIN

Figure 9: Query 6 New query plan with force indexes

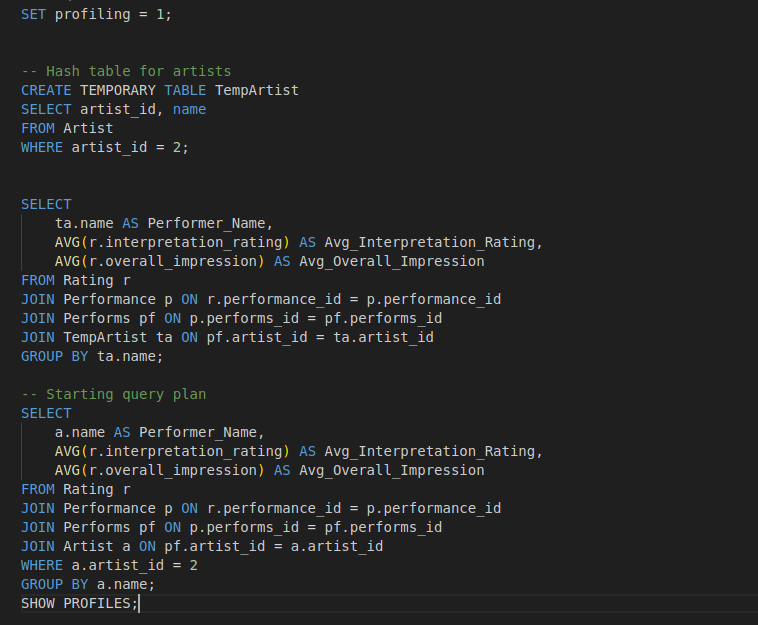
Με το νέο query plan λαμβάνουμε τα παρακάτω αποτελέσματα. Βλέπουμε ότι οι επιλογές που επιβάλαμε εμείς στον optimizer είχανε αρνητικό αντίκτυπο με το συνολικό χρόνο να αυξάνεται και το κόστος για την εκτέλεση του query να γίνεται μεγαλύτερο κατά 15 περίπου φορές. Από αυτό προκύπτει εύκολα ότι ο optimizer της mariadb κάνει αρκετά καλή δουλειά όσον αφορά το planning για την εκτέλεση ενός query και θέλει αρκετή προσοχή όταν ορίζουμε εμείς οι ίδιοι ένα συγκεκριμένο μονοπάτι για εκτέλεση.

Figure 10: Query 6 with force index traces

Figure 11: Query 6 with force index traces EXPLAIN

Στη συνέχεια και με τα δύο queries θα δοκιμάσουμε τη λογική του hash join για να δούμε πως εφαρμόζεται σε αυτά.

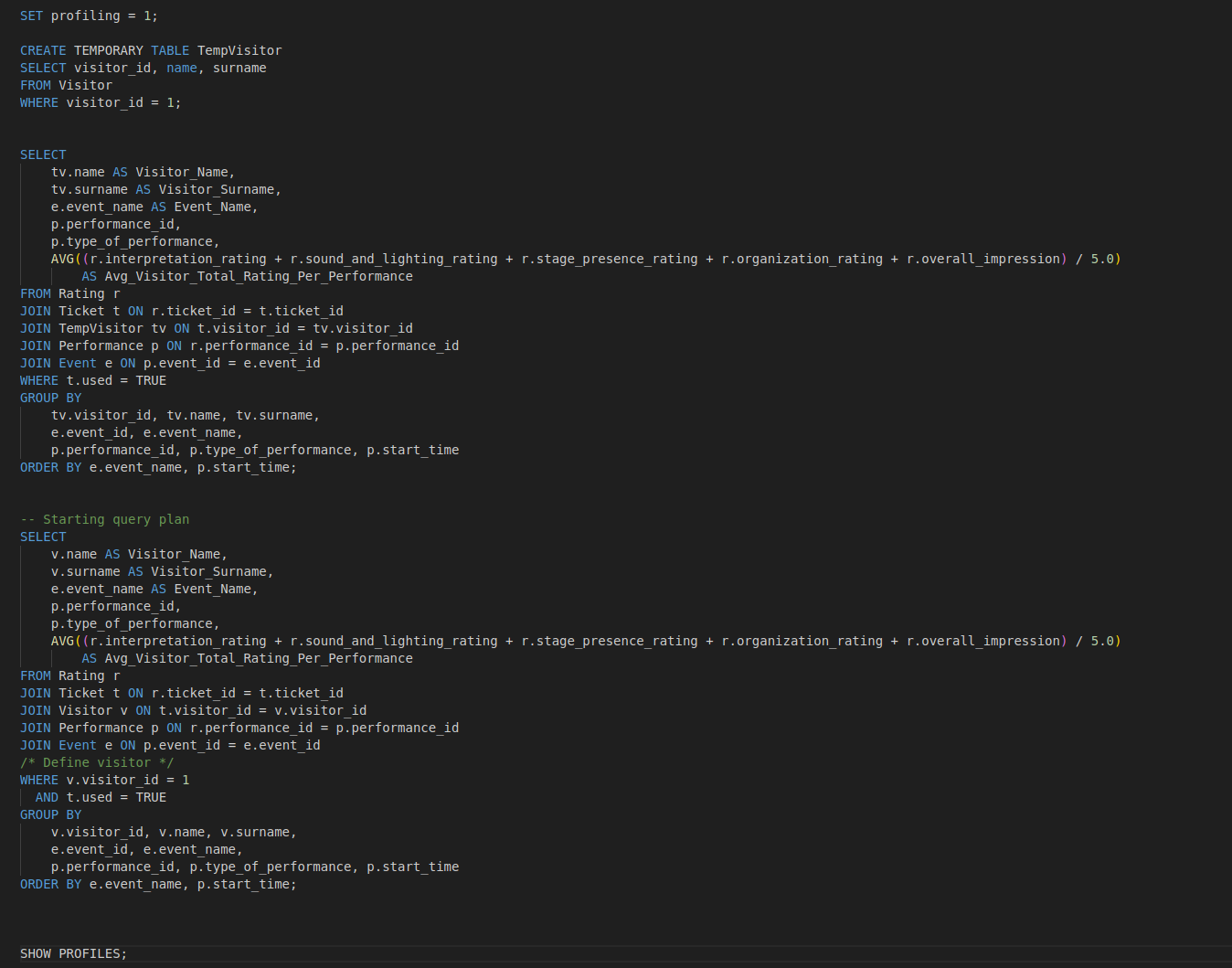
Αρχικά, προσπαθήσαμε να ορίσουμε hash\_join=on ή γενικότερα να επιβάλουμε τον optimizer να κάνει χρήση hash join αλλά δεν γινότανε επιτρεπτό. Για τον λόγο αυτό υλοποιήσαμε μία τεχνική hash join χειροκίνητα παρακάτω, όπου δημιουργήσαμε ένα temporary table όπου βάλαμε το artist\_id που θέλουμε για join key στη συνέχεια. Στην συνέχεια συμπληρώσαμε με σχεδόν ίδιο το αρχικό query απλά στο σημείο όπου προηγουμένως οριζόταν το artist\_id, τώρα γίνεται αναφορά στον προσωρινό πίνακα. Έπειτα κάνουμε τη σύγκριση με το αρχικό query μας και παρατηρούμε τα αποτελέσματα στους χρόνους εκτέλεσης:



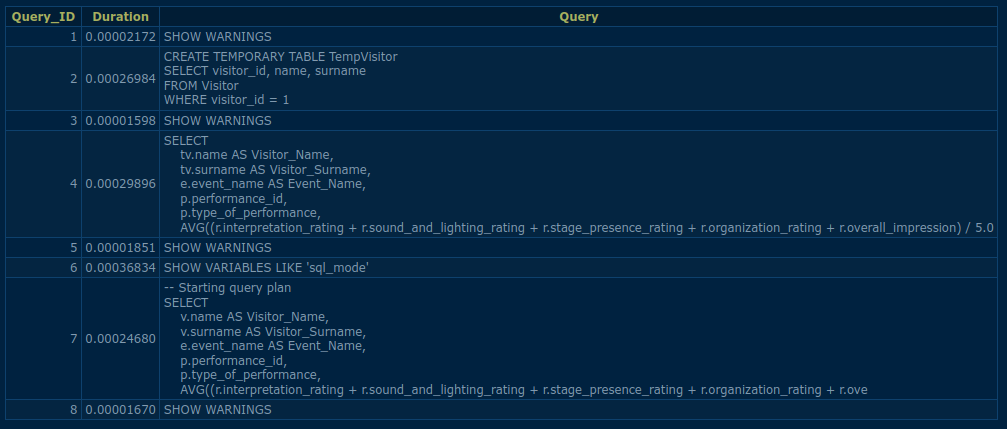


Από αυτή τη σύγκριση βλέπουμε ότι το query μας που έκανε χρήση του tempArtist table είναι οριακά πιο γρήγορο (σχεδόν αμελητέο) από το απευθείας join. Παρόλα αυτά αν υπολογίσουμε και το κόστος κατασκευής του πίνακα τότε το απευθείας join είναι συνολικά πιο γρήγορο. Ωστόσο, για να μπορούσε να γίνει καλύτερη σύγκριση με τη hash join λογική θα έπρεπε να είχαμε πολλά περισσότερα entries στους πίνακές μας.

Αντίστοιχα τώρα για το query 6 εφαρμόζουμε την ίδια λογική φτιάχνοντας έναν temporary table για τον visitor μας και ακολουθήσαμε την ίδια λογική με προηγουμένως.



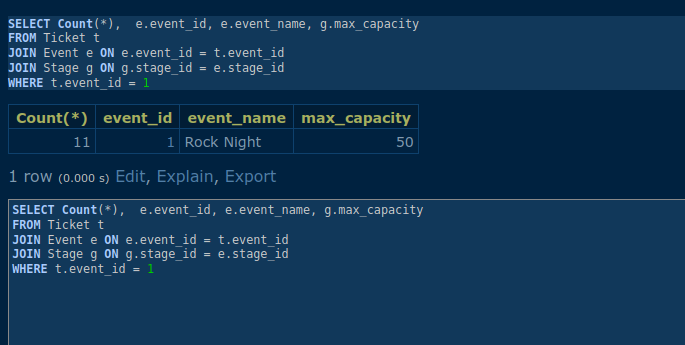
Τα αποτελέσματα εμφανίζονται παρακάτω:



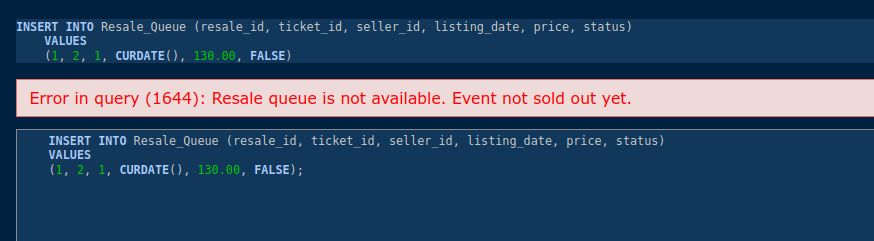
Ομοίως με πριν το συνολικό κόστος είναι μεγαλύτερο από το απευθείας join.

***Resale Queue***

Θα παρουσιάσουμε τώρα την λειτουργικότητα της resale queue. Βλέπουμε για το event με event\_id = 1 ότι τα συνολικά εισιτήρια είναι 11 και το max capacity του stage είναι 50.



Στην προσπάθειά μας να κάνουμε εισαγωγή στη Resale queue για το συγκεκριμένο event λαμβάνουμε το παρακάτω μήνυμα. Σε περίπτωση εισαγωγής και διαφορετικής τιμής από την αρχική θα είχαμε πάλι αντίστοιχο error



Θα πάμε τώρα και θα φτιάξουμε ένα δοκιμαστικό σενάριο

Ξεκινάμε κάνοντας νέο δοκιμαστικό stage

[INSERT](https://mariadb.com/kb/en/insert/) INTO [Stage](http://localhost:8080/?server=mariadb&username=root&db=festival_database&table=Stage) (name, description, max\_capacity, infos\_technical\_equipment, photo\_url, photo\_description)  
VALUES  
('test-stage','something', 3, 'fs', 'gds','sdg');

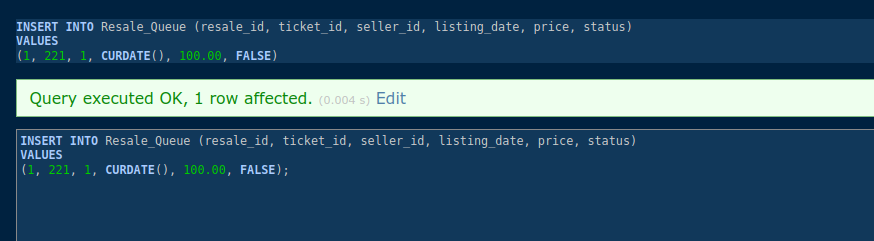
Στη συνέχεια φτιάχνουμε ένα event για αυτό το stage

[INSERT](https://mariadb.com/kb/en/insert/) INTO [Event](http://localhost:8080/?server=mariadb&username=root&db=festival_database&table=Event) (event\_id, festival\_id, stage\_id, event\_name, event\_date, duration, photo\_url, photo\_description)  
VALUES   
(41, 1, 31, 'test-event', '2025-09-16', 3,'sd','sd');

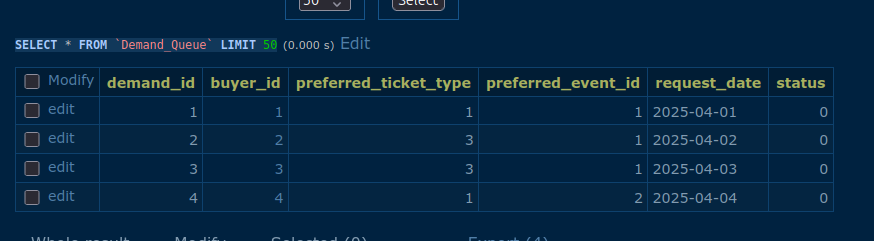
Δημιουργούμε 3 εισιτήρια

[INSERT](https://mariadb.com/kb/en/insert/) INTO [Ticket](http://localhost:8080/?server=mariadb&username=root&db=festival_database&table=Ticket) (ticket\_id, event\_id, visitor\_id, ticket\_type\_id, purchase\_date, price, payment\_method\_id, ean\_code, used, photo\_url, photo\_description)  
VALUES  
(221, 41, 1, 2,'2025-06-02 14:30:00', 100, 1, '1234567890123', FALSE, 'ASF', 'ASF'),  
(222, 41, 2, 2,'2025-06-02 14:30:00', 100, 1, '1234567890123', FALSE, 'ASF', 'ASF'),  
(223, 41, 3, 3,'2025-06-02 14:30:00', 100, 1, '1234567890123', FALSE, 'ASF', 'ASF');

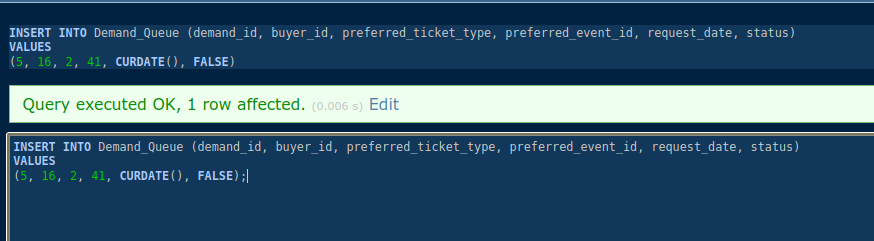
Στη συνέχεια κάνουμε εισαγωγή στη resale queue και βλέπουμε ότι εισάγεται κανονικά μιας και έχει γίνει sold out το event

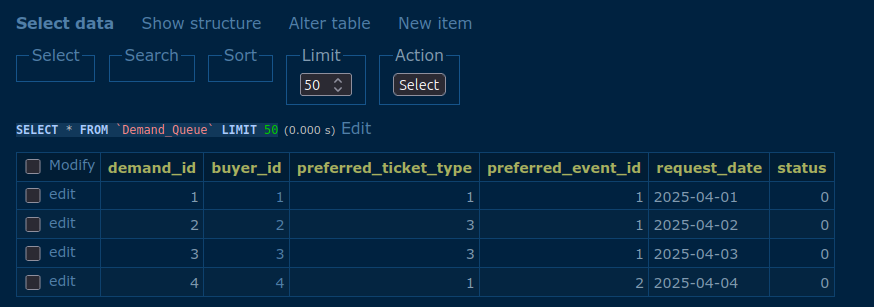


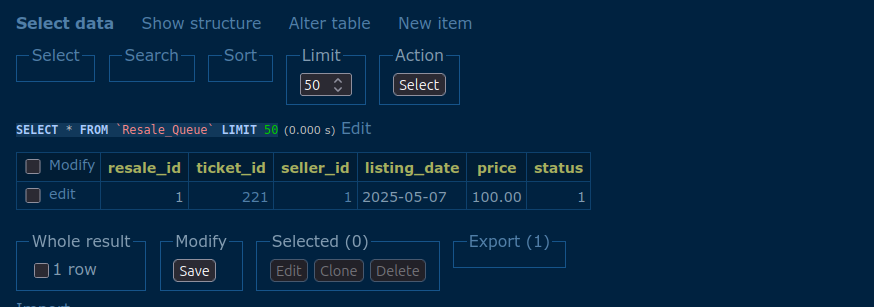
Θα πάμε τώρα να κάνουμε εισαγωγή στην demand queue από τον resale buyer με resale\_buyer\_id = 16 και όνομα Apostolos Venieris με age = 29, email = [apostolos.v@example.com](mailto:apostolos.v@example.com) και τηλέφωνο +306925678902

Η demand queue έχει ήδη αυτές τις εισαγωγές  


και έχουμε

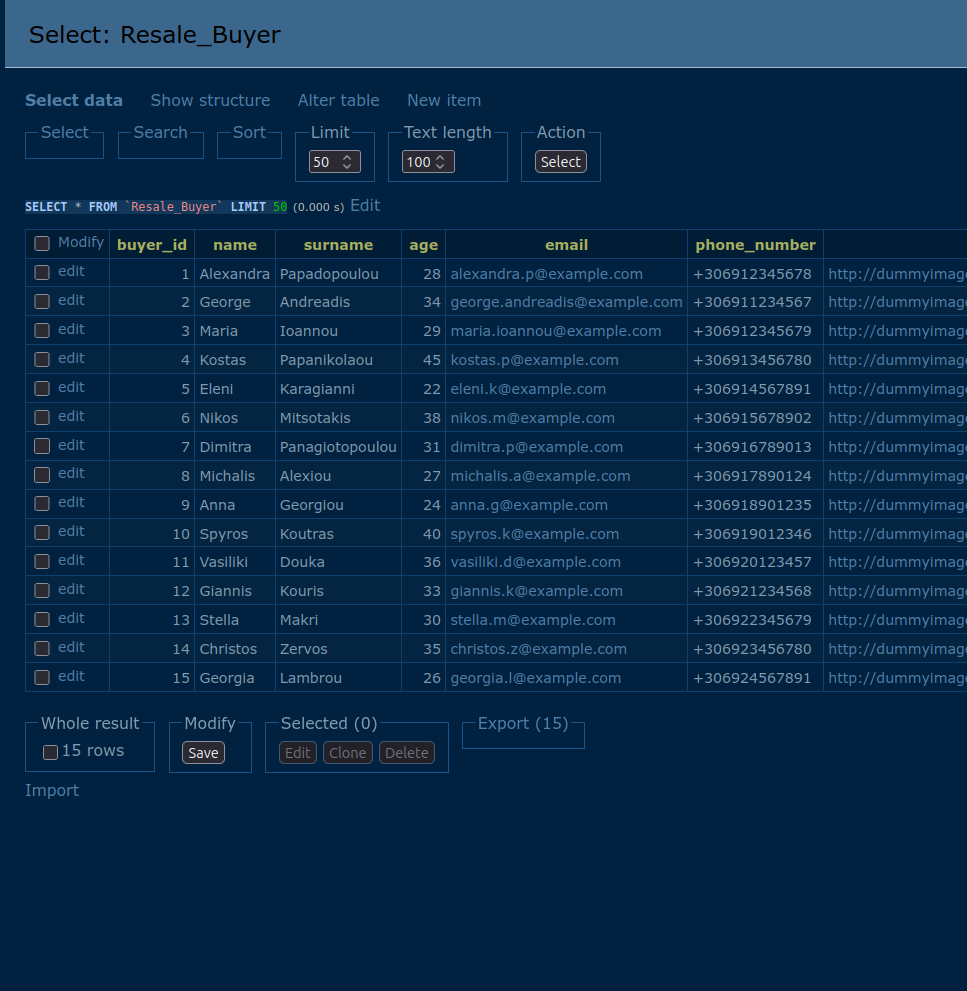


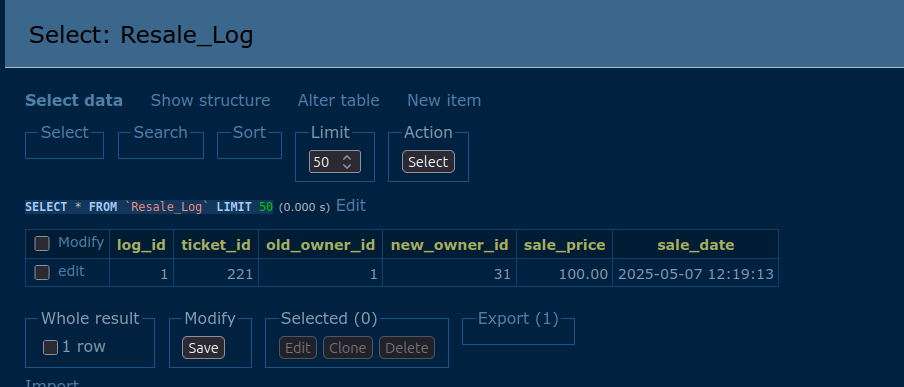
Ωστόσο αν πάμε να ξανά ελέγξουμε την Demand\_Queue δεν θα βρούμε εγγραφή και αυτό γιατί λόγω του trigger που έχουμε υλοποιήσει μόλις βρέθηκε αντιστοιχία έγινε το transaction.

Βλέπουμε στη συνέχεια στην Resale Queue ότι πλέον το status της εισαγωγής είναι 1 (προηγουμένως ήταν 0 δηλαδη FALSE )και μπαίνει σε κατάσταση που δεν μπορεί να πραγματοποιηθεί

Ταυτόχρονα κοιτάμε τους visitors και βλέπουμε πλέον να υπάρχει καινούργιος visitor με τα στοιχεία του resale buyer.

Ενώ όταν κοιτάξουμε στον πίνακα με τους resale buyers πλέον δεν υπάρχει ο αγοραστής μας



Τέλος στο trigger έχουμε βάλει να καταγράφει τη συναλλαγή σε ένα log όπως και φαίνεται με την ώρα που πραγματοποιήθηκε και τα στοιχεία της συναλλαγής.

Αντίστοιχα υλοποιήθηκε και το procedure για κατευθείαν αγορά ενός υπαρκτού εισιτηρίου από κάποιον resale buyer που δεν είχε προηγουμένως δηλώσει ενδιαφέρον. Επιπλέον, με την ίδια λογική αν υπάρχει εγγραφή στο Demand\_Queue που δεν έχει καλυφθεί και εισάγεται μία που την καλύπτει, γίνεται πάλι το transaction με τον ίδιο τρόπο. Με τον τρόπο αυτό επιτυγχάνεται η λειτουργία FIFO που ζητήθηκε από την εκφώνηση. Να σημειωθεί ότι εισήχθει νέος visitor μετά την αγορά, διότι ο visitor με visitor\_id = 1 έχει ήδη και άλλα εισιτήρια στην κατοχή του. Σε περίπτωση που δεν είχε, έχει υλοποιηθεί στο trigger η αντικατάσταση στο visitor\_id = 1 των στοοιχείων του resale\_buyer.

**ΠΑΡΑΔΟΧΕΣ**

* Στο πλαίσιο του σχεδιασμού του συστήματος, θεωρήθηκε ότι το Pulse University Festival μπορεί να διοργανώνει πολλαπλα φεστιβάλ, κάθε ένα με το δικό του έτος, τοποθεσία και χαρακτηριστικά. Για το λόγο αυτό και δεδομένου ότι θεωρούμε την έναρξη της εφαρμογής από το 2025, τα φεστιβάλ του Pulse University αρχίζουν από το έτος 2025. Επομένως θα θεωρήσουμε ως συμμετοχές, αυτές για το έτος του 2025 θεωρώντας ότι θα είναι αυτό που τελείωσε μετά το πέρας του χρόνου.
* Θεωρούμε ότι ένας staff member δεν μπορεί να δουλέψει σε άλλο event την ίδια μέρα. Έτσι γίνονται κατάλληλα οι εισαγωγές και αντίστοιχα υπάρχει περιορισμός με κατάλληλο trigger
* Για το προσωπικό κρατήσαμε τους περιορισμούς που ζητούνται, ενώ για το τεχνικό προσωπικό δεν ορίσαμε κάποιο κατώτατο όριο. Για τον λόγο αυτό σε αντίστοιχο query ή κάποια μοργή παρουσίασης μπορεί να εμφανιστεί ότι μία σκηνή έχει 0 τεχνικό προσωπικό (δεν χρειάζεται)
* Για οποιδήποτε query που ζητούσε παρουσίαση δεδομένων για καλλιτέχνες, οι απαντήσεις υλοποιήθηκαν να παρουσιάζουν δεδομένα για τους artist. Σε περίπτωση που ζητούνταν και band ο τρόπος υλοποίησης είναι παρόμοιος αν όχι ίδιος
* Καλύφθηκαν όλες οι απαιτήσεις της εκφώνησης είτε από περιορισμούς είτε από αντίστοιχα triggers

Front End

Το login page της σελίδας μας είναι το παρακάτω και μπορεί κάποιος να συνδεθεί με username: root και password: root

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, σχεδίαση

Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.

Στη συνέχεια μετά τη σύνδεση προχωράει και βλέπει τα αντίστοιχα queries που υλοποιήθηκαν.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, αριθμός

Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.

Ενώ αν πατήσει σε κάποιο query θα δει κάτι σαν το παρακάτω:

Εικόνα που περιέχει κείμενο, ηλεκτρονικές συσκευές, στιγμιότυπο οθόνης, αριθμός

Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.