

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ  
ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

# ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ

Χρήστος Ηλιακόπουλος el20233

Παναγιώτης Τσακλάνος el18937

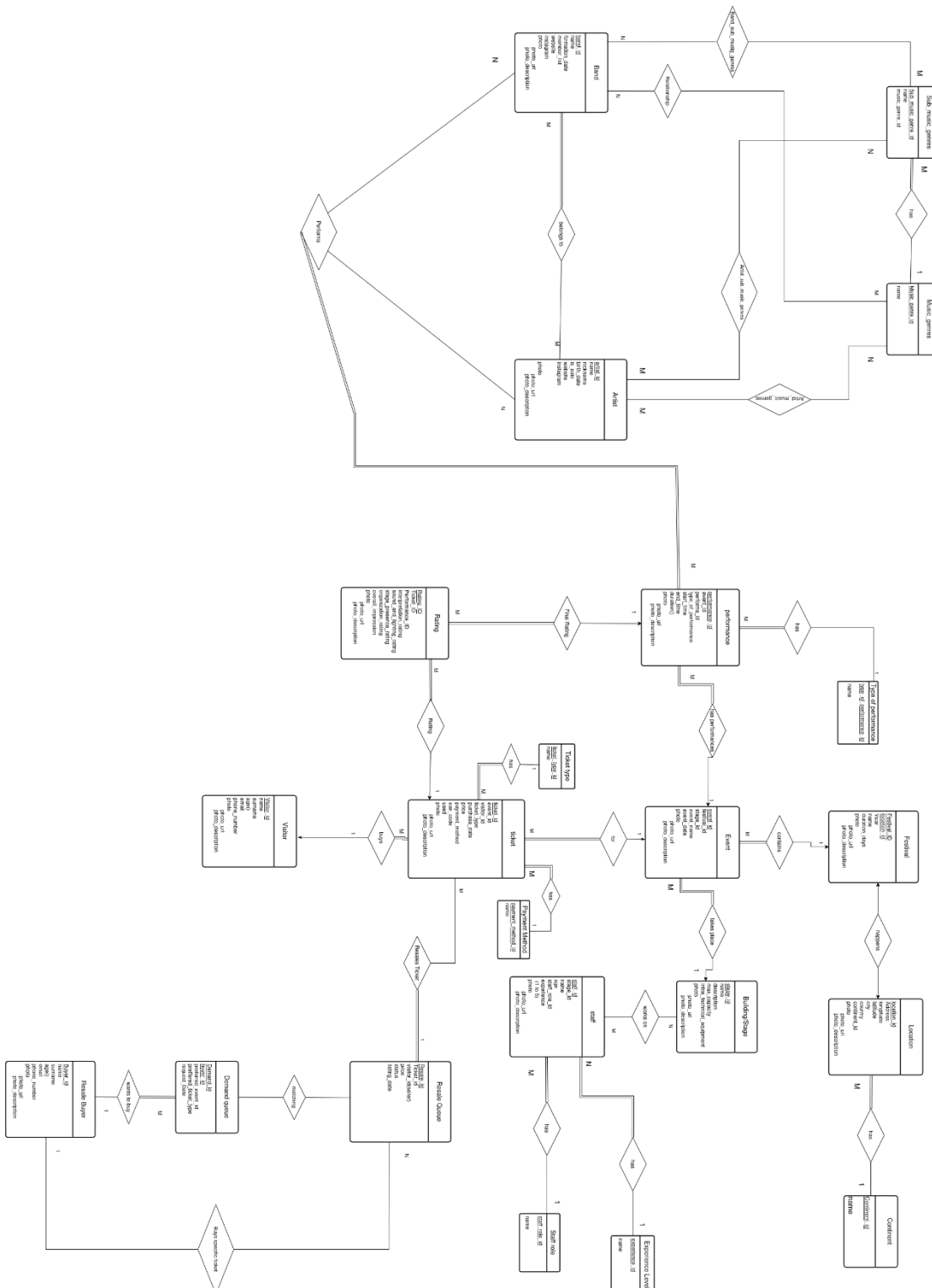
Δημήτρης Τζέλλος el21623

## **ΕΙΣΑΓΩΓΗ**

Η παρούσα εργασία παρουσιάζει τη σχεδίαση και την υλοποίηση μιας βάσης δεδομένων για το διεθνές φεστιβάλ Pulse University. Μας ζητήθηκε ο σχεδιασμός και η υλοποίηση 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, φωτογραφία 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 χωρητικότητα, 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 φυσικά 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 experience\_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 των 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 ιστοσελίδα 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 υποείδη του κάθε καλλιτέχνη θα αναλυθεί παρακάτω

Η οντότητα Band αναφέρει 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 επισκέπτη 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 αναγνωριστικό 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 συνδεεται 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 event 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 αναγνωριστικό του αγοραστή, τον προτιμώμενο τύπο εισιτηρίου και το προτιμώμενο 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 ξεπερασθεί, η εισαγωγή απορρίπτεται και επιστρέφεται σχετικό μήνυμα σφάλματος.



```

/*Trigger that checks the limit of VIP tickets for a stage*/
CREATE TRIGGER trg_check_vip_tickets
BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
    DECLARE vip_limit INT;
    DECLARE current_vip_tickets INT;
    DECLARE stage_capacity INT;
    DECLARE stage_id INT;

    IF NEW.ticket_type_id = 1 THEN

        -- Get the stage_id from the event
        SELECT e.stage_id INTO stage_id
        FROM Event e
        WHERE e.event_id = NEW.event_id;

        -- Get the max capacity of the stage
        SELECT s.max_capacity INTO stage_capacity
        FROM Stage s
        WHERE s.stage_id = stage_id;

        -- Count current VIP tickets for that stage
        SELECT COUNT(*) INTO current_vip_tickets
        FROM Ticket t
        JOIN Event e ON t.event_id = e.event_id
        WHERE t.ticket_type_id = 1
        AND e.stage_id = stage_id;

        -- Calculate 10% VIP limit
        SET vip_limit = FLOOR(stage_capacity * 0.10);

        -- If current VIP tickets exceed or hit the limit, block insert
        IF current_vip_tickets >= vip_limit THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'VIP tickets for this stage have reached the 10% limitation';
        END IF;

    END IF;
END;

```

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.

```

CREATE TRIGGER chk_after_insert_to_performance_for_brake
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
    DECLARE last_end_time TIME;

    /*Find the last end time for event*/
    SELECT MAX(p.end_time) INTO last_end_time
    FROM Performance p
    WHERE event_id = NEW.event_id
    AND p.end_time <= NEW.start_time;

    IF last_end_time IS NOT NULL THEN
        IF TIMESTAMPDIFF(MINUTE, last_end_time, NEW.start_time) < 5
        OR TIMESTAMPDIFF(MINUTE, last_end_time, NEW.start_time) > 30 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Brake between two performances must be at least 5 minutes and maximum 30 minutes';
        END IF;
    END IF;
END;

```

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.

```

--
CREATE TRIGGER check_staff_availability_trigger
BEFORE INSERT ON Works_on
FOR EACH ROW
BEGIN
    DECLARE conflict_events INT;

    SELECT COUNT(*)
    INTO conflict_events
    FROM Works_on w
    JOIN Event e1 ON w.event_id = e1.event_id
    JOIN Event e2 ON e2.event_id = NEW.event_id
    WHERE w.staff_id = NEW.staff_id
    AND e1.event_date = e2.event_date
    AND w.stage_id != NEW.stage_id;

    IF conflict_events > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Staff member already assigned to a different stage on this date.';
    END IF;
END;

```

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.

```
-- Trigger to check if a rating occurs from a used ticket
CREATE TRIGGER check_if_tck_is_used
BEFORE INSERT ON Rating
FOR EACH ROW
BEGIN
    DECLARE ticket_used BOOLEAN;
    DECLARE ticket_id_str VARCHAR(20);

    SET ticket_id_str = CAST(NEW.ticket_id AS CHAR);

    SELECT used INTO ticket_used
    FROM Ticket
    WHERE ticket_id = NEW.ticket_id;

    IF ticket_used = FALSE THEN
        set @message_text = CONCAT_WS('Cannot insert rating: ticket with ID', ticket_id_str, ' was not used.');
```

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.

```

/*Trigger that blocks inserts to Resale queue if the event is not sold out */
CREATE TRIGGER trg_resale_queue_opens
BEFORE INSERT ON Resale_Queue
FOR EACH ROW
BEGIN
    DECLARE current_count INT;
    DECLARE max_capacity INT;
    DECLARE event_id_for_ticket INT;

    /*Get event id from ticket*/
    SELECT event_id INTO event_id_for_ticket
    FROM Ticket
    WHERE ticket_id = NEW.ticket_id;

    /*Count how many tickets have been sold for specific event*/
    SELECT COUNT(*) INTO current_count
    FROM Ticket
    WHERE event_id = event_id_for_ticket;

    /*Load Capacity of stage that is connected to the event*/
    SELECT s.max_capacity INTO max_capacity
    FROM Event e
    JOIN Stage s ON e.stage_id = s.stage_id
    WHERE e.event_id = event_id_for_ticket;

    /*Check if you can activate resale queue*/
    IF current_count < max_capacity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Resale queue is not available. Event not sold out yet.';
    END IF;

END;

```

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`.

```

CREATE TRIGGER trg_after_resale_insert
AFTER INSERT ON Resale_Queue
FOR EACH ROW
BEGIN
    DECLARE v_demand_id INT;
    DECLARE v_buyer_id INT;
    DECLARE ticket_count INT;
    DECLARE new_visitor_id INT;

    /* Βρες matching demand (με βάση event και ticket type)*/
    SELECT dq.demand_id, dq.buyer_id
    INTO v_demand_id, v_buyer_id
    FROM Demand_Queue dq
    JOIN Ticket t ON t.ticket_id = NEW.ticket_id
    WHERE dq.preferred_event_id = t.event_id
        AND dq.preferred_ticket_type = t.ticket_type_id
        AND dq.status = FALSE
    LIMIT 1;

    /*Αν υπάρχει matching demand*/
    IF v_demand_id IS NOT NULL THEN

        /*Count how many tickets the seller owns*/
        SELECT COUNT(*) INTO ticket_count
        FROM Ticket
        WHERE visitor_id = NEW.seller_id;

        IF ticket_count > 1 THEN
            /*Create new visitor from Resale Buyer info*/
            INSERT INTO Visitor (name, surname, age, email, phone_number, photo_url, photo_description)
            SELECT name, surname, age, email, phone_number, photo_url, photo_description
            FROM Resale_Buyer
            WHERE buyer_id = v_buyer_id;

            /*Get the new visitor id*/
            SET new_visitor_id = LAST_INSERT_ID();

            /*Reassign ticket to new visitor*/
            UPDATE Ticket
            SET visitor_id = new_visitor_id
            WHERE ticket_id = NEW.ticket_id;
        ELSE
            /* Κάνε update τα στοιχεία του επισκέπτη με του αγοραστή*/
            UPDATE Visitor
            JOIN Resale_Buyer rb ON rb.buyer_id = v_buyer_id
            SET
                Visitor.name = rb.name,
                Visitor.surname = rb.surname,
                Visitor.age = rb.age,
                Visitor.email = rb.email,
                Visitor.phone_number = rb.phone_number,
                Visitor.photo_url = rb.photo_url,
                Visitor.photo_description = rb.photo_description
            WHERE Visitor.visitor_id = NEW.seller_id;
        END IF;

        /*Καταγραφή στο log*/
        INSERT INTO Resale_Log (ticket_id, old_owner_id, new_owner_id, sale_price)
        VALUES (
            NEW.ticket_id,
            NEW.seller_id,
            IF(ticket_count > 1, new_visitor_id, v_buyer_id),
            NEW.price
        );

        /*Είληξε τις εγγραφές από queues*/
        /*ΔΕΝ ΜΠΟΡΕΙΣ ΝΑ ΚΑΝΕΙΣ UPDATE TO TABLE ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΕΙ ΤΟ TRIGGER ΟΤΑΝ ΚΑΛΕΙΤΑΙ*/
        /*UPDATE Resale_Queue SET status = TRUE WHERE resale_id = NEW.resale_id;*/
        UPDATE Demand_Queue SET status = TRUE WHERE demand_id = v_demand_id;
        DELETE FROM Resale_Buyer WHERE buyer_id = v_buyer_id;
    END IF;
END;

```

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`.

```

CREATE TRIGGER trg_after_demand_queue_insert
AFTER INSERT ON Demand_Queue
FOR EACH ROW
BEGIN
    DECLARE v_resale_id INT;
    DECLARE v_seller_id INT;
    DECLARE ticket_count INT;
    DECLARE new_visitor_id INT;
    DECLARE v_ticket_id INT;
    DECLARE v_event_id INT;
    DECLARE v_ticket_type INT;
    DECLARE resale_price FLOAT;

    /*Matching*/
    SELECT rq.resale_id, rq.seller_id, rq.ticket_id, rq.price
    INTO v_resale_id, v_seller_id, v_ticket_id, resale_price
    FROM Resale_Queue rq
    JOIN Ticket t ON t.ticket_id = rq.ticket_id
    WHERE t.event_id = NEW.preferred_event_id
    AND t.ticket_type_id = NEW.preferred_ticket_type
    AND rq.status = FALSE
    LIMIT 1;

    IF v_resale_id IS NOT NULL THEN

        /*Count how many tickets the seller owns*/
        SELECT COUNT(*) INTO ticket_count
        FROM Ticket
        WHERE visitor_id = v_seller_id;

        IF ticket_count > 1 THEN
            /*Create new visitor from Resale_Buyer info*/
            INSERT INTO Visitor (name, surname, age, email, phone number, photo url, photo description)
            SELECT name, surname, age, email, phone number, photo url, photo description
            FROM Resale_Buyer
            WHERE buyer_id = NEW.buyer_id;

            /*Get the latest visitor id to use it on the ticket*/
            SET new_visitor_id = LAST_INSERT_ID();

            UPDATE Ticket
            SET visitor_id = new_visitor_id
            WHERE ticket_id = v_ticket_id;

        ELSE /*Update infos about visitor*/
            UPDATE Visitor
            JOIN Resale_Buyer rb ON rb.buyer_id = NEW.buyer_id
            SET
                Visitor.name = rb.name,
                Visitor.surname = rb.surname,
                Visitor.age = rb.age,
                Visitor.email = rb.email,
                Visitor.phone number = rb.phone number,
                Visitor.photo url = rb.photo url,
                Visitor.photo description = rb.photo description
            WHERE Visitor.visitor_id = v_seller_id;
        END IF;

        INSERT INTO Resale_Log (ticket_id, old_owner_id, new_owner_id, sale_price)
        VALUES (
            v_ticket_id,
            v_seller_id,
            IF(ticket_count > 1, new_visitor_id, NEW.buyer_id),
            resale_price
        );

        /*Σβήσε τις εγγραφές από queues*/
        /*ΔΕΝ ΜΠΟΡΕΙΣ ΝΑ ΚΑΝΕΙΣ UPDATE TO TABLE ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΕΙ ΤΟ TRIGGER ΟΤΑΝ ΚΑΛΕΙΤΑΙ
        ΘΑ ΔΟΚΙΜΑΣΩ ΜΕ BEFORE ΚΑΙ NEW.status := TRUE
        */
        UPDATE Resale_Queue SET status = TRUE WHERE resale_id = v_resale_id;
        DELETE FROM Resale_Buyer WHERE buyer_id = NEW.buyer_id;

    END IF;

```

Το trigger `check_if_artist_is_solo_blns_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.

```

-- Trigger to check if the artist whos going to be inserted on a band is solo or not
CREATE TRIGGER check_if_artist_is_solo_blngs_to
BEFORE INSERT ON Belongs_to
FOR EACH ROW
BEGIN

    DECLARE v_is_solo BOOLEAN;

    SELECT is_solo INTO v_is_solo
    FROM Artist
    WHERE artist_id = NEW.artist_id;

    IF v_is_solo = TRUE THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist must not be solo to be on a band';
    END IF;

END;

```

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.

```

-- Trigger to prevent ticket insertion when max capacity has been reached for a stage
CREATE TRIGGER check_if_max_cap_is_reached_to_insert_ticket
BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN

    DECLARE number_of_tickets INT;
    DECLARE max_capacity_of_stage INT;

    /*COUNT of tickets for specific event*/
    SELECT COUNT(*) INTO number_of_tickets
    FROM Ticket
    WHERE event_id = NEW.event_id;

    /*Take max capacity of stage for this event*/
    SELECT s.max_capacity INTO max_capacity_of_stage
    FROM Event e
    JOIN Stage s ON s.stage_id = e.stage_id
    WHERE e.event_id = NEW.event_id;

    IF number_of_tickets >= max_capacity_of_stage THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Tickets are sold out for this event';
    END IF;

END;

```

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 παίρνει το `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.



```

-- Trigger to check if the artist or band is available for the performance
CREATE TRIGGER check_performer_availability_on_insert
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
    DECLARE v_artist_id INT;
    DECLARE v_band_id INT;
    DECLARE conflict_count INT;
    DECLARE current_year INT;
    DECLARE years_participated INT;

    /*Take the artist-band*/
    SELECT artist_id, band_id INTO v_artist_id, v_band_id
    FROM Performs
    WHERE performs_id = NEW.performs_id;

    /*Take the year of the festival*/
    SELECT f.year INTO current_year
    FROM Event e
    JOIN Festival f ON f.festival_id = e.festival_id
    WHERE e.event_id = NEW.event_id;

    IF v_artist_id IS NOT NULL THEN
        SELECT COUNT(*) INTO conflict_count
        FROM Performance p
        JOIN Performs pf ON pf.performs_id = p.performs_id
        JOIN Event e ON e.event_id = p.event_id
        JOIN Event new_e ON new_e.event_id = NEW.event_id
        WHERE pf.artist_id = v_artist_id
        AND e.event_id != NEW.event_id
        AND e.event_date = new_e.event_date
        AND (
            (NEW.start_time BETWEEN p.start_time AND p.end_time)
            OR (NEW.end_time BETWEEN p.start_time AND p.end_time)
            OR (p.start_time BETWEEN NEW.start_time AND NEW.end_time)
            OR (p.end_time BETWEEN NEW.start_time AND NEW.end_time)
        );

        IF conflict_count > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'This artist is already performing elsewhere at an overlapping time.';
        END IF;

        -- Έλεγχος για συμμετοχή σε 3 συνεχόμενα φεστιβάλ
        SELECT COUNT(DISTINCT f.year) INTO years_participated
        FROM Performance p
        JOIN Performs pf ON pf.performs_id = p.performs_id
        JOIN Event e ON e.event_id = p.event_id
        JOIN Festival f ON f.festival_id = e.festival_id
        WHERE pf.artist_id = v_artist_id
        AND f.year IN (current_year - 1, current_year - 2, current_year - 3);

        -- Αν βρεθούν και τα δύο προηγούμενα έτη, απορρίπτουμε την εισαγωγή
        IF years_participated = 3 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Artist cannot participate in 4 consecutive years.';
        END IF;

    ELSEIF v_band_id IS NOT NULL THEN
        SELECT COUNT(*) INTO conflict_count
        FROM Performance p
        JOIN Performs pf ON pf.performs_id = p.performs_id
        JOIN Event e ON e.event_id = p.event_id
        JOIN Event new_e ON new_e.event_id = NEW.event_id
        WHERE pf.band_id = v_band_id
        AND e.event_id != NEW.event_id
        AND e.event_date = new_e.event_date
        AND (
            (NEW.start_time BETWEEN p.start_time AND p.end_time)
            OR (NEW.end_time BETWEEN p.start_time AND p.end_time)
            OR (p.start_time BETWEEN NEW.start_time AND NEW.end_time)
            OR (p.end_time BETWEEN NEW.start_time AND NEW.end_time)
        );

        IF conflict_count > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'This band is already performing elsewhere at an overlapping time.';
        END IF;

        -- Έλεγχος για 3 συνεχόμενες χρονιές
        SELECT COUNT(DISTINCT f.year) INTO years_participated
        FROM Performance p
        JOIN Performs pf ON pf.performs_id = p.performs_id
        JOIN Event e ON e.event_id = p.event_id
        JOIN Festival f ON f.festival_id = e.festival_id
        WHERE pf.band_id = v_band_id
        AND f.year IN (current_year - 1, current_year - 2, current_year - 3);

        IF years_participated = 3 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Band cannot participate in 4 consecutive years.';
        END IF;
    END IF;
END;

```

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.

```
-- Trigger to check if the festival is in a different location each year
CREATE TRIGGER trg_check_different_location_per_year
BEFORE INSERT ON Festival
FOR EACH ROW
BEGIN
    DECLARE prev_location INT;

    -- find the location of the previous year's festival
    SELECT location_id INTO prev_location
    FROM Festival
    WHERE name = NEW.name
    AND year = NEW.year - 1
    LIMIT 1;

    -- Check if the previous location is the same as the new one
    IF prev_location IS NOT NULL AND prev_location = NEW.location_id THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Το φεστιβάλ δεν μπορεί να διεξαχθεί στην ίδια τοποθεσία δύο συνεχόμενα έτη.';
    END IF;
END;
```

**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`.

```

1  -- =====
2  -- Procedure: buy_specific_ticket
3  -- Description:
4  --     Handles the direct purchase of a specific resale ticket by a buyer.
5  --     The procedure performs the following steps:
6  --         1. Retrieves the ticket associated with the given resale ID.
7  --         2. Updates the current ticket owner (Visitor) with the buyer's information.
8  --         3. Records the purchase in the Buys_specific_ticket log table.
9  --         4. Deletes the resale entry from the Resale_Queue.
10 --     This procedure assumes that resale_id exists before calling and that
11 --     no foreign key is enforced on resale_id in the log table to allow deletion.
12 -- Parameters:
13 --     IN in_buyer_id    INT    → The buyer's ID (from Resale Buyer table).
14 --     IN in_resale_id  INT    → The resale entry ID (from Resale_Queue table).
15
16 --     CALL buy_specific_ticket(<buyer_id>, <resale_id>);
17 -- =====
18
19 DELIMITER $$
20
21 CREATE PROCEDURE buy_specific_ticket(
22     IN in_buyer_id INT,
23     IN in_resale_id INT
24 )
25 BEGIN
26
27     DECLARE ticket_id var INT;
28     DECLARE current_visitor_id INT;
29     DECLARE ticket_count INT;
30     DECLARE new_visitor_id INT;
31     /*Find ticket id from resale queue*/
32     SELECT ticket_id
33     INTO ticket_id_var
34     FROM Resale_Queue
35     WHERE resale_id = in_resale_id;
36
37     /*Find visitor from the ticket*/
38     SELECT visitor_id
39     INTO current_visitor_id
40     FROM Ticket
41     WHERE ticket_id = ticket_id_var;
42
43     /*Count how many tickets the seller owns*/
44     SELECT COUNT(*) INTO ticket_count
45     FROM Ticket
46     WHERE visitor_id = current_visitor_id;
47
48     IF ticket_count > 1 THEN
49         /*Create new visitor*/
50
51         INSERT INTO Visitor (name, surname, age, email, phone_number, photo_url, photo_description)
52         SELECT name, surname, age, email, phone_number, photo_url, photo_description
53         FROM Resale_Buyer
54         WHERE buyer_id = in_buyer_id;
55
56         SET new_visitor_id = LAST_INSERT_ID();
57
58         UPDATE Ticket
59         SET visitor_id = new_visitor_id
60         WHERE ticket_id = ticket_id_var;
61
62     ELSE
63         /*Set the buyer's values to visitor's values (change buyer with visitor)*/
64         UPDATE Visitor
65         SET

```

```

61
62 ELSE
63     /*Set the buyer's values to visitor's values (change buyer with visitor)*/
64     UPDATE Visitor
65     SET
66     name = (SELECT name FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
67     surname = (SELECT surname FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
68     age = (SELECT age FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
69     email = (SELECT email FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
70     phone_number = (SELECT phone_number FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
71     photo_url = (SELECT photo_url FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
72     photo_description = (SELECT photo_description FROM Resale_Buyer WHERE buyer_id = in_buyer_id)
73     WHERE visitor_id = current_visitor_id;
74 END IF;
75
76 /*Logging the transaction*/
77 INSERT INTO Buys_specific_ticket(buyer_id, resale_id, interest_date, status)
78 VALUES
79 (in_buyer_id, in_resale_id, NOW(), 'completed');
80
81 /*Delete the item from the resale queue*/
82 DELETE FROM Resale_Queue
83 WHERE resale_id = in_resale_id;
84 DELETE FROM Resale_Buyer
85 WHERE buyer_id = in_buyer_id;
86
87
88 END$$
89

```

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.

```

90 --=====
91 --Procedure to insert data into Works_On Table
92 --
93 --=====
94 DELIMITER $$
95
96 CREATE PROCEDURE Insert_Staff_Assignment (
97     IN in_staff_id INT,
98     IN in_stage_id INT,
99     IN in_event_id INT
100 )
101 BEGIN
102     -- Check if assignment already exists
103     IF EXISTS (
104         SELECT 1 FROM Works_on
105         WHERE staff_id = in_staff_id AND stage_id = in_stage_id AND event_id = in_event_id
106     ) THEN
107         SIGNAL SQLSTATE '45000'
108         SET MESSAGE_TEXT = 'Η ανάθεση υπάρχει ήδη.';
109     ELSE
110         INSERT INTO Works_on (staff_id, stage_id, event_id)
111         VALUES (in_staff_id, in_stage_id, in_event_id);
112     END IF;
113 END $$
114

```

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

```
117 -----
118 | -- Check staff coverage per stage and event based on role requirements
119 | -----
120 CREATE OR REPLACE PROCEDURE Check_Stage_Staff_Coverage()
121 BEGIN
122     SELECT
123         w.event_id,
124         w.stage_id,
125         stg.name AS stage_name,
126         stf.staff_role_id,
127         r.name AS role_name,
128         COUNT(*) AS assigned_staff,
129         stg.max_capacity,
130         CASE
131             WHEN stf.staff_role_id = 2 THEN ROUND(stg.max_capacity * 0.05)
132             WHEN stf.staff_role_id = 3 THEN ROUND(stg.max_capacity * 0.02)
133             ELSE 0
134         END AS required_staff,
135         CASE
136             WHEN stf.staff_role_id IN (2, 3) THEN
137                 CASE
138                     WHEN COUNT(*) >=
139                         CASE
140                             WHEN stf.staff_role_id = 2 THEN ROUND(stg.max_capacity * 0.05)
141                             WHEN stf.staff_role_id = 3 THEN ROUND(stg.max_capacity * 0.02)
142                         END
143                     THEN '✓ OK'
144                     ELSE '✗ Understaffed'
145                 END
146             ELSE 'N/A'
147         END AS status
148     FROM Works_on w
149     JOIN Stage stg ON w.stage_id = stg.stage_id
150     JOIN Staff stf ON w.staff_id = stf.staff_id
151     JOIN Staff_role r ON stf.staff_role_id = r.staff_role_id
152     WHERE stf.staff_role_id IN (1, 2, 3)
153     GROUP BY w.event_id, w.stage_id, stf.staff_role_id;
154 END $$
155
156 DELIMITER ;
```

### Query 4

Για το query 4 πραγματοποιήσαμε την παρακάτω υλοποίηση

```
EXPLAIN FORMAT = JSON
SELECT
  a.name AS Performer_Name,
  AVG(r.interpretation_rating) AS Avg_Interpretation_Rating,
  AVG(r.overall_impression) AS Avg_Overall_Impression
FROM Rating r
JOIN Performance p ON r.performance_id = p.performance_id
JOIN Performs pf ON p.performs_id = pf.performs_id
JOIN Artist a ON pf.artist_id = a.artist_id
WHERE a.artist_id = 2
GROUP BY a.name
```

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.

| Status                 | Duration |
|------------------------|----------|
| Starting               | 0.000002 |
| Query end              | 0.000000 |
| closing tables         | 0.000000 |
| Query end              | 0.000000 |
| Starting cleanup       | 0.000000 |
| Freeing items          | 0.000001 |
| Updating status        | 0.000003 |
| Reset for next command | 0.000001 |

Figure 1: Query 4 without force index traces



## EXPLAIN

```
{
  "query_block": {
    "select_id": 1,
    "cost": 0.01880477,
    "nested_loop": [
      {
        "table": {
          "table_name": "a",
          "access_type": "const",
          "possible_keys": ["PRIMARY"],
          "key": "PRIMARY",
          "key_length": "4",
          "used_key_parts": ["artist_id"],
          "ref": ["const"],
          "rows": 1,
          "filtered": 100
        }
      },
      {
        "table": {
          "table_name": "pf",
          "access_type": "ref",
          "possible_keys": ["PRIMARY", "idx_performs_artist_id"],
          "key": "idx_performs_artist_id",
          "key_length": "5",
          "used_key_parts": ["artist_id"],
          "ref": ["const"],
          "loops": 1,
          "rows": 5,
          "cost": 0.002379345,
          "filtered": 100,
          "using_index": true
        }
      },
      {
        "table": {
          "table_name": "p",
          "access_type": "ref",
          "possible_keys": ["PRIMARY", "performs_id", "idx_perf_event_start"],
          "key": "performs_id",
          "key_length": "4",
          "used_key_parts": ["performs_id"],
          "ref": ["festival_database.pf.performs_id"],
          "loops": 5,
          "rows": 1,
          "cost": 0.005686225,
          "filtered": 100,
          "using_index": true
        }
      },
      {
        "table": {
          "table_name": "r",
          "access_type": "ref",
          "possible_keys": ["idx_rating_performance_id"],
          "key": "idx_rating_performance_id",
          "key_length": "4",
          "used_key_parts": ["performance_id"],
          "ref": ["festival_database.p.performance_id"],
          "loops": 5,
          "rows": 1,
          "cost": 0.0107392,
          "filtered": 100
        }
      }
    ]
  }
}
```

Figure 2: Query 4 without force index EXPLAIN

```
CREATE INDEX idx_rating_performance_id ON Rating(performance_id);
EXPLAIN FORMAT = JSON
SELECT
  a.name AS Performer_Name,
  AVG(r.interpretation_rating) AS Avg_Interpretation_Rating,
  AVG(r.overall_impression) AS Avg_Overall_Impression
FROM Rating r FORCE INDEX(idx_rating_performance_id)
JOIN Performance p FORCE INDEX(PRIMARY) ON r.performance_id = p.performance_id
JOIN Performs pf FORCE INDEX(PRIMARY) ON p.performs_id = pf.performs_id
JOIN Artist a FORCE INDEX (PRIMARY) ON pf.artist_id = a.artist_id
WHERE a.artist_id = 2
GROUP BY a.name;
```

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 είχαν 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.

| Status                 | Duration |
|------------------------|----------|
| Starting               | 0.000003 |
| Query end              | 0.000000 |
| closing tables         | 0.000000 |
| Query end              | 0.000000 |
| Starting cleanup       | 0.000001 |
| Freeing items          | 0.000000 |
| Updating status        | 0.000003 |
| Reset for next command | 0.000000 |

Figure 4: Query 4 force index traces

## EXPLAIN

```
{
  "query_block": {
    "select_id": 1,
    "cost": 0.1588842,
    "nested_loop": [
      {
        "table": {
          "table_name": "a",
          "access_type": "const",
          "possible_keys": ["PRIMARY"],
          "key": "PRIMARY",
          "key_length": "4",
          "used_key_parts": ["artist_id"],
          "ref": ["const"],
          "rows": 1,
          "filtered": 100
        }
      },
      {
        "table": {
          "table_name": "r",
          "access_type": "ALL",
          "possible_keys": ["idx_rating_performance_id"],
          "loops": 1,
          "rows": 75,
          "cost": 0.0230798,
          "filtered": 100
        }
      },
      {
        "table": {
          "table_name": "p",
          "access_type": "eq_ref",
          "possible_keys": ["PRIMARY"],
          "key": "PRIMARY",
          "key_length": "4",
          "used_key_parts": ["performance_id"],
          "ref": ["festival_database.r.performance_id"],
          "loops": 75,
          "rows": 1,
          "cost": 0.0679022,
          "filtered": 100
        }
      },
      {
        "table": {
          "table_name": "pf",
          "access_type": "eq_ref",
          "possible_keys": ["PRIMARY"],
          "key": "PRIMARY",
          "key_length": "4",
          "used_key_parts": ["performs_id"],
          "ref": ["festival_database.p.performs_id"],
          "loops": 75,
          "rows": 1,
          "cost": 0.0679022,
          "filtered": 100,
          "attached_condition": "pf.artist_id = 2"
        }
      }
    ]
  }
}
```

Figure 5: Query 4 with force index EXPLAIN

## Query 6

Για το query 6 πραγματοποιήσαμε την εξής υλοποίηση

```
EXPLAIN FORMAT = JSON
SELECT
  v.name AS Visitor_Name,
  v.surname AS Visitor_Surname,
  e.event_name AS Event_Name,
  p.performance_id,
  p.type_of_performance,
  AVG((r.interpretation_rating + r.sound_and_lighting_rating + r.stage_presence_rating + r.organization_rating + r.overall_impression) / 5.0)
  AS Avg_Visitor_Total_Rating_Per_Performance
FROM Rating r
JOIN Ticket t ON r.ticket_id = t.ticket_id
JOIN Visitor v ON t.visitor_id = v.visitor_id
JOIN Performance p ON r.performance_id = p.performance_id
JOIN Event e ON p.event_id = e.event_id
/* Define visitor */
WHERE v.visitor_id = 1
AND t.used = TRUE
GROUP BY
  v.visitor_id, v.name, v.surname,
  e.event_id, e.event_name,
  p.performance_id, p.type_of_performance, p.start_time
ORDER BY e.event_name, p.start_time;
```

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.

| Status                 | Duration |
|------------------------|----------|
| Starting               | 0.000005 |
| Query end              | 0.000001 |
| closing tables         | 0.000000 |
| Query end              | 0.000001 |
| Starting cleanup       | 0.000003 |
| Freeing items          | 0.000001 |
| Updating status        | 0.000003 |
| Reset for next command | 0.000000 |

Figure 7: Query 6 without force index traces

## EXPLAIN

```
{
  "query_block": {
    "select_id": 1,
    "cost": 0.01994562,
    "filesort": {
      "sort_key": "e.event_name, p.start_time",
      "temporary_table": {
        "nested_loop": [
          {
            "table": {
              "table_name": "v",
              "access_type": "const",
              "possible_keys": ["PRIMARY"],
              "key": "PRIMARY",
              "key_length": "4",
              "used_key_parts": ["visitor_id"],
              "ref": ["const"],
              "rows": 1,
              "filtered": 100
            }
          },
          {
            "table": {
              "table_name": "t",
              "access_type": "ref",
              "possible_keys": ["PRIMARY", "idx_ticket_visitor_used"],
              "key": "idx_ticket_visitor_used",
              "key_length": "5",
              "used_key_parts": ["visitor_id", "used"],
              "ref": ["const", "const"],
              "loops": 1,
              "rows": 4,
              "cost": 0.00223266,
              "filtered": 100,
              "using_index": true
            }
          },
          {
            "table": {
              "table_name": "r",
              "access_type": "ref",
              "possible_keys": [
                "idx_rating_performance_id",
                "idx_rating_ticket_id"
              ],
              "key": "idx_rating_ticket_id",
              "key_length": "4",
              "used_key_parts": ["ticket_id"],
              "ref": ["festival_database.t.ticket_id"],
              "loops": 4,
              "rows": 1,
              "cost": 0.00891904,
              "filtered": 100
            }
          }
        ]
      }
    }
  },
  "table": {
    "table_name": "v",
    "access_type": "const",
    "possible_keys": ["PRIMARY"],
    "key": "PRIMARY",
    "key_length": "4",
    "used_key_parts": ["visitor_id"],
    "ref": ["const"],
    "rows": 1,
    "filtered": 100
  }
}
```



```

SELECT
  v.name AS Visitor_Name,
  v.surname AS Visitor_Surname,
  e.event_name AS Event_Name,
  p.performance_id,
  p.type_of_performance,
  AVG(
    (r.interpretation_rating + r.sound_and_lighting_rating + r.stage_presence_rating + r.organization_rating + r.overall_impression) / 5.0
  ) AS Avg_Visitor_Total_Rating_Per_Performance
FROM Rating r FORCE INDEX (idx_rating_ticket_id)
JOIN Ticket t FORCE INDEX (PRIMARY) ON r.ticket_id = t.ticket_id
JOIN Visitor v FORCE INDEX (PRIMARY) ON t.visitor_id = v.visitor_id
JOIN Performance p FORCE INDEX (idx_perf_event_start) ON r.performance_id = p.performance_id
JOIN Event e FORCE INDEX (PRIMARY) ON p.event_id = e.event_id
WHERE v.visitor_id = 1
AND t.used = TRUE
GROUP BY
  v.visitor_id, v.name, v.surname,
  e.event_id, e.event_name,
  p.performance_id, p.type_of_performance, p.start_time
ORDER BY e.event_name, p.start_time;

```

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 είχαν 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.

| Status                 | Duration |
|------------------------|----------|
| Starting               | 0.00000  |
| Query end              | 0.00000  |
| closing tables         | 0.00000  |
| Query end              | 0.00000  |
| Starting cleanup       | 0.00000  |
| Freeing items          | 0.00000  |
| Updating status        | 0.00000  |
| Reset for next command | 0.00000  |

Figure 10: Query 6 with force index traces

```

EXPLAIN
{
  "query_block": {
    "select_id": 1,
    "cost": 0.2970346,
    "filesort": {
      "sort_key": "e.event_name, p.start_time",
      "temporary_table": {
        "nested_loop": [
          {
            "table": {
              "table_name": "v",
              "access_type": "const",
              "possible_keys": ["PRIMARY"],
              "key": "PRIMARY",
              "key_length": "4",
              "used_key_parts": ["visitor_id"],
              "ref": ["const"],
              "rows": 1,
              "filtered": 100
            }
          },
          {
            "table": {
              "table_name": "r",
              "access_type": "ALL",
              "possible_keys": ["idx_rating_ticket_id"],
              "loops": 1,
              "rows": 75,
              "cost": 0.0230798,
              "filtered": 100
            }
          },
          {
            "table": {
              "table_name": "t",
              "access_type": "eq_ref",
              "possible_keys": ["PRIMARY"],
              "key": "PRIMARY",
              "key_length": "4",
              "used_key_parts": ["ticket_id"],
              "ref": ["festival_database.r.ticket_id"],
              "loops": 75,
              "rows": 1,
              "cost": 0.0679022,
              "filtered": 100,
              "attached_condition": "t.visitor_id = 1 and t.used = 1"
            }
          }
        ]
      }
    },
    {
      "table": {
        "table_name": "p",
        "access_type": "ref",
        "possible_keys": ["idx_perf_event_start"],
        "key": "idx_perf_event_start",
        "key_length": "4",
        "used_key_parts": ["performance_id"],
        "ref": ["festival_database.r.performance_id"],
        "loops": 75,
        "rows": 1,
        "cost": 0.1381504,
        "filtered": 100
      }
    },
    {
      "table": {
        "table_name": "e",
        "access_type": "eq_ref",
        "possible_keys": ["PRIMARY"],
        "key": "PRIMARY",
        "key_length": "4",
        "used_key_parts": ["event_id"],
        "ref": ["festival_database.p.event_id"],
        "loops": 75,
        "rows": 1,
        "cost": 0.0679022,
        "filtered": 100
      }
    }
  ]
}

```

Fig-

ure 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:

```
SET profiling = 1;

-- Hash table for artists
CREATE TEMPORARY TABLE TempArtist
SELECT artist_id, name
FROM Artist
WHERE artist_id = 2;

SELECT
  ta.name AS Performer_Name,
  AVG(r.interpretation_rating) AS Avg_Interpretation_Rating,
  AVG(r.overall_impression) AS Avg_Overall_Impression
FROM Rating r
JOIN Performance p ON r.performance_id = p.performance_id
JOIN Performs pf ON p.performs_id = pf.performs_id
JOIN TempArtist ta ON pf.artist_id = ta.artist_id
GROUP BY ta.name;

-- Starting query plan
SELECT
  a.name AS Performer_Name,
  AVG(r.interpretation_rating) AS Avg_Interpretation_Rating,
  AVG(r.overall_impression) AS Avg_Overall_Impression
FROM Rating r
JOIN Performance p ON r.performance_id = p.performance_id
JOIN Performs pf ON p.performs_id = pf.performs_id
JOIN Artist a ON pf.artist_id = a.artist_id
WHERE a.artist_id = 2
GROUP BY a.name;
SHOW PROFILES;
```

| Query_ID | Duration   | Query  |
|----------|------------|--|
| 1        | 0.00001242 | SHOW WARNINGS  |
| 2        | 0.00022757 | SHOW VARIABLES LIKE 'sql_mode'   |
| 3        | 0.00014365 | -- 1. Δημιουργείς μικρό πίνακα hash<br>CREATE TEMPORARY TABLE TempArtist<br>SELECT artist_id, name<br>FROM Artist<br>WHERE artist_id = 2   |
| 4        | 0.00000784 | SHOW WARNINGS  |
| 5        | 0.00012691 | -- 2. Κάνεις το join με αυτόν<br>SELECT<br>ta.name AS Performer_Name,<br>AVG(r.interpretation_rating) AS Avg_Interpretation_Rating,<br>AVG(r.overall_impression) AS Avg_Overall_Impression<br>FROM Rating r<br>JOIN Performance p ON r.performance_id = p.performance_id<br>JOIN Performs pf ON                |
| 6        | 0.00000724 | SHOW WARNINGS  |
| 7        | 0.00013341 | -- Starting query<br>SELECT<br>a.name AS Performer_Name,<br>AVG(r.interpretation_rating) AS Avg_Interpretation_Rating,<br>AVG(r.overall_impression) AS Avg_Overall_Impression<br>FROM Rating r<br>JOIN Performance p ON r.performance_id = p.performance_id<br>JOIN Performs pf ON p.performs_id = pf.performs |
| 8        | 0.00000909 | SHOW WARNINGS  |

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.

```
SET profiling = 1;

CREATE TEMPORARY TABLE TempVisitor
SELECT visitor_id, name, surname
FROM Visitor
WHERE visitor_id = 1;

SELECT
    tv.name AS Visitor_Name,
    tv.surname AS Visitor_Surname,
    e.event_name AS Event_Name,
    p.performance_id,
    p.type_of_performance,
    AVG((r.interpretation_rating + r.sound_and_lighting_rating + r.stage_presence_rating + r.organization_rating + r.overall_impression) / 5.0)
        AS Avg_Visitor_Total_Rating_Per_Performance
FROM Rating r
JOIN Ticket t ON r.ticket_id = t.ticket_id
JOIN TempVisitor tv ON t.visitor_id = tv.visitor_id
JOIN Performance p ON r.performance_id = p.performance_id
JOIN Event e ON p.event_id = e.event_id
WHERE t.used = TRUE
GROUP BY
    tv.visitor_id, tv.name, tv.surname,
    e.event_id, e.event_name,
    p.performance_id, p.type_of_performance, p.start_time
ORDER BY e.event_name, p.start_time;

-- Starting query plan
SELECT
    v.name AS Visitor_Name,
    v.surname AS Visitor_Surname,
    e.event_name AS Event_Name,
    p.performance_id,
    p.type_of_performance,
    AVG((r.interpretation_rating + r.sound_and_lighting_rating + r.stage_presence_rating + r.organization_rating + r.overall_impression) / 5.0)
        AS Avg_Visitor_Total_Rating_Per_Performance
FROM Rating r
JOIN Ticket t ON r.ticket_id = t.ticket_id
JOIN Visitor v ON t.visitor_id = v.visitor_id
JOIN Performance p ON r.performance_id = p.performance_id
JOIN Event e ON p.event_id = e.event_id
/* Define visitor */
WHERE v.visitor_id = 1
    AND t.used = TRUE
GROUP BY
    v.visitor_id, v.name, v.surname,
    e.event_id, e.event_name,
    p.performance_id, p.type_of_performance, p.start_time
ORDER BY e.event_name, p.start_time;

SHOW PROFILES;
```

Τα αποτελέσματα εμφανίζονται παρακάτω:

| Query_ID | Duration   | Query  |
|----------|------------|--|
| 1        | 0.00002172 | SHOW WARNINGS  |
| 2        | 0.00026984 | CREATE TEMPORARY TABLE TempVisitor<br>SELECT visitor_id, name, surname<br>FROM Visitor<br>WHERE visitor_id = 1   |
| 3        | 0.00001598 | SHOW WARNINGS  |
| 4        | 0.00029896 | SELECT<br>tv.name AS Visitor_Name,<br>tv.surname AS Visitor_Surname,<br>e.event_name AS Event_Name,<br>p.performance_id,<br>p.type_of_performance,<br>AVG((r.interpretation_rating + r.sound_and_lighting_rating + r.stage_presence_rating + r.organization_rating + r.overall_impression) / 5.0   |
| 5        | 0.00001851 | SHOW WARNINGS  |
| 6        | 0.00036834 | SHOW VARIABLES LIKE 'sql_mode'   |
| 7        | 0.00024680 | -- Starting query plan<br>SELECT<br>v.name AS Visitor_Name,<br>v.surname AS Visitor_Surname,<br>e.event_name AS Event_Name,<br>p.performance_id,<br>p.type_of_performance,<br>AVG((r.interpretation_rating + r.sound_and_lighting_rating + r.stage_presence_rating + r.organization_rating + r.ove |
| 8        | 0.00001670 | SHOW WARNINGS  |

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.

```
SELECT Count(*), e.event_id, e.event_name, g.max_capacity
FROM Ticket t
JOIN Event e ON e.event_id = t.event_id
JOIN Stage g ON g.stage_id = e.stage_id
WHERE t.event_id = 1
```

| Count(*) | event_id | event_name | max_capacity |
|----------|----------|------------|--------------|
| 11       | 1        | Rock Night | 50           |

1 row (0.000 s) [Edit](#), [Explain](#), [Export](#)

```
SELECT Count(*), e.event_id, e.event_name, g.max_capacity
FROM Ticket t
JOIN Event e ON e.event_id = t.event_id
JOIN Stage g ON g.stage_id = e.stage_id
WHERE t.event_id = 1
```

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

```
INSERT INTO Resale_Queue (resale_id, ticket_id, seller_id, listing_date, price, status)
VALUES
(1, 2, 1, CURDATE(), 130.00, FALSE)
```

Error in query (1644): Resale queue is not available. Event not sold out yet.

```
INSERT INTO Resale_Queue (resale_id, ticket_id, seller_id, listing_date, price, status)
VALUES
(1, 2, 1, CURDATE(), 130.00, FALSE);
```

Θα πάμε τώρα και θα φτιάξουμε ένα δοκιμαστικό σενάριο

Ξεκινάμε κάνοντας νέο δοκιμαστικό stage

```
INSERT INTO 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 INTO 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 INTO 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

```
INSERT INTO Resale_Queue (resale_id, ticket_id, seller_id, listing_date, price, status)  
VALUES  
(1, 221, 1, CURDATE(), 100.00, FALSE)
```

Query executed OK, 1 row affected. (0.004 s) [Edit](#)

```
INSERT INTO Resale_Queue (resale_id, ticket_id, seller_id, listing_date, price, status)  
VALUES  
(1, 221, 1, CURDATE(), 100.00, FALSE);
```

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 και τηλέφωνο +306925678902

Η demand queue έχει ήδη αυτές τις εισαγωγές

50

Select

SELECT \* FROM `Demand\_Queue` LIMIT 50 (0.000 s) Edit

|                                 |           |          |                       |                    |              |        |
|---------------------------------|-----------|----------|-----------------------|--------------------|--------------|--------|
| <input type="checkbox"/> Modify | demand_id | buyer_id | preferred_ticket_type | preferred_event_id | request_date | status |
| <input type="checkbox"/> edit   | 1         | 1        | 1                     | 1                  | 2025-04-01   | 0      |
| <input type="checkbox"/> edit   | 2         | 2        | 3                     | 1                  | 2025-04-02   | 0      |
| <input type="checkbox"/> edit   | 3         | 3        | 3                     | 1                  | 2025-04-03   | 0      |
| <input type="checkbox"/> edit   | 4         | 4        | 1                     | 2                  | 2025-04-04   | 0      |

What's next?   Modify   Select (0)   Export (0)

και έχουμε

```
INSERT INTO Demand_Queue (demand_id, buyer_id, preferred_ticket_type, preferred_event_id, request_date, status)
VALUES
(5, 16, 2, 41, CURDATE(), FALSE)
```

Query executed OK, 1 row affected. (0.006 s) Edit

```
INSERT INTO Demand_Queue (demand_id, buyer_id, preferred_ticket_type, preferred_event_id, request_date, status)
VALUES
(5, 16, 2, 41, CURDATE(), FALSE);
```

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.

Select data Show structure Alter table New item

Select Search Sort Limit 50 Action Select

SELECT \* FROM `Demand\_Queue` LIMIT 50 (0.000 s) Edit

| <input type="checkbox"/> Modify | demand_id | buyer_id | preferred_ticket_type | preferred_event_id | request_date | status |
|---------------------------------|-----------|----------|-----------------------|--------------------|--------------|--------|
| <input type="checkbox"/> edit   | 1         | 1        | 1                     | 1                  | 2025-04-01   | 0      |
| <input type="checkbox"/> edit   | 2         | 2        | 3                     | 1                  | 2025-04-02   | 0      |
| <input type="checkbox"/> edit   | 3         | 3        | 3                     | 1                  | 2025-04-03   | 0      |
| <input type="checkbox"/> edit   | 4         | 4        | 1                     | 2                  | 2025-04-04   | 0      |

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

Select data Show structure Alter table New item

Select Search Sort Limit 50 Action Select

SELECT \* FROM `Resale\_Queue` LIMIT 50 (0.000 s) Edit

| <input type="checkbox"/> Modify | resale_id | ticket_id | seller_id | listing_date | price  | status |
|---------------------------------|-----------|-----------|-----------|--------------|--------|--------|
| <input type="checkbox"/> edit   | 1         | 221       | 1         | 2025-05-07   | 100.00 | 1      |

Whole result 1 row Modify Save Selected (0) Edit Clone Delete Export (1)



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

|   |            |           |          |     |                               |                      |                  |
|---|------------|-----------|----------|-----|-------------------------------|----------------------|------------------|
| SELECT * FROM `Visitor` LIMIT 50 (0.000 s) Edit |            |           |          |     |                               |                      |                  |
| <input type="checkbox"/> Modify                 | visitor_id | name      | surname  | age | email                         | phone_number         |                  |
| <input type="checkbox"/> edit                   | 1          | Christos  | Iliak    | 18  | sad@gmail.com                 | 3245823598           | NULL             |
| <input type="checkbox"/> edit                   | 2          | Jessica   | Smith    | 78  | andreayang@lewis.biz          | 349-205-1094         | NULL             |
| <input type="checkbox"/> edit                   | 3          | Jennifer  | Wright   | 71  | fortega@alexander.com         | 525.695.7372         | NULL             |
| <input type="checkbox"/> edit                   | 4          | Michele   | George   | 63  | donna22@yahoo.com             | (642)984-3607x38539  | NULL             |
| <input type="checkbox"/> edit                   | 5          | Samantha  | Mahoney  | 48  | etyler@curry.biz              | (268)467-1120x433    | NULL             |
| <input type="checkbox"/> edit                   | 6          | Tyler     | Bowers   | 23  | amurray@hotmail.com           | +1-900-542-6104x199  | NULL             |
| <input type="checkbox"/> edit                   | 7          | Morgan    | Lopez    | 77  | jessicasandoval@yahoo.com     | +1-563-434-9639x3574 | NULL             |
| <input type="checkbox"/> edit                   | 8          | Kendra    | Malone   | 18  | megan74@hotmail.com           | 346.331.4088x954     | NULL             |
| <input type="checkbox"/> edit                   | 9          | Angela    | Garcia   | 32  | bboyle@porter-hall.com        | +1-238-243-7780x3096 | NULL             |
| <input type="checkbox"/> edit                   | 10         | Larry     | Cook     | 89  | marcus59@johnson.com          | 667-201-7080x1843    | NULL             |
| <input type="checkbox"/> edit                   | 11         | William   | Roberts  | 51  | poliver@valenzuela.com        | 118-331-5852x751     | NULL             |
| <input type="checkbox"/> edit                   | 12         | Joshua    | Perry    | 27  | trevor66@zamora-houston.com   | +1-631-816-2550x3973 | NULL             |
| <input type="checkbox"/> edit                   | 13         | Brandon   | Jones    | 33  | tolson@hotmail.com            | (344)697-7043        | NULL             |
| <input type="checkbox"/> edit                   | 14         | Ryan      | Richards | 25  | mary55@santos.com             | (623)196-1554        | NULL             |
| <input type="checkbox"/> edit                   | 15         | Joshua    | Pena     | 55  | ccabrera@hernandez.net        | +1-995-460-2271x8125 | NULL             |
| <input type="checkbox"/> edit                   | 16         | Shelby    | Brooks   | 14  | avelez@fuller.org             | +1-158-274-9086x7815 | NULL             |
| <input type="checkbox"/> edit                   | 17         | Jessica   | Williams | 90  | uhall@yahoo.com               | +1-620-830-0149x010  | NULL             |
| <input type="checkbox"/> edit                   | 18         | Ernest    | Gill     | 41  | xkeith@yahoo.com              | 001-345-578-5063x155 | NULL             |
| <input type="checkbox"/> edit                   | 19         | Jennifer  | Cohen    | 62  | denniswalker@yahoo.com        | 6757328427           | NULL             |
| <input type="checkbox"/> edit                   | 20         | Justin    | Singh    | 51  | shenderson@barr.org           | 001-576-383-0941x919 | NULL             |
| <input type="checkbox"/> edit                   | 21         | Larry     | Parker   | 33  | yescobar@hotmail.com          | 693-973-1093         | NULL             |
| <input type="checkbox"/> edit                   | 22         | Michael   | White    | 49  | nicole58@doyle.info           | (270)643-5844        | NULL             |
| <input type="checkbox"/> edit                   | 23         | Joseph    | O'Neill  | 15  | lauragreen@zimmerman.info     | 001-530-815-5325x493 | NULL             |
| <input type="checkbox"/> edit                   | 24         | Michael   | Payne    | 43  | hoganlauren@yahoo.com         | 719-616-7229x28726   | NULL             |
| <input type="checkbox"/> edit                   | 25         | Stacy     | Johnson  | 26  | barrerabryce@hotmail.com      | 001-235-724-7363x247 | NULL             |
| <input type="checkbox"/> edit                   | 26         | Kelly     | Foster   | 53  | hardingsydney@hotmail.com     | +1-305-989-2424x462  | NULL             |
| <input type="checkbox"/> edit                   | 27         | Tyler     | Rice     | 87  | emitchell@yahoo.com           | 001-731-726-1004x002 | NULL             |
| <input type="checkbox"/> edit                   | 28         | John      | Jennings | 76  | lauren78@oliver-parsons.com   | 758-874-6054x2311    | NULL             |
| <input type="checkbox"/> edit                   | 29         | Daniel    | Harris   | 35  | timothysmith@powers-kelly.com | +1-862-712-0393x0845 | NULL             |
| <input type="checkbox"/> edit                   | 30         | Lisa      | Mccarthy | 87  | nathan90@blackburn.info       | 001-171-669-3925x669 | NULL             |
| <input type="checkbox"/> edit                   | 31         | Apostolos | Venieris | 29  | apostolos.v@example.com       | +306925678902        | http://dummyimag |

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

Select: Resale\_Buyer

Select data

Show structure

Alter table

New item

Select

Search

Sort

Limit50

Text length100

ActionSelect

SELECT \* FROM `Resale\_Buyer` LIMIT 50 (0.000 s) Edit

| <div><div><div>Modify</div></div></div> | buyer_id | name      | surname         | age | email                        | phone_number  |                  |
|---|----------|-----------|-----------------|-----|------------------------------|---------------|------------------|
| <div><div><div>edit</div></div></div>   | 1        | Alexandra | Papadopoulou    | 28  | alexandra.p@example.com      | +306912345678 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 2        | George    | Andreadis       | 34  | george.andreadis@example.com | +306911234567 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 3        | Maria     | Ioannou         | 29  | maria.ioannou@example.com    | +306912345679 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 4        | Kostas    | Papanikolaou    | 45  | kostas.p@example.com         | +306913456780 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 5        | Eleni     | Karagianni      | 22  | eleni.k@example.com          | +306914567891 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 6        | Nikos     | Mitsotakis      | 38  | nikos.m@example.com          | +306915678902 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 7        | Dimitra   | Panagiotopoulou | 31  | dimitra.p@example.com        | +306916789013 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 8        | Michalis  | Alexiou         | 27  | michalis.a@example.com       | +306917890124 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 9        | Anna      | Georgiou        | 24  | anna.g@example.com           | +306918901235 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 10       | Spyros    | Koutras         | 40  | spyros.k@example.com         | +306919012346 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 11       | Vasiliki  | Douka           | 36  | vasiliki.d@example.com       | +306920123457 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 12       | Glannis   | Kouris          | 33  | glannis.k@example.com        | +306921234568 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 13       | Stella    | Makri           | 30  | stella.m@example.com         | +306922345679 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 14       | Christos  | Zervos          | 35  | christos.z@example.com       | +306923456780 | http://dummyimag |
| <div><div><div>edit</div></div></div>   | 15       | Georgia   | Lambrou         | 26  | georgia.l@example.com        | +306924567891 | http://dummyimag |

Whole result

15 rows

Modify

Save

Selected (0)

Edit

Clone

Delete

Export (15)

Import

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.

Select: Resale\_Log

Select data

Show structure

Alter table

New item

Select

Search

Sort

Limit

Action

50

Select

SELECT \* FROM `Resale\_Log` LIMIT 50 (0.000 s) Edit

|                                 |        |           |              |              |            |                     |
|---------------------------------|--------|-----------|--------------|--------------|------------|---------------------|
| <input type="checkbox"/> Modify | log_id | ticket_id | old_owner_id | new_owner_id | sale_price | sale_date           |
| <input type="checkbox"/> edit   | 1      | 221       | 1            | 31           | 100.00     | 2025-05-07 12:19:13 |

Whole result

Modify

Selected (0)

Export (1)

☐ 1 row

Save

Edit

Clone

Delete

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 στοιχείων του resale\_buyer.

## ΠΑΡΑΔΟΧΕΣ

---

- Στο πλαίσιο του σχεδιασμού του συστήματος, θεωρήθηκε ότι το Pulse University Festival μπορεί να διοργανώνει πολλαπλά 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
- 

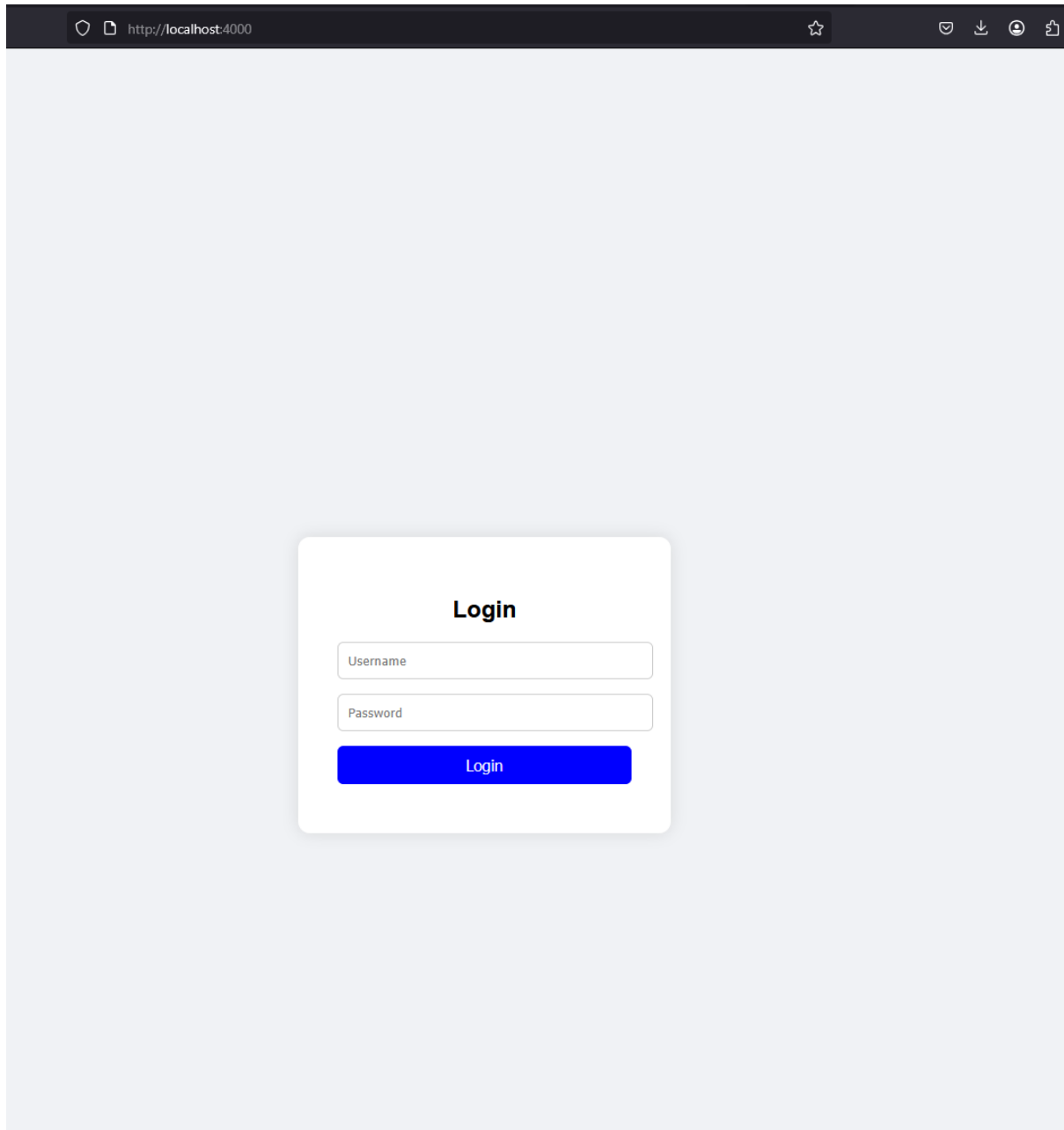
- Για το προσωπικό κρατήσαμε τους περιορισμούς που ζητούνται, ενώ για το τεχνικό προσωπικό δεν ορίσαμε κάποιο κατώτατο όριο. Για τον λόγο αυτό σε αντίστοιχο query ή κάποια μορφή παρουσίασης μπορεί να εμφανιστεί ότι μία σκηνή έχει 0 τεχνικό προσωπικό (δεν χρειάζεται)
- 

- Για οποιοδήποτε 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

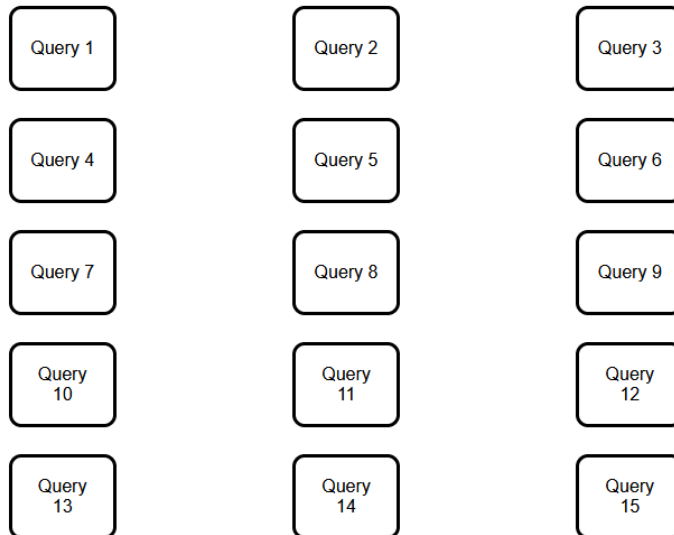
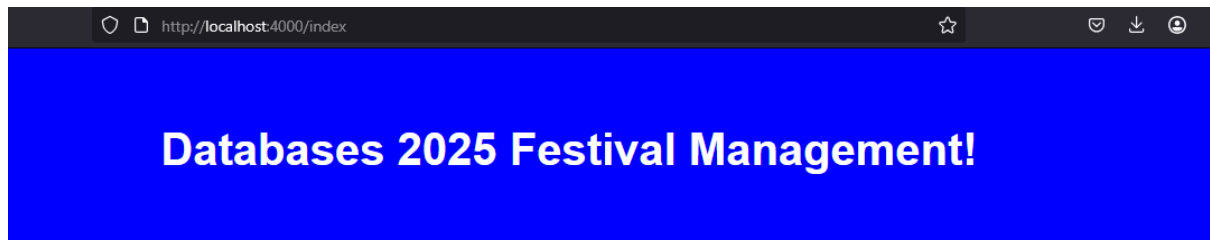
Το login page της σελίδας μας είναι το παρακάτω και μπορεί κάποιος να συνδεθεί με username: root και password: root



The screenshot shows a web browser window with the address bar displaying `http://localhost:4000`. The page content is a simple login form centered on a light blue background. The form is a white rounded rectangle containing the following elements:

- A title **Login** in bold black text.
- A text input field labeled "Username".
- A text input field labeled "Password".
- A blue button labeled "Login".

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



[Log Out](#)

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

←

→

↻

🔍 <http://localhost:4000/query-1?>

📄

☆

🔒

⬇

👤

📄

☰

Query 1 Results

SQL Query:

```
SELECT
  f.name AS festival_name,
  f.year AS festival_year,
  pm.name AS Payment_method,
  tt.name AS Ticket_type,
  SUM(t.price) AS Total_revenue
FROM Ticket t
JOIN Event e ON t.event_id = e.event_id
JOIN Festival f ON e.festival_id = f.festival_id
LEFT JOIN Payment_method pm ON pm.payment_method_id = t.payment_method_id
LEFT JOIN Ticket_type tt ON tt.ticket_type_id = t.ticket_type_id
GROUP BY f.festival_id, f.name, f.year, pm.name, tt.name
ORDER BY f.year, f.name, pm.name, tt.name;
```

| Festival      | Year | Payment Method | Ticket Type | Total Revenue (€) |
|---------------|------|----------------|-------------|-------------------|
| Electro World | 2025 | Bank Transfer  | Backstage   | 240.50            |
| Electro World | 2025 | Bank Transfer  | General     | 180.48            |
| Electro World | 2025 | Bank Transfer  | VIP         | 466.96            |
| Electro World | 2025 | Credit Card    | Backstage   | 411.44            |
| Electro World | 2025 | Credit Card    | General     | 443.25            |
| Electro World | 2025 | Credit Card    | VIP         | 243.59            |
| Electro World | 2025 | Debit Card     | Backstage   | 620.33            |
| Electro World | 2025 | Debit Card     | General     | 210.43            |
| Electro World | 2025 | Debit Card     | VIP         | 639.39            |
| Folk Routes   | 2025 | Bank Transfer  | Backstage   | 391.63            |
| Folk Routes   | 2025 | Bank Transfer  | General     | 571.02            |
| Folk Routes   | 2025 | Credit Card    | Backstage   | 287.63            |
| Folk Routes   | 2025 | Credit Card    | VIP         | 502.99            |
| Folk Routes   | 2025 | Debit Card     | Backstage   | 488.83            |