ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

Εξαμηνιαία Εργασία

Ακαδημαϊκό έτος 2024-2025

Ομαδα 123

Διονύσιος Εφραίμ Πλατανάς 03122279

Δημήτριος Βασιλαράς 03122184

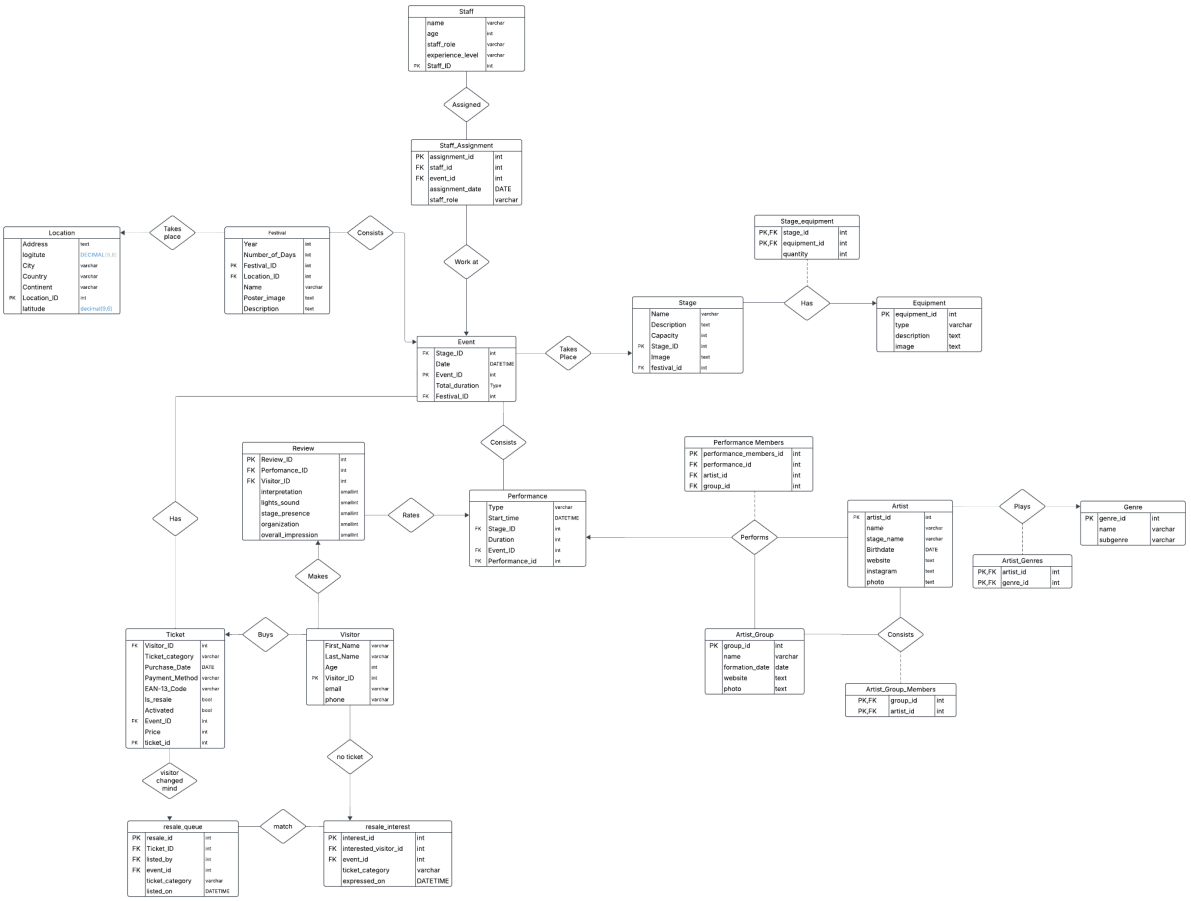
Βασίλειος Καλιάτσης 03122148

Περιεχόμενα:

Github repository: https://github.com/dplatanas/ECE_NTUA_DATA_BASES

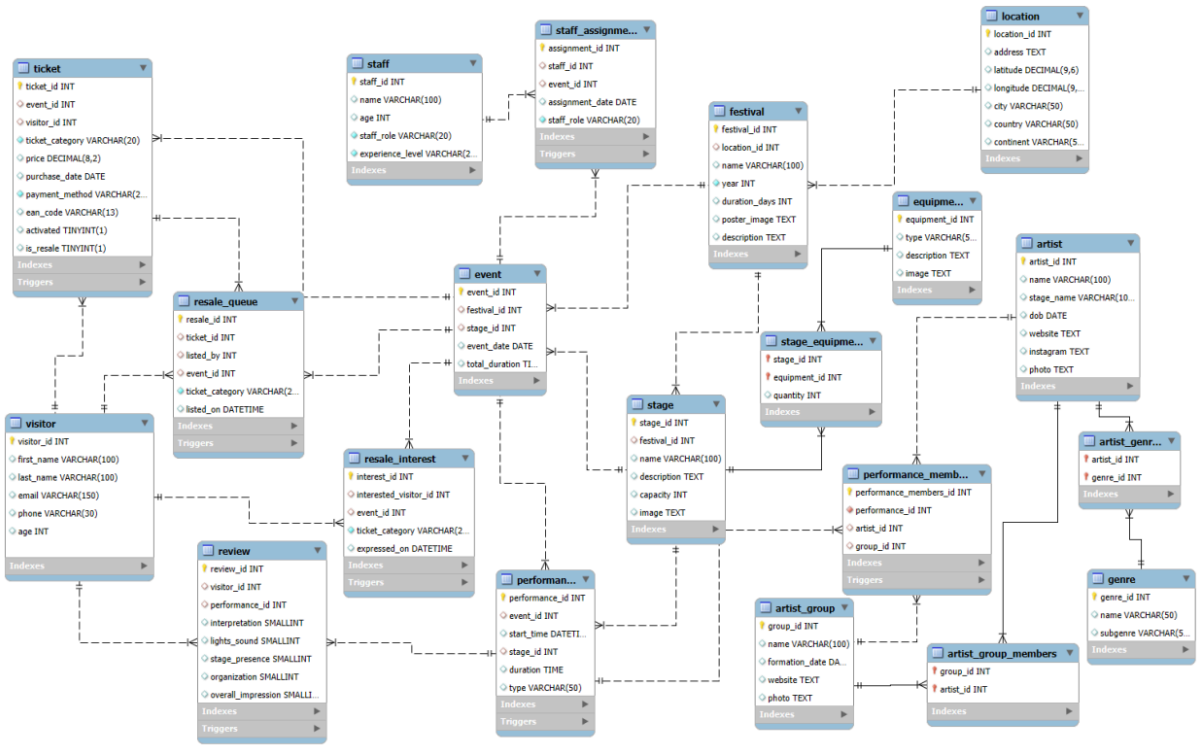
1. Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity - Relationship Diagram)

Σχεδιάσαμε το ERD, αξιοποιώντας την εφαρμογή Lucidchart, όπως φαίνεται παρακάτω:



2. Διάγραμμα Σχεσιακού Σχήματος (Relational Diagram)

Μέσω του προγράμματος MySQL Workbench, με χρήση reverse engineering, παράξαμε το Relational Diagram που ζητείται:



Το σχήμα μας οργανώνει όλα τα στοιχεία που χρειάζονται για τη λειτουργία ενός πολύπλευρου φεστιβάλ μουσικής. Κάθε φεστιβάλ (πίνακας festival) συνδέεται με μια τοποθεσία (location) και μπορεί να έχει πολλαπλές σκηνές (stage), στις οποίες ανατίθεται συγκεκριμένος εξοπλισμός (stage_equipment), ο οποίος φαίνεται στον πίνακα equipment.

Οι εμφανίσεις (performance) τοποθετούνται σε ημέρες και ώρες εντός ενός φεστιβάλ και συσχετίζονται με καλλιτέχνες ή γκρουπ μέσω του πίνακα performance_members. Οι καλλιτέχνες (artist) μπορούν να ανήκουν σε γκρουπ (artist_group) και να ασχολούνται με πολλά μουσικά είδη (artist_genre), τα οποία αποθηκεύονται στον πίνακα genre. Τα μέλη ενός γκρουπ αποθηκεύονται ξεχωριστά στον πίνακα artist_group_members.

Κάθε εμφάνιση μπορεί να βαθμολογηθεί από επισκέπτες (visitor) μέσω του πίνακα review, ενώ τα εισιτήρια (ticket) μπορούν να μεταπωληθούν (resale_queue) σε όσους βρίσκονται σε ουρά αναμονής (resale_interest), εάν κάποιος επισκέπτης αλλάξει γνώμη και δε θέλει να παρευρεθεί στην παράσταση (event).

Τέλος, το προσωπικό (staff) ανατίθεται (staff_assignment) σε κάθε event με συγκεκριμένο ρόλο (τεχνικός, ασφάλεια, υποστήριξη) και επίπεδο.

3. Σχεσιακο Σχημα (Relational Schema)

3.1 Αναλυση του relational schema

Δημιουργήσαμε το relational schema, το οποίο αρχικά κάνει drop οτιδήποτε θα παράξει μετά ως αρχικοποίηση (για να μην έχουμε error ότι ήδη υπάρχει το σχήμα ή τα tables) και ύστερα δημιουργεί τα ακόλουθα tables:

location:

Αποθηκεύουμε ό,τι χρειαζόμαστε για τη τοποθεσία του festival.
Το location_id είναι το πρωτεύον κλειδί αυτού του table, με τύπο:
INT AUTO_INCREMENT PRIMARY KEY.
Να σημειωθεί και ότι με τον τύπο DECIMAL(9,6) αποθηκεύουμε κατάλληλα τις γεωγραφικές συντεταγμένες που μας αφορούν, ως 9 δεκαδικά ψηφία, 6 από τα οποία βρίσκονται δεξιά της υποδιαστολής.

festival:
Αποθηκεύουμε όλα τα διαφορετικά festival, με πρωτεύον κλειδί το festival_id.
Με CHECK, ελέγχουμε ότι οι μέρες που διαρκεί κάθε festival είναι από μία έως επτά.
Συνδέουμε, επίσης, το κάθε festival με την τοποθεσία στην οποία διεξάγεται, μέσω του foreign key location_id.

stage:
Αποθηκεύουμε όλες τις διαφορετικές μουσικές σκηνές στις οποίες γίνονται οι παραστάσεις.
Έχουμε ως primary key το stage_id, ελέγχουμε ότι η μέγιστη χωρητικότητα κάθε σκηνής είναι θετικός αριθμός, και συνδέουμε κάθε σκηνή με το αντίστοιχο festival (μέσω του foreign key: festival_id).

equipment:
Εδώ αποθηκεύουμε όλον τον τεχνικό εξοπλισμό μας με αντίστοιχο description και φωτογραφία.

stage_equipment:
Με αυτό το table, βλέπουμε τι εξοπλισμό έχει η κάθε σκηνή. Συσχετίζουμε, άρα, τα tables equipment και stage (αξιοποιώντας τα equipment_id και stage_id ως foreign keys).
Λόγω της παραπάνω σχέσης, έχουμε ως πρωτεύον κλειδί αυτού του πίνακα το tuple (stage_id, equipment_id), ώστε να μην μπορεί ένα ηχείο, φως, μικρόφωνο κλπ να βρίσκεται σε πάνω από μία σκηνή ταυτόχρονα.
Ελέγχουμε με CHECK, ότι κάθε σκηνή έχει μη αρνητικό πλήθος τεχνικού εξοπλισμού.

event:

Αποθηκεύουμε τις διαφορετικές παραστάσεις μας, κάνοντας έναν τυπικό έλεγχο ότι δεν ξεπερνούν 12 ώρες διάρκεια. Με foreign keys τα festival_id, stage_id, βλέπουμε σε ποια σκηνή διεξάγεται η κάθε παράσταση και σε ποιο festival ανήκει.
Εδώ έχουμε πρωτεύον κλειδί το event_id.

artist:
Στο συγκεκριμένο πίνακα αποθηκεύονται όλοι οι καλλιτέχνες, μαζί με στοιχεία που ζητούνται από την εκφώνηση της εργασίας.
Έχουμε ως πρωτεύον κλειδί το artist_id.

artist_group:
Αποθηκεύουμε τα διαφορετικά συγκροτήματα που συμμετέχουν στο festival, μαζί με στοιχεία που ζητούνται από την εκφώνηση της εργασίας.
Έχουμε ως πρωτεύον κλειδί το group_id.

artist_group_members:
Εδώ φαίνονται τα μέλη (artists) του κάθε group. Ως primary key έχουμε το ζευγάρι (group_id, artist_id) καθώς ένας artist μπορεί να είναι σε παραπάνω από ένα groups και προφανώς ένα group έχει πάνω από έναν artist.

genre:
Αποθηκεύουμε το σύνολο των μουσικών ειδών. Ως primary key έχουμε το genre_id που διακρίνει το κάθε είδος ή υποείδος. Ακόμα κρατάμε το όνομα κάθε είδους και επιπλέον τα υποείδη που μπορεί να χαρακτηρίζουν κάποιο είδος.

artist_genres:
Εδώ φαίνεται το μουσικό είδος ή υποείδος κάθε καλλιτέχνη (σύμφωνα με το FK genre_id). Ως primary key κρατάμε το ζευγάρι (artist_id, genre_id) αφού κάθε artist μπορεί να έχει παραπάνω από ένα είδος ή υποείδος μουσικής και αυτό διακρίνεται μέσω του genre_id.

performance:
Αποθηκεύονται πληροφορίες για τις εμφανίσεις των καλλιτεχνών. Ως primary key έχουμε το performance_id που διακρίνει κάθε διαφορετική εμφάνιση ενός γκρουπ ή ενός καλλιτέχνη. Κρατάμε την παράσταση και τη σκηνή (event_id, stage_id), το πότε ξεκινάει και πόσο διαρκεί μια εμφάνιση (start_time, duration) και το είδος της (type). Εδώ βάζουμε και ένα CHECK για το duration καθώς μια εμφάνιση δεν μπορεί να διαρκεί πάνω από 3 ώρες.

performance_members:
Εδώ φαίνονται οι artists και τα groups που παίρνουν μέρος σε κάθε εμφάνιση. Ως primary key έχουμε το performance_members_id. Κρατάμε τα performance_id, artist_id και group_id και επίσης διευκρινίζεται ως UNIQUE KEY (performance_id, artist_id, group_id) για τη μοναδικότητα των (group_id, artist_id) σε κάθε εμφάνιση.

visitor:
Αποθηκεύονται οι επισκέπτες του φεστιβάλ. Ως primary key έχουμε το visitor_id που διακρίνει κάθε επισκέπτη. Ακόμα κρατάμε παραπάνω πληροφορίες από τον καθένα, δηλαδή το ονοματεπώνυμο του (first_name, last_name) καθώς και email, κινητό και ηλικία (email, phone, age). Κάθε email για κάθε visitor είναι μοναδικό, το οποίο διευκρινίζεται με UNIQUE και επίσης με CHECK ελέγχουμε πως η ηλικία σίγουρα δεν είναι αρνητικός αριθμός.

ticket:
Αποθηκεύονται τα εισιτήρια του φεστιβάλ για τις διάφορες παραστάσεις. Ως primary key έχουμε το ticket_id που διακρίνει κάθε εισιτήριο. Επίσης, σε αυτό το table αναγράφεται και ο κάτοχος του εισιτηρίου visitor_id και το event_id που δείχνει την παράσταση για την οποία προορίζεται το εισιτήριο. Ακόμα φαίνονται και παραπάνω πληροφορίες για τα εισιτήρια (ticket_category, price, purchase_date, payment_method, ean_code). Επίσης, χρησιμοποιείται μια boolean μεταβλητή activated που δείχνει αν ένα εισιτήριο είναι ενεργό ή όχι και μια boolean μεταβλητή is_resale που δείχνει αν ο κάτοχος ενός εισιτηρίου άλλαξε γνώμη και θέλει να θέσει το εισιτήριο του για μεταπώληση (TRUE όταν θέλει, FALSE όταν δεν έχει αλλάξει γνώμη).

resale_queue:
Εδώ αποθηκεύονται τα εισιτήρια που μπαίνουν στην ουρά μεταπώλησης. Η εισαγωγή εισιτηρίων εδώ γίνεται όταν κάποιο ticket_id (δηλαδή για κάποιο εισιτήριο) στο table ticket είναι μη ενεργοποιημένο ακόμα και το attribute is_resale έχει γίνει TRUE. Ως primary key έχουμε το resale_id και παράλληλα κρατάμε τα ticket_id, event_id, ticket_category ως πληροφορίες για το εισιτήριο, το listed_by για να φαίνεται ο μεταπωλητής και το listed_on για την ημερομηνία που δηλώθηκε κάποιο εισιτήριο για μεταπώληση ώστε αυτό να το αξιοποιήσουμε στη συνέχεια για τη λειτουργία με σειρά FIFO της ουράς. Οι παραπάνω λεπτομέρειες για το εισιτήριο δεν χρειάζονται καθώς θα εμφανίζονται ξανά αν το συγκεκριμένο εισιτήριο πουληθεί και μπει ξανά στο ticket table με το προηγούμενο αρχικό ticket_id. Εκεί θα φανεί και η τιμή του εισιτηρίου η οποία και στη μεταπώληση δεν θα αλλάζει. (Περισσότερες λεπτομέρειες στη περιγραφή λειτουργίας των διαδικασιών για την ουρά).

resale_interest:
Εδώ αποθηκεύονται οι ενδιαφερόμενοι για ένα εισιτήριο και είδος εισιτηρίου για μια παράσταση που έχει γίνει sold out, δηλαδή ο αριθμός των εισιτηρίων που έχουν πωληθεί έχει γίνει ίσος με τη χωρητικότητα της σκηνής που παίζεται η παράσταση. Ως primary key έχουμε το interest_id που συμβολίζει ένα διαφορετικό ενδιαφέρον για μια παράσταση (όχι απαραίτητα με διαφορετικούς ενδιαφερόμενους). Ακόμα, κρατάμε τον ενδιαφερόμενο interested_visitor_id, το event_id και ticket_category για τα οποία ενδιαφέρεται, και το expressed_on δηλαδή την ημερομηνία που εκδήλωσε το ενδιαφέρον για την παράσταση, κάτι το

οποίο χρησιμοποιείται στην προτεραιότητα (λειτουργία σε σειρά FIFO) των ενδιαφερομένων για τις παραστάσεις στο κομμάτι της αντιστοίχισης με εισιτήρια που είναι διαθέσιμα για επαναγορά στην ουρά μεταπώλησης.

review:
Με αυτόν τον πίνακα αποθηκεύουμε τις αξιολογήσεις που μπορεί να κάνει ένας επισκέπτης για οποιαδήποτε εμφάνιση παρακολουθήσει. Με χρήση CHECKS, επιβάλλουμε οι τιμές των αξιολογήσεων να ακολουθούν την κλίμακα Likert. Για να σιγουρευτούμε ότι κάθε επισκέπτης μπορεί να κάνει το πολύ μια αξιολόγηση για κάθε εμφάνιση, έχουμε: UNIQUE (visitor_id, performance_id), όπου τα visitor_id και performance_id είναι foreign keys που συνδέουν τους επισκέπτες με τις εμφανίσεις.

staff:
Εδώ αποθηκεύουμε όλο το προσωπικό που δουλεύει στο festival. Κάνουμε τους κατάλληλους ελέγχους, ώστε η ηλικία να είναι θετικός αριθμός, οι ρόλοι που μπορεί να πάρει το προσωπικό να είναι “support”, “security”, ή “technician”. Κατηγοριοποιούμε, επίσης, το προσωπικό με βάση την εμπειρία του, στις κατηγορίες: 'intern', 'junior', 'average', 'experienced', 'senior'.

staff_assignment:
Ο πίνακας αυτός αποθηκεύει τις διάφορες αναθέσεις που γίνονται στο προσωπικό, ελέγχοντας να τηρείται η κατηγοριοποίηση του σε: 'technician', 'security', 'support'.

3.2 TRIGGERS

Για τον έλεγχο των εισαγόμενων δεδομένων, πέρα από τα διάφορα CHECKS που φαίνονται στο 3.1, αξιοποιούμε και διάφορα triggers. Έτσι, μπορούμε να υλοποιήσουμε πιο περίπλοκους ελέγχους, για δεδομένα τα οποία δεν είναι στατικά μέσα στο σύστημά μας (π.χ. αριθμός εισιτηρίων για μια παράσταση). Πιο συγκεκριμένα, έχουμε 12 triggers:

—Trigger που ελέγχει αν υπάρχουν επικαλυπτόμενες εμφανίσεις:
(chk_performance_no_overlap_insert)

code:

```
-- Overlapping performance on the same stage and date
CREATE TRIGGER chk_performance_no_overlap_insert
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
  DECLARE cnt INT;
  DECLARE v_stage INT;
  DECLARE v_date DATE;
```

```
  -- Finding the stage and date of the new performance
  SELECT stage_id, event_date
    INTO v_stage, v_date
    FROM Event
   WHERE event_id = NEW.event_id;

  -- Find how many existing performances overlap
  SELECT COUNT(*) INTO cnt
    FROM performance p
    JOIN Event ev ON p.event_id = ev.event_id
   WHERE ev.stage_id    = v_stage
     AND ev.event_date  = v_date
     AND TIME_TO_SEC(NEW.start_time) < TIME_TO_SEC(p.start_time)  +
TIME_TO_SEC(p.duration)
     AND TIME_TO_SEC(p.start_time) < TIME_TO_SEC(NEW.start_time)  +
TIME_TO_SEC(NEW.duration);

  IF cnt > 0 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Overlapping performance on the same stage
and date';
  END IF;
END$$
```

—Triggers που απαγορεύει τη συμμετοχή του ίδιου καλλιτέχνη (συγκροτήματος) για περισσότερα από 3 συνεχόμενα έτη. Το chk_no_4yrs_in_row_insert κάνει τον έλεγχο κατά την είσοδο μιας εμφάνισης του καλλιτέχνη στο σύστημα, ενώ το chk_no_4yrs_in_row_update κατά την αλλαγή αυτής της εμφάνισης.

code:

```
-- No more than 3 years in a row
-- BEFORE INSERT trigger
CREATE TRIGGER chk_no_4yrs_in_row_insert
BEFORE INSERT ON performance_members
FOR EACH ROW
BEGIN
  DECLARE v_year INT;

  -- Find the event year
  SELECT YEAR(e.event_date)
    INTO v_year
    FROM performance p
    JOIN event e ON p.event_id = e.event_id
   WHERE p.performance_id = NEW.performance_id;

  -- For individual artist
  IF NEW.artist_id IS NOT NULL THEN
    IF (
      SELECT COUNT(DISTINCT YEAR(e2.event_date))
        FROM performance_members pm2
```

```
        JOIN performance p2 ON pm2.performance_id =
p2.performance_id
        JOIN event e2         ON p2.event_id          = e2.event_id
       WHERE pm2.artist_id = NEW.artist_id
         AND YEAR(e2.event_date) IN (v_year-1, v_year-2, v_year-3)
    ) = 3 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An artist has already participated for
3 years in a row';
    END IF;

  -- For group
  ELSEIF NEW.group_id IS NOT NULL THEN
    IF (
      SELECT COUNT(DISTINCT YEAR(e2.event_date))
        FROM performance_members pm2
        JOIN performance p2 ON pm2.performance_id =
p2.performance_id
        JOIN event e2         ON p2.event_id          = e2.event_id
       WHERE pm2.group_id = NEW.group_id
         AND YEAR(e2.event_date) IN (v_year-1, v_year-2, v_year-3)
    ) = 3 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A group has already participated for 3
years in a row';
    END IF;
  END IF;
END$$

-- BEFORE UPDATE trigger (in case someone reassigns an existing
row)
CREATE TRIGGER chk_no_4yrs_in_row_update
BEFORE UPDATE ON performance_members
FOR EACH ROW
BEGIN
  DECLARE v_year INT;

  SELECT YEAR(e.event_date)
    INTO v_year
    FROM performance p
    JOIN event e ON p.event_id = e.event_id
   WHERE p.performance_id = NEW.performance_id;

-- For individual artist
  IF NEW.artist_id IS NOT NULL THEN
    IF (
      SELECT COUNT(DISTINCT YEAR(e2.event_date))
        FROM performance_members pm2
        JOIN performance p2 ON pm2.performance_id =
p2.performance_id
```

```
        JOIN event e2         ON p2.event_id          = e2.event_id
       WHERE pm2.artist_id = NEW.artist_id
         AND (pm2.performance_members_id <>
OLD.performance_members_id)
         AND YEAR(e2.event_date) IN (v_year-1, v_year-2, v_year-3)
    ) = 3 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An artist has already participated for
3 years in a row';
    END IF;

-- For group
  ELSEIF NEW.group_id IS NOT NULL THEN
    IF (
      SELECT COUNT(DISTINCT YEAR(e2.event_date))
        FROM performance_members pm2
        JOIN performance p2 ON pm2.performance_id =
p2.performance_id
        JOIN event e2         ON p2.event_id          = e2.event_id
       WHERE pm2.group_id = NEW.group_id
         AND (pm2.performance_members_id <>
OLD.performance_members_id)
         AND YEAR(e2.event_date) IN (v_year-1, v_year-2, v_year-3)
    ) = 3 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A group has already participated for 3
years in a row';
    END IF;
  END IF;
END$$
```

—Triggers που διασφαλίζει ότι ένας καλλιτέχνης (συγκρότημα) δεν μπορεί να εμφανίζεται σε δύο σκηνές ταυτόχρονα. Το chk_no_artist_overlap_insert ελέγχει κατά την είσοδο των δεδομένων, ενώ το chk_no_artist_overlap_update εάν μεταβληθεί το ωράριο κάποιας εμφάνισης.

code:

```
-- Artist in only one performance at a time
-- BEFORE INSERT
CREATE TRIGGER chk_no_artist_overlap_insert
BEFORE INSERT ON performance_members
FOR EACH ROW
BEGIN
  DECLARE v_start DATETIME;
  DECLARE v_duration TIME;
  DECLARE v_end DATETIME;
  DECLARE v_conflicts INT;
```

```
    -- get the start and duration of the performance they're being
assigned to
     SELECT p.start_time, p.duration
       INTO v_start, v_duration
       FROM performance p
      WHERE p.performance_id = NEW.performance_id;

    SET v_end = ADDTIME(v_start, v_duration);

    -- count any other performance for the same artist that
overlaps in time
    SELECT COUNT(*)
      INTO v_conflicts
      FROM performance_members pm2
      JOIN performance p2
        ON pm2.performance_id = p2.performance_id
     WHERE pm2.artist_id = NEW.artist_id
       AND pm2.performance_id <> NEW.performance_id
       AND p2.start_time     < v_end
       AND ADDTIME(p2.start_time, p2.duration) > v_start;

    IF v_conflicts > 0 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An artist has two overlapping
performances';
    END IF;

END$$


-- BEFORE UPDATE
CREATE TRIGGER chk_no_artist_overlap_update
BEFORE UPDATE ON performance_members
FOR EACH ROW
BEGIN
  DECLARE v_start DATETIME;
  DECLARE v_duration TIME;
  DECLARE v_end DATETIME;
  DECLARE v_conflicts INT;


    -- get the start and duration of the (possibly new)
performance
    SELECT p.start_time, p.duration
      INTO v_start, v_duration
      FROM performance p
     WHERE p.performance_id = NEW.performance_id;
```

```
    SET v_end = ADDTIME(v_start, v_duration);

    -- count any other performance for this artist that overlaps,
    -- excluding the very row being updated
    SELECT COUNT(*)
      INTO v_conflicts
      FROM performance_members pm2
      JOIN performance p2
        ON pm2.performance_id = p2.performance_id
     WHERE pm2.artist_id = NEW.artist_id
       AND pm2.performance_members_id <>
OLD.performance_members_id
       AND p2.start_time      < v_end
       AND ADDTIME(p2.start_time, p2.duration) > v_start;

    IF v_conflicts > 0 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An artist has two overlapping
performances';
    END IF;

END$$
```

—Trigger που περιορίζει τα VIP tickets στο 10% της χωρητικότητας κάθε σκηνής. (trg_ticket_vip_insert)

code:

```
-- VIP tickets need to be <= 10% of total capacity for an event
CREATE
TRIGGER trg_ticket_vip_insert
BEFORE INSERT ON ticket
FOR EACH ROW
BEGIN
  -- 1) Declare variables
  DECLARE cap         INT DEFAULT 0;
  DECLARE vip_count   INT DEFAULT 0;
  DECLARE max_vip     INT DEFAULT 0;

  -- 2) Only enforce on VIP tickets
  IF NEW.ticket_category = 'VIP' THEN

    -- 2a) Find the stage capacity for this event
    SELECT s.capacity
      INTO cap
    FROM event ev
    JOIN stage s ON ev.stage_id = s.stage_id
    WHERE ev.event_id = NEW.event_id;

    -- 2b) Compute the 10% VIP limit
```

```
    SET max_vip = FLOOR(cap * 0.10);

    -- 2c) Count existing VIP tickets
    SELECT COUNT(*)
      INTO vip_count
    FROM ticket t
    WHERE t.event_id        = NEW.event_id
      AND t.ticket_category = 'VIP';

    -- 2d) If issuing this one would exceed the cap, abort
    IF vip_count + 1 > max_vip THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT =
          'Cannot issue VIP ticket: would exceed 10% of stage
capacity.';
    END IF;

  END IF;
END $$
```

—Trigger για την απαγόρευση ενός επισκέπτη να αξιολογήσει μια παράσταση, εαν δεν την έχει παρακολουθήσει. (check_ticket_review)

code:

```
-- Checks to make sure that only visitors with activated tickets
-- can review the corresponding perforomance
CREATE TRIGGER check_ticket_review
BEFORE INSERT ON review
FOR EACH ROW
BEGIN
     DECLARE activated_ BOOLEAN;
     SELECT t.activated INTO activated_
     FROM ticket t
     JOIN performance p ON p.event_id = t.event_id
     WHERE p.performance_id = NEW.performance_id
     AND t.visitor_id = NEW.visitor_id
     LIMIT 1;

     if activated_ = FALSE THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Ticket not activated, visitor cannot
review';
     END IF;
END$$
```

—Triggers για τον περιορισμό “ Το προσωπικό ασφαλείας πρέπει να καλύπτει τουλάχιστον το 5% του συνολικού αριθμού θεατών σε κάθε σκηνή και το βοηθητικό προσωπικό το 2%. ” Το trg_staff_assignment_before_insert κάνει τον έλεγχο κατά την είσοδο των δεδομένων, το trg_staff_assignment_before_update κατά την αλλαγή πόστου απο τα ενδιαφερόμενα σε κάποιο άλλο, και το

trg_staff_assignment_before_delete κατα την αφαίρεση ενος υπαλλήλου απο τις θέσεις ασφαλειας ή βοηθητικου προσωπικου.

code:

```
-- Enforce both support (2%) and security (5%)
-- BEFORE INSERT
CREATE TRIGGER trg_staff_assignment_before_insert
BEFORE INSERT ON staff_assignment
FOR EACH ROW
BEGIN
  DECLARE total_visitors    INT;
  DECLARE needed            INT;
  DECLARE current_count     INT;

  -- count how many tickets already sold
  SELECT COUNT(*) INTO total_visitors
    FROM ticket
   WHERE event_id = NEW.event_id;

  -- if this is a support hire, check 2%
  IF NEW.staff_role = 'support' THEN
    SET needed = CEIL((total_visitors) * 0.02);

    SELECT COUNT(*) INTO current_count
      FROM staff_assignment
     WHERE event_id   = NEW.event_id
       AND staff_role = 'support';

    -- include this new hire
    SET current_count = current_count + 1;

    IF current_count < needed THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Must have ≥2% support staff';
    END IF;
  END IF;

  -- if this is a security hire, check 5%
  IF NEW.staff_role = 'security' THEN
    SET needed = CEIL((total_visitors) * 0.05);

    SELECT COUNT(*) INTO current_count
      FROM staff_assignment
     WHERE event_id   = NEW.event_id
       AND staff_role = 'security';

    SET current_count = current_count + 1;

    IF current_count < needed THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Must have ≥5% security staff';
```

```
    END IF;
  END IF;
END$$

-- BEFORE DELETE
-- Prevent removing support/security if it would drop you below
threshold
CREATE TRIGGER trg_staff_assignment_before_delete
BEFORE DELETE ON staff_assignment
FOR EACH ROW
BEGIN
  DECLARE total_visitors    INT;
  DECLARE needed            INT;
  DECLARE remaining_count   INT;

  -- count current tickets
  SELECT COUNT(*) INTO total_visitors
    FROM ticket
   WHERE event_id = OLD.event_id;

  -- support removal
  IF OLD.staff_role = 'support' THEN
    SET needed = CEIL((total_visitors) * 0.02);

    SELECT COUNT(*) INTO remaining_count
      FROM staff_assignment
     WHERE event_id   = OLD.event_id
       AND staff_role = 'support';

    -- subtract the one being removed
    SET remaining_count = remaining_count - 1;

    IF remaining_count < needed THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot unassign support: would drop
below 2%';
    END IF;
  END IF;

  -- security removal
  IF OLD.staff_role = 'security' THEN
    SET needed = CEIL((total_visitors) * 0.05);

    SELECT COUNT(*) INTO remaining_count
      FROM staff_assignment
     WHERE event_id   = OLD.event_id
       AND staff_role = 'security';

    SET remaining_count = remaining_count - 1;
```

```
    IF remaining_count < needed THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot unassign security: would drop
below 5%';
    END IF;
  END IF;
END$$

-- BEFORE UPDATE
-- Prevent demoting support/security if it would violate
thresholds
CREATE TRIGGER trg_staff_assignment_before_update
BEFORE UPDATE ON staff_assignment
FOR EACH ROW
BEGIN
  DECLARE total_visitors    INT;
  DECLARE needed            INT;
  DECLARE remaining_count   INT;

  -- only care if you're changing someone *from* support or
security
  IF (OLD.staff_role = 'support' AND NEW.staff_role <> 'support')
   OR (OLD.staff_role = 'security' AND NEW.staff_role <>
'security') THEN

    SELECT COUNT(*) INTO total_visitors
      FROM ticket
     WHERE event_id = OLD.event_id;

    IF OLD.staff_role = 'support' THEN
      SET needed = CEIL((total_visitors) * 0.02);
      SELECT COUNT(*) INTO remaining_count
        FROM staff_assignment
       WHERE event_id   = OLD.event_id
         AND staff_role = 'support';
      SET remaining_count = remaining_count - 1;
      IF remaining_count < needed THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Cannot demote support: would drop
below 2%';
      END IF;
    END IF;

    IF OLD.staff_role = 'security' THEN
      SET needed = CEIL((total_visitors) * 0.05);
      SELECT COUNT(*) INTO remaining_count
        FROM staff_assignment
       WHERE event_id   = OLD.event_id
         AND staff_role = 'security';
      SET remaining_count = remaining_count - 1;
```

```
      IF remaining_count < needed THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Cannot demote security: would drop
below 5%';
      END IF;
    END IF;

  END IF;
END$$
```

— Triggers για εξασφάλιση τουλάχιστον 5, και το πολύ 30 λεπτά διάλειμμα μεταξύ διαδοχικών εμφανίσεων. Το trg_performance_break_insert ελέγχει τον προαναφερόμενο περιορισμό κατά τη φόρτωση των δεδομένων στη βάση μας (INSERT), ενώ το trg_performance_break_update κάνει τον έλεγχο για τυχόν μεταβολές στα ωράρια των εμφανίσεων.

code:

```
-- Checking the breaks for continues performances
-- BEFORE INSERT
CREATE TRIGGER trg_performance_break_insert
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
    DECLARE prev_end DATETIME;
    DECLARE next_start DATETIME;
    DECLARE new_end DATETIME;

    -- Compute new performance end time
    SET new_end = ADDTIME(NEW.start_time, NEW.duration);

    -- Find the immediately preceding performance
    SELECT ADDTIME(start_time, duration)
      INTO prev_end
    FROM performance
    WHERE event_id = NEW.event_id
      AND stage_id = NEW.stage_id
      AND start_time < NEW.start_time
    ORDER BY start_time DESC
    LIMIT 1;

    -- If a preceding performance exists, check the break
    IF prev_end IS NOT NULL THEN
      IF TIMESTAMPDIFF(MINUTE, prev_end, NEW.start_time) < 5
         OR TIMESTAMPDIFF(MINUTE, prev_end, NEW.start_time) > 30
THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Break between performances must be
between 5 and 30 minutes';
      END IF;
```

```
    END IF;

    -- Find the immediately following performance
    SELECT start_time
      INTO next_start
    FROM performance
    WHERE event_id = NEW.event_id
      AND stage_id = NEW.stage_id
      AND start_time > NEW.start_time
    ORDER BY start_time ASC
    LIMIT 1;

    -- If a following performance exists, check the break
    IF next_start IS NOT NULL THEN
      IF TIMESTAMPDIFF(MINUTE, new_end, next_start) < 5
         OR TIMESTAMPDIFF(MINUTE, new_end, next_start) > 30 THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Break between performances must be
between 5 and 30 minutes';
      END IF;
    END IF;
END;
$$

-- BEFORE UPDATE
CREATE TRIGGER trg_performance_break_update
BEFORE UPDATE ON performance
FOR EACH ROW
BEGIN
    DECLARE prev_end DATETIME;
    DECLARE next_start DATETIME;
    DECLARE new_end DATETIME;

    -- Compute updated performance end time
    SET new_end = ADDTIME(NEW.start_time, NEW.duration);

    -- Same logic for preceding row, but exclude the row being
updated
    SELECT ADDTIME(start_time, duration)
      INTO prev_end
    FROM performance
    WHERE event_id = NEW.event_id
      AND stage_id = NEW.stage_id
      AND start_time < NEW.start_time
      AND performance_id <> NEW.performance_id
    ORDER BY start_time DESC
    LIMIT 1;
```

```
    IF prev_end IS NOT NULL THEN
      IF TIMESTAMPDIFF(MINUTE, prev_end, NEW.start_time) < 5
         OR TIMESTAMPDIFF(MINUTE, prev_end, NEW.start_time) > 30
THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Break between performances must be
between 5 and 30 minutes';
      END IF;
    END IF;

    -- Same logic for following row
    SELECT start_time
      INTO next_start
    FROM performance
    WHERE event_id = NEW.event_id
      AND stage_id = NEW.stage_id
      AND start_time > NEW.start_time
      AND performance_id <> NEW.performance_id
    ORDER BY start_time ASC
    LIMIT 1;

    IF next_start IS NOT NULL THEN
      IF TIMESTAMPDIFF(MINUTE, new_end, next_start) < 5
         OR TIMESTAMPDIFF(MINUTE, new_end, next_start) > 30 THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Break between performances must be
between 5 and 30 minutes';
      END IF;
    END IF;
END;
$$
```

3.3 Υλοποίηση της ουράς μεταπώλησης

```
DELIMITER $$
CREATE PROCEDURE check_resale_ticket()
BEGIN
     INSERT INTO resale_queue(ticket_id, listed_by, event_id,
ticket_category, listed_ON)
     SELECT t.ticket_id, t.visitor_id, t.event_id,
t.ticket_category, NOW()
     FROM ticket t
     WHERE t.activated = FALSE AND t.is_resale = TRUE;

     UPDATE ticket
```

```
      SET visitor_id = NULL, purchase_date = NULL, is_resale =
FALSE
      WHERE activated = FALSE AND is_resale = TRUE;
END $$

CREATE PROCEDURE match_all_interested_to_resale_tickets()
BEGIN

    DROP TEMPORARY TABLE IF EXISTS temp_all_matches;
    CREATE TEMPORARY TABLE temp_all_matches (
        ticket_id INT,
        interest_id INT,
        event_id INT,
        PRIMARY KEY (ticket_id, interest_id)
    );

    INSERT INTO temp_all_matches (ticket_id, interest_id,
event_id)
    SELECT
        rq.ticket_id,
        ri.interest_id,
        rq.event_id
    FROM (
        SELECT
            rq.ticket_id,
            rq.event_id,
            rq.listed_on,
            t.ticket_category,
            ROW_NUMBER() OVER (
                PARTITION BY rq.event_id, t.ticket_category
                ORDER BY rq.listed_on DESC
            ) AS ticket_rank
        FROM resale_queue rq
        JOIN ticket t ON rq.ticket_id = t.ticket_id
        WHERE t.activated = FALSE
    ) rq
    JOIN (
        SELECT
            ri.interest_id,
            ri.event_id,
            ri.expressed_on,
            ri.ticket_category,
            ROW_NUMBER() OVER (
                PARTITION BY ri.event_id, ri.ticket_category
                ORDER BY ri.expressed_on ASC
            ) AS interest_rank
        FROM resale_interest ri
    ) ri ON rq.event_id = ri.event_id
```

```
        AND rq.ticket_category = ri.ticket_category
    WHERE rq.ticket_rank = ri.interest_rank;

    UPDATE ticket t
    JOIN temp_all_matches tm ON t.ticket_id = tm.ticket_id
    JOIN resale_interest ri ON tm.interest_id =
ri.interest_id
    SET t.visitor_id = ri.interested_visitor_id,
        t.purchase_date = NOW(),
        t.is_resale = FALSE;

    DELETE FROM resale_queue WHERE ticket_id IN (SELECT
ticket_id FROM temp_all_matches);
    DELETE FROM resale_interest WHERE interest_id IN (SELECT
interest_id FROM temp_all_matches);

    DROP TEMPORARY TABLE IF EXISTS temp_all_matches;
END $$

DELIMITER ;
SET GLOBAL event_scheduler = ON;
CREATE EVENT check_resale_ticket_event
ON SCHEDULE EVERY 5 SECOND
DO
  CALL check_resale_ticket();

CREATE EVENT match_resale_event
ON SCHEDULE EVERY 15 SECOND
DO
  CALL match_all_interested_to_resale_tickets();
```

Οι δύο παραπάνω διαδικασίες χρησιμοποιούνται στο κομμάτι και στη λειτουργία της ουράς μεταπώλησης και ουράς αναμονής (ο λόγος που χρησιμοποιήθηκαν διαδικασίες και όχι triggers είναι επειδή στη MySQL δεν μας δίνεται η δυνατότητα ελέγχου αλλαγής και ταυτόχρονης τροποποίησης ενός table στο ίδιο trigger). Η πρώτη διαδικασία χρησιμοποιείται για την εισαγωγή εισιτηρίων στο table resale_queue (ουρά μεταπώλησης) από το table ticket σε περίπτωση που κάποιος επισκέπτης που κατέχει ένα μη ενεργοποιημένο εισιτήριο (**activated = FALSE**) για μία παράσταση και θέλει να το μεταπωλήσει (**is_resale = TRUE**). Η διαδικασία ψάχνει μέσα στο table ticket να βρει δεδομένα που έχουν αυτές τις δύο τιμές. Εάν βρεθεί κάποιο εισιτήριο που είναι μη ενεργοποιημένο και ο κάτοχός του θέλει να το μεταπωλήσει, τότε από αυτό το εισιτήριο εξάγουμε τα απαραίτητα attributes και βάσει αυτών κάνουμε insert στο table resale_queue, εισάγοντας έτσι έμμεσα αυτό το εισιτήριο στην ουρά μεταπώλησης. Παράλληλα, για να μη χάσουμε τις παραπάνω απαραίτητες πληροφορίες του εισιτηρίου (ean_code, payment_method) και να μη τις μεταφέρουμε επίσης και στο

resale_queue, επιλέγουμε να θέτουμε τις πληροφορίες για τον κάτοχο και την ημερομηνία αγοράς σε NULL, αντί να διαγράφεται εντελώς το ticket_id από το table και να χάνονται οι πληροφορίες (έτσι κι αλλιώς το ticket υπάρχει απλά δεν έχει κάποιον κάτοχο όταν μπει στην ουρά μεταπώλησης). Έτσι, το εισιτήριο αυτό έχει μπει στην ουρά μεταπώλησης και στη συνέχεια αν βρεθεί κάποιος ενδιαφερόμενος για αυτό και αγοραστεί ξανά, η αντιστοίχιση στο table ticket θα γίνει μέσω του ticket_id και οι τιμές visitor_id και payment_date θα ανανεωθούν στον νέο αγοραστή και την ημερομηνία που το αγόρασε αυτός αντίστοιχα. Έτσι μάλιστα, διατηρούμε και την πληροφορία για την τιμή αγοράς αφού αυτή στη μεταπώληση δεν αλλάζει οπότε αφού δεν τροποποιείται στο table ticket δεν υπάρχει πρόβλημα. Τώρα, για να μπορέσουμε να έχουμε έναν σχετικά διαρκή έλεγχο στο table ticket για το αν κάποιος έχει αλλάξει γνώμη και θέλει να διαθέσει το εισιτήριό του για μεταπώληση, δημιουργούμε ένα EVENT για αυτή τη διαδικασία κάθε 5 δευτερόλεπτα. Έτσι ουσιαστικά κάθε 5 δευτερόλεπτα θα ελέγχεται το table ticket για να βλέπουμε εάν κάποιος επισκέπτης θέλει να διαθέσει ένα εισιτήριό του για μεταπώληση (Υπήρχε και εναλλακτική ιδέα με trigger και πιο αποδοτική που θα έλεγχε εάν υπήρχε UPDATE στο ticket στην τιμή is_resale και τότε θα έκανε τα ακόλουθα UPDATE στο ticket αλλά αυτό στη MySQL δεν εφαρμόζεται). Η δεύτερη διαδικασία χρησιμοποιείται για να αντιστοιχίζει κάποιους πιθανούς ενδιαφερόμενους στην ουρά αναμονής με εισιτήρια στην ουρά μεταπώλησης για παραστάσεις και κατηγορίες εισιτηρίων για τα οποία ενδιαφέρονται. Και στις δύο ουρές ακολουθείται λειτουργία με σειρά FIFO δηλαδή φροντίζεται πάντα αν ένα εισιτήριο στην ουρά μεταπώλησης μπορεί να δοθεί σε κάποιον ενδιαφερόμενο δηλαδή υπάρχουν ενδιαφερόμενοι για την παράσταση και το είδος αυτού του εισιτηρίου να επιλέγεται πάντα αυτός που εκδήλωσε πρώτος το ενδιαφέρον αυτό και αντίστοιχα αν υπάρχουν πολλά εισιτήρια που μπορεί να αντιστοιχούν σε έναν ενδιαφερόμενο να επιλέγεται πάντα αυτό που τοποθετήθηκε στην ουρά αναμονής πιο παλιά. Εάν επομένως γίνει μία μεταπώληση, στη συνέχεια διαγράφονται από τις ουρές και ο ενδιαφερόμενος αυτός και το εισιτήριο που αγοράστηκε ξανά και ταυτόχρονα μέσω του ticket_id που το έχουμε κρατήσει στο resale_queue βρίσκουμε τη θέση του στο table ticket (που θυμίζουμε πως δεν είχε διαγραφεί εντελώς) και ανανεώνουμε τα πεδία visitor_id, payment_date και το is_resale αντίστοιχα στις τιμές visitor_id του νέου ενδιαφερόμενου, την τωρινή ημερομηνία και FALSE. Έτσι, πλέον το εισιτήριο αυτό με το ticket_id δεν έχει NULL πεδία καθώς έχει νέο κάτοχο. Τονίζεται πως για να γίνουν όλα αυτά πιο εύκολα στη διαδικασία χρησιμοποιήθηκε ένα προσωρινό table για τη μετάβαση και σύγκριση δεδομένων πιο εύκολα το οποίο προφανώς στο τέλος της διαδικασίας φροντίζουμε να το κάνουμε DROP. Αν δεν βρεθεί κάποιος ενδιαφερόμενος για τα εισιτήρια στην ουρά μεταπώλησης ή δε βρεθεί κάποιο εισιτήριο για έναν ενδιαφερόμενο τότε δεν γίνεται κάποια ενέργεια για αυτές τις περιπτώσεις, άρα αυτά τα tuples παραμένουν στα tables resale_interest και resale_queue αντίστοιχα μέχρι ίσως μελλοντικά βρεθεί κάποια αντιστοίχιση. Για να μπορεί να υποστηριχθεί αυτή η λειτουργία και ο μελλοντικός έλεγχος, επειδή και εδώ δεν μπορούν να χρησιμοποιηθούν triggers για την τροποποίηση του ίδιου table μέσα στο trigger, δημιουργούμε και εδώ ένα EVENT

κάθε 15 δευτερόλεπτα για τη δεύτερη συνάρτηση ώστε να ελέγχονται οι πίνακες για πιθανή αντιστοίχιση.

3.4 Indexes

Για την πιο αποτελεσματική υλοποίηση των queries, και για την επιτάχυνση της εκτέλεσης τους, αξιοποιούμε διάφορα indexes στο DDL. Με τη χρήση των indexes, όταν εκτελείται ένα ερώτημα με συνθήκη (SELECT… WHERE…), σύνδεση πινάκων (JOIN…) ή ταξινόμηση (ORDER BY…)(GROUP BY…), η βάση μπορεί να χρησιμοποιήσει το ευρετήριο για να βρει μόνο τις εγγραφές που χρειάζεται, χωρίς να φορτώνει ολόκληρο τον πίνακα στη μνήμη. Πιο συγκεκριμένα, έχουμε ορίσει ευρετήρια για γρήγορη πρόσβαση:

- στη χρονια που διεξαγεται το festival,
- στη σκηνη στην οποια βρισκεται καποιο equipment
- στα μελη καθε γκρουπ
- στα genres καθε καλλιτεχνη
- στις προγραμματισμενες εμφανισεις καθε καλλιτεχνη
- στα εισητηρια που αντιστοιχουν σε εναν επισκεπτη
- στις αξιολογησεις που εχουν κανει οι επισκεπτες
- και στις αναθεσεις του προσωπικου με βαση το event και την ημερομηνια

Άρα μπορούμε να τρέχουμε με μεγαλύτερη ταχύτητα οποιοδήποτε query σχετίζεται με τα παραπάνω.

4. Generating Dummy Data

Μετά την δημιουργία του DDL της βάσης, επόμενο βήμα είναι το πέρασμα δεδομένων με χρήση dummy_data για να μπορέσουμε να εξακριβώσουμε την ορθή λειτουργία της βάσης. Για την δημιουργία των δεδομένων υλοποιήθηκε ένα python script αντικειμενοστραφής λογική όπου υλοποιήθηκαν ξεχωριστές συναρτήσεις για κάθε ξεχωριστό table οι οποίες δημιουργούν δυναμικά δεδομένα για την βάση δεδομένων. Η δυναμική δημιουργία συνεπάγεται στις περιπτώσεις όπου θέλουμε τα δεδομένα να τηρούν τους κανόνες που έχουμε δημιουργήσει μεσω των triggers διότι σε διαφορετική περίπτωση τα δεδομένα δεν θα αποθηκευτούν ποτέ στην βάση λόγω εκτέλεσης των triggers, οπότε πολλές φορές δημιουργείται το αρχικό table κι έπειτα το table το οποιο εξαρτάται από τα δεδομένα αυτού προκειμένου υπολογίζονται οι παράμετροι σωστά (όπως για παράδειγμα η χωρητικότητα των stages σε συσχέτιση με τα εισιτήρια). Έπειτα οι συναρτήσεις (generate.function) πρέπει να κληθούν με την σωστή σειρά ώστε τα δεδομένα του αρχικού table να έχουν παραχθεί για να μπορέσουν να δημιουργηθούν τα επόμενα εξαρτημένα από αυτό tables. Στο τέλος, δημιουργείται το τελικό αρχείο **load.sql** στο οποίο έχουν αποθηκευτεί για κάθε ξεχωριστό table

οι INSERTION εντολές με την χρήση της εντολή (**INSERT INTO** table () **VALUES …**). Σε αυτό το σημείο είμαστε σε θέση να φορτώσουμε τα dummy_data στην βάση.

Προκειμένου τα δεδομένα μας να είναι ρεαλιστικά χρησιμοποιήθηκε η συνάρτηση της python faker. Η βιβλιοθήκη **Faker** είναι σχεδιασμένη να παράγει ψεύτικα (fake) δεδομένα για δοκιμές, γέμισμα βάσεων, anonymization κ.λπ. Κεντρικό της συστατικό είναι το σύστημα των *providers*, δηλαδή μικρών υπο-βιβλιοθηκών που προσφέρουν συγκεκριμένο τύπο δεδομένων. Για παράδειγμα αν επιθυμούμε την δημιουργία μια διεύθυνσης καλούμε τον provider → faker.providers.address. Σημαντική σημείωση για την Faker είναι πώς δεν είναι ενσωματωμένη στο standar πακέτο βιβλιοθηκών της python και για την προσθήκη της εκτελούμε την εντολή (σε περιβάλλον linux) **pip install faker** σε virtual environment επειδή η python απαγορεύει την αλλαγή των βιβλιοθηκών της.

5. Queries

1. Βρείτε τα έσοδα του φεστιβάλ, ανά έτος από την πώληση εισιτηρίων, λαμβάνοντας υπόψη όλες τις κατηγορίες εισιτηρίων και παρέχοντας ανάλυση ανά είδος πληρωμής.

Για να υλοποιήσουμε αυτό το ερώτημα ενώνουμε τους πίνακες ticket, event και festival για να φέρουμε σε επαφή το έτος του φεστιβάλ και τον τρόπο πληρωμής κάθε εισιτηρίου. Στη συνέχεια, αθροίζουμε την τιμή (price) όλων των εισιτηρίων για κάθε συνδυασμό έτους και μεθόδου πληρωμής. Σχηματίζουμε ομάδες ανά έτος και μέθοδο πληρωμής ώστε το άθροισμα να υπολογίζεται σωστά σε κάθε ομάδα. Τέλος, ταξινομούμε τα αποτελέσματα κατά αύξον έτος και, εντός κάθε έτους, κατά τρόπο πληρωμής.

code:

```
SELECT
    f.year,
    t.payment_method,
    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
GROUP BY
    f.year, t.payment_method
ORDER BY
    f.year, t.payment_method;
```

2. Βρείτε όλους τους καλλιτέχνες που ανήκουν σε ένα συγκεκριμένο μουσικό είδος με ένδειξη αν συμμετείχαν σε εκδηλώσεις του φεστιβάλ για το συγκεκριμένο έτος;

Για να υλοποιήσουμε αυτό το ερώτημα ενώνουμε τους πίνακες Artist, artist_genres και Genre ώστε να φέρουμε σε επαφή το artist_id, το όνομα, το σκηνικό όνομα και το είδος κάθε καλλιτέχνη. Με LEFT JOIN εισάγουμε ένα υποερώτημα που επιλέγει distinct artist_id από performance_members, Performance, Event και Festival φιλτράροντας για το ζητούμενο έτος ώστε να εντοπίσουμε ποιοι καλλιτέχνες συμμετείχαν στο φεστιβάλ εκείνη τη χρονιά. Στη συνέχεια χρησιμοποιούμε CASE … WHEN pm.artist_id IS NOT NULL THEN 'Ναι' ELSE 'Όχι' για να εμφανίσουμε στη στήλη “συμμετοχή_φέτος” αν ο καλλιτέχνης έπαιξε στο φεστιβάλ του συγκεκριμένου έτους. Τέλος με WHERE g.name = 'Rock' φιλτράρουμε μόνο τους καλλιτέχνες του είδους Rock.

code:

```
SELECT
    a.artist_id,
    a.name,
    a.stage_name,
    g.name AS genre_name,
    CASE
        WHEN pm.artist_id IS NOT NULL THEN 'Ναι'  -- Αλλαγή
εδώ
        ELSE 'Όχι'
    END AS συμμετοχή_φέτος
FROM
    Artist a
JOIN
    Artist_Genres ag ON a.artist_id = ag.artist_id
JOIN
    Genre g ON ag.genre_id = g.genre_id
LEFT JOIN (
    SELECT
        DISTINCT artist_id  -- Εδώ επιστρέφουμε μόνο
artist_id
    FROM
        Performance_members pm
    JOIN
            performance p ON p.performance_id =
pm.performance_id
    JOIN
        Event e ON p.event_id = e.event_id
    JOIN
        Festival f ON e.festival_id = f.festival_id
    WHERE
        f.year = 2023
```

```
) pm ON a.artist_id = pm.artist_id
WHERE
    g.name = 'Rock';
```

3. Βρείτε ποιοι καλλιτέχνες έχουν εμφανιστεί ως warm up περισσότερες από 2 φορές στο ίδιο φεστιβάλ;

Για να απαντήσουμε σε αυτό το ερώτημα, ενώνουμε τους πίνακες Performance_members, Performance, Artist, Event και Festival ώστε να έχουμε πρόσβαση στους καλλιτέχνες, στον τύπο της εμφάνισης και στο φεστιβάλ. Φιλτράρουμε ώστε να ληφθούν υπόψη μόνο οι εμφανίσεις τύπου 'warm up'. Ομαδοποιούμε τα αποτελέσματα ανά καλλιτέχνη και φεστιβάλ, μετρώντας πόσες φορές κάθε καλλιτέχνης εμφανίστηκε ως warm up. Χρησιμοποιούμε HAVING COUNT(*) > 2 για να κρατήσουμε μόνο όσους είχαν περισσότερες από δύο τέτοιες εμφανίσεις στο ίδιο φεστιβάλ. Τέλος, ταξινομούμε τα αποτελέσματα φθίνουσα κατά αριθμό εμφανίσεων.

code:

```
SELECT
    a.artist_id,
    a.name,
    a.stage_name,
    f.festival_id,
    f.year,
    COUNT(*) AS warm_up_appearances
FROM
    Performance_members pm
JOIN
     performance p ON pm.performance_id = p.performance_id
JOIN
    Artist a ON pm.artist_id = a.artist_id
JOIN
    Event e ON p.event_id = e.event_id
JOIN
    Festival f ON e.festival_id = f.festival_id
WHERE
    p.type = 'warm up'
GROUP BY
    a.artist_id, a.name, a.stage_name, f.festival_id, f.year
HAVING
    COUNT(*) > 2
ORDER BY
    warm_up_appearances DESC;
```

4. Για κάποιο καλλιτέχνη, βρείτε το μέσο όρο αξιολογήσεων (Ερμηνεία καλλιτεχνών) και εμφάνιση (Συνολική εντύπωση).

Για να απαντήσουμε στο ερώτημα, ενώνουμε τους πίνακες Artist, Performance_members και Review ώστε να συνδέσουμε κάθε καλλιτέχνη με τις αξιολογήσεις των εμφανίσεών του. Φιλτράρουμε για τον συγκεκριμένο καλλιτέχνη με βάση το artist_id. Υπολογίζουμε τον μέσο όρο της αξιολόγησης της ερμηνείας και της συνολικής εντύπωσης και στρογγυλοποιούμε τα αποτελέσματα σε δύο δεκαδικά ψηφία. Ομαδοποιούμε με βάση τον καλλιτέχνη για να γίνει σωστά ο υπολογισμός.

code:

```
SELECT
    a.artist_id,
    a.name AS artist_name,
    a.stage_name,
    ROUND(AVG(r.interpretation), 2) AS μέσος_όρος_ερμηνείας,
    ROUND(AVG(r.overall_impression), 2) AS
μέσος_όρος_συνολικής_εντύπωσης
FROM
    Artist a
JOIN
    Performance_members pm ON a.artist_id = pm.artist_id
JOIN
    Review r ON pm.performance_id = r.performance_id
WHERE
    a.artist_id = 30   -- Αντικαταστήστε με το επιθυμητό
artist_id
GROUP BY
    a.artist_id, a.name, a.stage_name;
```

Εναλλακτικό query plan χρησιμοποιωντας force index:

```
SELECT
    a.artist_id,
    a.name AS artist_name,
    a.stage_name,
    ROUND(AVG(r.interpretation), 2) AS μέσος_όρος_ερμηνείας,
    ROUND(AVG(r.overall_impression), 2) AS
μέσος_όρος_συνολικής_εντύπωσης
FROM
    Artist a
JOIN
    Performance_members pm FORCE
INDEX(idx_performance_artist)
    ON a.artist_id = pm.artist_id
JOIN
     Review r FORCE INDEX(idx_review_performance)
     ON pm.performance_id = r.performance_id
WHERE
    a.artist_id = 20   -- Αντικαταστήστε με το επιθυμητό
artist_id
```

```
GROUP BY
    a.artist_id, a.name, a.stage_name;
```

Το εναλλακτικό query plan χρησιμοποιεί αναγκαστικά index που υποδεικνύουμε. Το FORCE INDEX αναγκάζει την MySQL να χρησιμοποιήσει τον συγκεκριμένο index χωρίς να αφήσει τον optimizer να επιλέξει μόνος του, με αποτέλεσμα σε περίπτωση που ο optimizer δεν χρησιμοποιεί τον συγκεκριμένο index, που μπορεί να είναι πιο αποδοτικός, να έχει το αρχικό πλάνο μεγαλύτερο χρόνο εκτέλεσης. Εκτελώντας τους παρακάτω κωδικούς γράφοντας EXPLAIN πριν από τους κωδικούς των queries, βλέπουμε πληροφορίες για το πώς ο optimizer εκτελεί το ερώτημα.

```
EXPLAIN
SELECT
    a.artist_id,
    a.name AS artist_name,
    a.stage_name,
    ROUND(AVG(r.interpretation), 2) AS μέσος_όρος_ερμηνείας,
    ROUND(AVG(r.overall_impression), 2) AS
μέσος_όρος_συνολικής_εντύπωσης
FROM
    Artist a
JOIN
     Performance_members pm
     ON a.artist_id = pm.artist_id
JOIN
     Review r
     ON pm.performance_id = r.performance_id
WHERE
    a.artist_id = 20   -- Αντικαταστήστε με το επιθυμητό
artist_id
GROUP BY
    a.artist_id, a.name, a.stage_name;
```

Απο αυτον τον κωδικα παιρνουμε το παρακατω table:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	pm	ref	uk_perf_member,idx_performance_artist	idx_performance_artist	5	const	4	
1	SIMPLE	r	ref	idx_review_performance	idx_review_performance	5	music_festival.pm.performance_id	1	

Εδώ φαίνεται πως ακόμα και χωρίς το FORCE INDEX, ο optimizer επιλέγει τα επιθυμητά indexes, τα οποία μάλιστα φαίνεται πως είναι και οι βέλτιστοι. Για επιβεβαίωση, εκτελούμε τον κώδικα με το FORCE INDEX, προσθέτοντας EXPLAIN, και παίρνουμε τον παρακάτω πίνακα, ο οποίος είναι ίδιος.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	pm	ref	idx_performance_artist	idx_performance_artist	5	const	4	
1	SIMPLE	r	ref	idx_review_performance	idx_review_performance	5	music_festival.pm.performance_id	1	

Αν τωρα παμε στο αρχικο query plan και πριν το explain προσθεσουμε:

```
SET optimizer_switch =
'mrr=on,mrr_cost_based=off,join_cache_bka=on';
```

```
SET join_cache_level = 8;
SET join_buffer_size = 4 * 1024 * 1024;   -- 4MB
```

τοτε τρεχοντας ξανα αυτον τον κωδικα παιρνουμε το εξης table:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	pm	ref	uk_perf_member,idx_performance_artist	idx_performance_artist	5	const	4	
1	SIMPLE	r	ref	idx_review_performance	idx_review_performance	5	music_festival.pm.performance_id	1	Using join buffer (flat, BKA join); Rowid-ordered...

Η αλλαγή στη στρατηγική του join από **Nested Loop** σε **Batched Key Access (BKA)** με **Multi-Range Read (MRR)** αποτελεί σημαντική βελτιστοποίηση για ερωτήματα σε μεγάλα datasets.

Στο **Nested Loop Join**, η βάση δεδομένων προσπελαύνει τον δεύτερο πίνακα γραμμή-προς-γραμμή για κάθε κλειδί του πρώτου πίνακα, κάτι που προκαλεί πολλές τυχαίες προσπελάσεις στο δίσκο (random I/O) και υψηλό κόστος, ιδιαίτερα όταν οι πίνακες είναι μεγάλοι.

Με το **BKA**, τα κλειδιά από τον πρώτο πίνακα (π.χ. `reviews.performance_id`) ομαδοποιούνται σε batches, αντί να χρησιμοποιούνται ένα-ένα. Αυτό επιτρέπει μαζική ανάκτηση των αντίστοιχων γραμμών από τον δεύτερο πίνακα (π.χ. `performances`), μειώνοντας δραματικά τον αριθμό των προσπελάσεων.

Το **MRR** προσθέτει ένα επιπλέον επίπεδο βελτιστοποίησης: ταξινομεί τα row IDs (ή τα primary keys) πριν την ανάκτηση των δεδομένων, επιτρέποντας σειριακή προσπέλαση (sequential I/O), κάτι που είναι πολύ πιο αποδοτικό σε επίπεδο αποθήκευσης.

5. Βρείτε τους νέους καλλιτέχνες (ηλικία < 30 ετών) που έχουν τις περισσότερες συμμετοχές σε φεστιβάλ;

Για να απαντήσουμε στο ερώτημα, ενώνουμε τους πίνακες Artist, Performance_members, Performance, Event και Festival για να εντοπίσουμε τις συμμετοχές κάθε καλλιτέχνη σε φεστιβάλ. Υπολογίζουμε την ηλικία κάθε καλλιτέχνη με βάση την ημερομηνία γέννησης και κρατάμε μόνο όσους είναι κάτω των 30 ετών. Ομαδοποιούμε τα αποτελέσματα ανά καλλιτέχνη και μετράμε πόσες φορές έχει συμμετάσχει. Τέλος, ταξινομούμε τις συμμετοχές κατά φθίνουσα σειρά και κρατάμε τους 10 με τις περισσότερες εμφανίσεις.

code:

```
SELECT
    a.artist_id,
    a.name,
    a.stage_name,
    TIMESTAMPDIFF(YEAR, a.dob, CURDATE()) AS age,
    COUNT(*) AS συμμετοχές
FROM
    Artist a
JOIN
    Performance_members pm ON a.artist_id = pm.artist_id
```

```
JOIN
     performance p ON p.performance_id = pm.performance_id
JOIN
    Event e        ON p.event_id     = e.event_id
JOIN
    Festival f     ON e.festival_id = f.festival_id
WHERE
    TIMESTAMPDIFF(YEAR, a.dob, CURDATE()) < 30
GROUP BY
    a.artist_id, a.name, a.stage_name, age
ORDER BY
    συμμετοχές DESC
LIMIT 10;
```

6. Για κάποιο επισκέπτη, βρείτε τις παραστάσεις που έχει παρακολουθήσει και το μέσο όρο της αξιολόγησης του, ανά παράσταση.

Για να απαντήσουμε στο ερώτημα, ενώνουμε τους πίνακες Visitor, Review, Performance και Event ώστε να φέρουμε μαζί τον επισκέπτη, τις παραστάσεις που έχει παρακολουθήσει και τις αντίστοιχες ημερομηνίες. Φιλτράρουμε για έναν συγκεκριμένο επισκέπτη με βάση το ID. Ομαδοποιούμε τα αποτελέσματα ανά επισκέπτη και παράσταση, και υπολογίζουμε τον μέσο όρο της αξιολόγησής του για την ερμηνεία και τη συνολική εντύπωση σε κάθε παράσταση.

code:

```
SELECT
    a.artist_id,
    a.name,
    a.stage_name,
    TIMESTAMPDIFF(YEAR, a.dob, CURDATE()) AS age,
    COUNT(*) AS συμμετοχές
FROM
    Artist a
JOIN
    Performance_members pm ON a.artist_id = pm.artist_id
JOIN
     performance p ON p.performance_id = pm.performance_id
JOIN
    Event e        ON p.event_id     = e.event_id
JOIN
    Festival f     ON e.festival_id = f.festival_id
WHERE
    TIMESTAMPDIFF(YEAR, a.dob, CURDATE()) < 30
GROUP BY
    a.artist_id, a.name, a.stage_name, age
ORDER BY
    συμμετοχές DESC
LIMIT 10;
```

Εναλλακτικό query plan χρησιμοποιωντας force index:

```
SELECT
  v.visitor_id,
  v.first_name,
  v.last_name,
  p.performance_id,
  e.event_date,
  ROUND(AVG(r.interpretation), 2)        AS
avg_interpretation,
  ROUND(AVG(r.overall_impression), 2)    AS avg_overall
FROM Visitor v FORCE INDEX(PRIMARY)
JOIN Review r FORCE INDEX(idx_review_performance)
  ON v.visitor_id = r.visitor_id
JOIN Performance p FORCE INDEX(PRIMARY)
  ON r.performance_id = p.performance_id
JOIN Event e FORCE INDEX(PRIMARY)
  ON p.event_id = e.event_id
WHERE v.visitor_id = 101   -- ή όποιο άλλο ID θες
GROUP BY
  v.visitor_id,
  v.first_name,
  v.last_name,
  p.performance_id,
  e.event_date;
```

Το εναλλακτικό query plan χρησιμοποιεί αναγκαστικά τα index που υποδεικνύουμε. Το FORCE INDEX αναγκάζει την MySQL να χρησιμοποιήσει τον συγκεκριμένο index χωρίς να αφήσει τον optimizer να επιλέξει μόνος του, με αποτέλεσμα, σε περίπτωση που ο optimizer δεν χρησιμοποιεί τον συγκεκριμένο index που είναι αποδοτικός, να έχει το αρχικό πλάνο μεγαλύτερο χρόνο εκτέλεσης. Εμείς εδώ, βέβαια, επιλέγουμε να χρησιμοποιήσουμε για δοκιμή ένα index το οποίο πιθανότατα να μην είναι βέλτιστο, καθώς το JOIN δε θα πρέπει να "κοιτάει" αυτό, αλλά αυτό θα επιβεβαιωθεί στη συνέχεια. Εκτελώντας τους παρακάτω κώδικες, γράφοντας EXPLAIN πριν από τους κώδικες των queries, βλέπουμε πληροφορίες για το πώς ο optimizer εκτελεί το ερώτημα.

```
EXPLAIN
SELECT
  v.visitor_id,
  v.first_name,
  v.last_name,
  p.performance_id,
  e.event_date,
  ROUND(AVG(r.interpretation), 2)        AS
avg_interpretation,
```

```
  ROUND(AVG(r.overall_impression), 2)     AS avg_overall
FROM Visitor v
JOIN Review r
  ON v.visitor_id = r.visitor_id
JOIN Performance p
  ON r.performance_id = p.performance_id
JOIN Event e
  ON p.event_id = e.event_id
WHERE v.visitor_id = 101    -- ή όποιο άλλο ID θες
GROUP BY
  v.visitor_id,
  v.first_name,
  v.last_name,
  p.performance_id,
  e.event_date;
```

Απο τον παραπανω κωδικα που δε χρησιμοποιει force index παιρνουμε το παρακατω table:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	v	const	PRIMARY	PRIMARY	4	const	1	Using temporary; Using filesort
1	SIMPLE	r	ref	visitor_id,idx_review_performance,idx_review_...	visitor_id	5	const	1	Using index condition
1	SIMPLE	p	eq_ref	PRIMARY,idx_performance_event	PRIMARY	4	music_festival.r.performance_id	1	Using where
1	SIMPLE	e	eq_ref	PRIMARY	PRIMARY	4	music_festival.p.event_id	1	

Εδω παρατηρουμε πως στις 3 περιπτώσεις χρησιμοποιείται το primary key για join και στη δευτερη δηλαδη για το δευτερο join χρησιμοποιείται το visitor_id. Εαν κανουμε την ιδια διαδικασια με τα force index παιρνουμε τον πινακα:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	v	const	PRIMARY	PRIMARY	4	const	1	Using temporary; Using filesort
1	SIMPLE	r	ALL	idx_review_performance	NULL	NULL	NULL	146	Using where
1	SIMPLE	p	eq_ref	PRIMARY	PRIMARY	4	music_festival.r.performance_id	1	Using where
1	SIMPLE	e	eq_ref	PRIMARY	PRIMARY	4	music_festival.p.event_id	1	

Απο το key:NULL και το rows:146 καταλαβαινουμε οτι ο idx_review_performance οντως εχει σημαντικη αρνητικη επιδραση στο query και το αποτελεσμα πιθανοτατα να ειναι και λανθασμενο (δηλαδη οχι μονο πιο αργο). Αρα, παρατηρουμε οτι με το force index σε περιπτωση που επιλεξουμε λανθασμενο index δεν παιρνουμε παντα το πιο αποδοτικο αποτελεσμα.

7. Βρείτε ποιο φεστιβάλ είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού;

Για να βρούμε ποιο φεστιβάλ είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού, ενώνουμε τους πίνακες Festival, Event, Staff_Assignment και Staff, λαμβάνοντας υπόψη μόνο τις αναθέσεις με ρόλο «τεχνικός». Αντιστοιχίζουμε κάθε επίπεδο εμπειρίας σε αριθμητική τιμή (π.χ. intern=1, junior=2, κ.λπ.) και υπολογίζουμε τον μέσο όρο ανά φεστιβάλ. Στο τέλος, ταξινομούμε αυξανόμενα βάσει του μέσου όρου και επιλέγουμε το πρώτο αποτέλεσμα.

code:

```sql
SELECT
  f.festival_id,
  f.name AS festival_name,
  ROUND(AVG(
    CASE s.experience_level
      WHEN 'intern'      THEN 1
      WHEN 'junior'      THEN 2
      WHEN 'average'     THEN 3
      WHEN 'experienced' THEN 4
      WHEN 'senior'      THEN 5
    END
  ), 2) AS avg_experience
FROM Festival f
JOIN Event e
  ON f.festival_id = e.festival_id
JOIN Staff_Assignment sa
  ON e.event_id = sa.event_id
  AND sa.staff_role = 'technician' -- technician
JOIN Staff s
  ON sa.staff_id = s.staff_id
GROUP BY
  f.festival_id,
  f.name
ORDER BY
  avg_experience ASC
LIMIT 1;
```

8. Βρείτε το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία;

Για να βρούμε ποιο προσωπικό υποστήριξης δεν έχει προγραμματισμένη εργασία σε μια συγκεκριμένη ημερομηνία, επιλέγουμε όλα τα άτομα με ρόλο "support" από τον πίνακα Staff και κάνουμε αριστερή ένωση με τον πίνακα Staff_Assignment για την επιλεγμένη ημερομηνία και ρόλο. Τέλος, φιλτράρουμε τα αποτελέσματα ώστε να κρατήσουμε μόνο εκείνους που δεν έχουν αντιστοιχιστεί σε κάποια εργασία εκείνη την ημέρα.

code:

```sql
SELECT
  s.staff_id,
  s.name,
  s.staff_role
FROM Staff s
LEFT JOIN Staff_Assignment sa
  ON s.staff_id = sa.staff_id
  AND sa.assignment_date = '2025-06-15'    -- επιλεγμένη ημερομηνία
  AND sa.staff_role = 'support'
```

```
WHERE
  s.staff_role = 'support'
  AND sa.assignment_id IS NULL;
```

9. Βρείτε ποιοι επισκέπτες έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με περισσότερες από 3 παρακολουθήσεις;

Για να εντοπίσουμε επισκέπτες που έχουν παρακολουθήσει πάνω από 3 διαφορετικές παραστάσεις μέσα σε ένα έτος, ενώνουμε τους πίνακες Visitor, Ticket, Performance και Event ώστε να συνδέσουμε κάθε επισκέπτη με τις παραστάσεις που παρακολούθησε. Φιλτράρουμε τις παραστάσεις που πραγματοποιήθηκαν εντός του 2024, ομαδοποιούμε ανά επισκέπτη και κρατάμε μόνο όσους έχουν παρακολουθήσει περισσότερες από 3 διακριτές παραστάσεις.

code:

```
SELECT
    v.visitor_id,
    v.first_name,
    v.last_name,
    COUNT(DISTINCT p.performance_id) AS παραστάσεις
FROM
    Visitor v
JOIN
    Ticket t
  ON v.visitor_id = t.visitor_id
JOIN
    Performance p
  ON t.event_id = p.event_id
JOIN
    Event e
  ON p.event_id = e.event_id
WHERE
    -- Φιλτράρουμε τις παραστάσεις στο διάστημα ενός έτους
    e.event_date BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY
    v.visitor_id, v.first_name, v.last_name
HAVING
    COUNT(DISTINCT p.performance_id) > 3;
```

10. Πολλοί καλλιτέχνες καλύπτουν περισσότερα από ένα μουσικά είδη. Ανάμεσα σε ζεύγη πεδίων (π.χ. ροκ, τζαζ) που είναι κοινά στους καλλιτέχνες, βρείτε τα 3 κορυφαία (top-3) ζεύγη που εμφανίστηκαν σε φεστιβάλ.

Για να βρούμε τα 3 πιο συχνά εμφανιζόμενα ζεύγη μουσικών ειδών που έχουν εκπροσωπηθεί από τον ίδιο καλλιτέχνη σε φεστιβάλ, ενώνουμε τον πίνακα Artist με τον πίνακα Artist_Genres δύο φορές, ώστε να πάρουμε κάθε δυνατή

διαφορετική δυάδα ειδών ανά καλλιτέχνη. Χρησιμοποιούμε τη συνθήκη g1.genre_id < g2.genre_id για να αποφύγουμε διπλομέτρηση (π.χ. Rock–Jazz και Jazz–Rock να μετρηθούν μόνο μία φορά). Έπειτα, συνδέουμε τον καλλιτέχνη με τις εμφανίσεις του στα φεστιβάλ, και μετράμε για κάθε τέτοιο ζεύγος σε πόσα διαφορετικά φεστιβάλ εμφανίστηκε. Τέλος, ταξινομούμε φθίνουσα με βάση τον αριθμό εμφανίσεων και κρατάμε μόνο τα 3 κορυφαία ζεύγη.

code:

```
SELECT
    g1.name AS genre1,
    g2.name AS genre2,
    COUNT(DISTINCT f.festival_id) AS ζεύγη_εμφανίσεων
FROM Artist a

  -- πρώτο genre
  JOIN Artist_Genres ag1
    ON a.artist_id = ag1.artist_id
  JOIN Genre g1
    ON ag1.genre_id = g1.genre_id

  -- δεύτερο genre, αποφεύγουμε διπλομετρήσεις με genre_id
σύγκριση
  JOIN Artist_Genres ag2
    ON a.artist_id = ag2.artist_id
  JOIN Genre g2
    ON ag2.genre_id = g2.genre_id
    AND g1.genre_id < g2.genre_id

  -- συμμετοχές σε φεστιβάλ
  JOIN Performance_members pm
    ON a.artist_id = pm.artist_id
  JOIN     performance p
     ON pm.performance_id = p.performance_id
  JOIN Event e
    ON p.event_id = e.event_id
  JOIN Festival f
    ON e.festival_id = f.festival_id

GROUP BY
    g1.name, g2.name

ORDER BY
    ζεύγη_εμφανίσεων DESC

LIMIT 3;
```

11. Βρείτε όλους τους καλλιτέχνες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον καλλιτέχνη με τις περισσότερες συμμετοχές σε φεστιβάλ.

Για να εντοπίσουμε τους καλλιτέχνες που έχουν εμφανιστεί σε τουλάχιστον 5 λιγότερα φεστιβάλ από τον πιο ενεργό καλλιτέχνη, αρχικά με το CTE max_participations υπολογίζουμε ποιος έχει τις περισσότερες συμμετοχές σε διαφορετικά φεστιβάλ. Στη συνέχεια, αναζητούμε όλους τους άλλους καλλιτέχνες, υπολογίζοντας τις δικές τους συμμετοχές, και κρατάμε μόνο όσους έχουν αριθμό συμμετοχών μικρότερο κατά τουλάχιστον 5 σε σχέση με το μέγιστο.

code:

```
WITH max_participations AS (
  SELECT
    pm.artist_id,
    COUNT(DISTINCT e.festival_id) AS participations
  FROM Performance_members pm
  JOIN performance p ON p.performance_id = pm.performance_id
  JOIN Event e
    ON p.event_id = e.event_id
  GROUP BY pm.artist_id
  ORDER BY participations DESC
  LIMIT 1
)
SELECT
  a.artist_id,
  a.name              AS artist_name,
  COUNT(DISTINCT e.festival_id) AS participations
FROM Artist a
JOIN Performance_members pm
  ON a.artist_id = pm.artist_id
JOIN performance p
  ON p.performance_id = pm.performance_id
JOIN Event e
  ON p.event_id = e.event_id
GROUP BY
  a.artist_id,
  a.name
HAVING
  COUNT(DISTINCT e.festival_id) <= (
    SELECT participations
    FROM max_participations
  ) - 5;
```

12. Βρείτε το προσωπικό που απαιτείται για κάθε ημέρα του φεστιβάλ, παρέχοντας ανάλυση ανά κατηγορία (τεχνικό προσωπικό ασφαλείας, βοηθητικό προσωπικό);

Για να υπολογίσουμε το προσωπικό που απαιτείται ανά ημέρα του φεστιβάλ και ανά κατηγορία, ενώνουμε τους πίνακες Festival, Event και

Staff_Assignment, φιλτράροντας μόνο τις κατηγορίες προσωπικού που μας ενδιαφέρουν (τεχνικοί, ασφάλεια, υποστήριξη). Στη συνέχεια, ομαδοποιούμε τα δεδομένα ανά φεστιβάλ, ημερομηνία και ρόλο προσωπικού, ώστε να μετρήσουμε πόσα μοναδικά άτομα αντιστοιχούν σε κάθε συνδυασμό. Τέλος, ταξινομούμε τα αποτελέσματα ανά φεστιβάλ, ημερομηνία και κατηγορία προσωπικού.

code:

```
SELECT
    f.festival_id,
    f.name             AS festival_name,
    e.event_date       AS festival_date,
    sa.staff_role      AS category,
    COUNT(DISTINCT sa.staff_id) AS required_personnel
FROM Festival f
JOIN Event e
  ON f.festival_id = e.festival_id
JOIN Staff_Assignment sa
  ON e.event_id    = sa.event_id
     AND sa.staff_role IN ('technician','security','support')
GROUP BY
    f.festival_id,
    f.name,
    e.event_date,
    sa.staff_role
ORDER BY
    f.festival_id,
    e.event_date,
    sa.staff_role;
```

13. Βρείτε τους καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους.

Για να βρουμε τους καλλιτέχνες που έχουν συμμετασχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους, ενώνουμε τους πίνακες Artist, Performance_members, Performance, Event, Stage και Location για να έχουμε πρόσβαση στις ηπείρους που σχετίζονται με τις παραστάσεις του καλλιτέχνη. Στη συνέχεια, χρησιμοποιούμε την COUNT με το DISTINCT για να υπολογίσουμε τον αριθμό των μοναδικών ηπείρων στις οποίες έχει συμμετάσχει κάθε καλλιτέχνης. Τέλος, φιλτράρουμε για να δείξουμε μόνο τους καλλιτέχνες που έχουν συμμετάσχει σε τουλάχιστον 3 διαφορετικές ηπείρους.

code:

```
SELECT
    a.artist_id,
    a.name,
    COUNT(DISTINCT loc.continent) AS continents_count
FROM Artist a
```

```
JOIN performance_members pm
    ON a.artist_id = pm.artist_id
JOIN performance p
    ON pm.performance_id = p.performance_id
JOIN event e
    ON p.event_id = e.event_id
JOIN festival f
    ON e.festival_id = f.festival_id
JOIN location loc
    ON f.location_id = loc.location_id
GROUP BY
    a.artist_id,
    a.name
HAVING
    COUNT(DISTINCT loc.continent) >= 3;
```

14. Βρείτε ποια μουσικά είδη είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές με τουλάχιστον 3 εμφανίσεις ανά έτος;

Θελουμε να βρουμε ποια μουσικά είδη είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές, με τουλάχιστον 3 εμφανίσεις ανά έτος. Αρχικά, δημιουργείται ένα CTE (Common Table Expression) που υπολογίζει για κάθε μουσικό είδος τον αριθμό των εμφανίσεων σε κάθε έτος, με τον περιορισμό να είναι τουλάχιστον 3 εμφανίσεις ανά έτος. Επειτα συγκρίνουμε τα αποτελέσματα για δύο συνεχόμενα έτη (με την συνθήκη gc2.year = gc1.year + 1), και για τα μουσικά είδη που έχουν τον ίδιο αριθμό εμφανίσεων στις δύο χρονιές. Το τελικό αποτέλεσμα είναι μια λίστα με τα μουσικά είδη, τις δύο χρονιές και τον αριθμό των εμφανίσεων, όπου ο αριθμός εμφανίσεων είναι ίδιος για τις δύο χρονιές και πληρούνται οι προϋποθέσεις του ερωτήματος.

code:

```
WITH genre_counts AS (
  SELECT
    g.genre_id,
    g.name          AS genre_name,
    f.year,
    COUNT(*)        AS cnt
  FROM Genre g
  JOIN Artist_Genres ag ON g.genre_id = ag.genre_id
  JOIN Performance_members pm   ON ag.artist_id =
pm.artist_id
  JOIN performance p ON p.performance_id = pm.performance_id
  JOIN Event e          ON p.event_id = e.event_id
  JOIN Festival f       ON e.festival_id = f.festival_id
  GROUP BY
    g.genre_id, g.name, f.year
  HAVING COUNT(*) >= 3
```

```
)
SELECT
  gc1.genre_name,
  gc1.year     AS year1,
  gc2.year     AS year2,
  gc1.cnt      AS performances
FROM genre_counts gc1
JOIN genre_counts gc2
  ON gc1.genre_id = gc2.genre_id
  AND gc2.year = gc1.year + 1
  AND gc2.cnt  = gc1.cnt;
```

15. Βρείτε τους top-5 επισκέπτες που έχουν δώσει συνολικά την υψηλότερη βαθμολόγηση σε ένα καλλιτέχνη. (όνομα επισκέπτη, όνομα καλλιτέχνη και συνολικό σκορ βαθμολόγησης);

Θελουμε τους 5 κορυφαίους επισκέπτες που έχουν δώσει τη υψηλότερη συνολική βαθμολόγηση σε έναν καλλιτέχνη. Αρχικά, γίνεται σύνδεση των πινάκων Visitor, Review, Performance_members και Artist. Στη συνέχεια, υπολογίζεται το συνολικό σκορ για κάθε επίσκεψη και καλλιτέχνη, το οποίο είναι το άθροισμα των πεδίων interpretation, lights_sound, stage_presence, organization, και overall_impression από τον πίνακα Review. Το αποτέλεσμα ομαδοποιείται ανά επισκέπτη και καλλιτέχνη (με τη χρήση του GROUP BY), και τα αποτελέσματα ταξινομούνται κατά το συνολικό σκορ με φθίνουσα σειρά. Τέλος, επιστρέφονται οι πρώτοι 5 επισκέπτες με τη μεγαλύτερη βαθμολογία, μαζί με το όνομα του επισκέπτη, του καλλιτέχνη και το συνολικό σκορ. Με αυτόν τον τρόπο, μπορούμε να δούμε ποιοι επισκέπτες έχουν δώσει την υψηλότερη βαθμολογία σε κάθε καλλιτέχνη.

code:

```
SELECT
    v.first_name          AS visitor_first_name,
    v.last_name           AS visitor_last_name,
    a.name                AS artist_name,
    SUM(
      r.interpretation +
      r.lights_sound +
      r.stage_presence +
      r.organization +
      r.overall_impression
    )                      AS total_score
FROM Visitor v
JOIN Review r
  ON v.visitor_id = r.visitor_id
JOIN performance_members pm
  ON r.performance_id = pm.performance_id
JOIN Artist a
```

```
  ON pm.artist_id = a.artist_id
GROUP BY
    v.visitor_id,
    a.artist_id
ORDER BY
    total_score DESC
LIMIT 5;
```

6. Ανάλυση εγκατάστασης server

Για την ενεργοποίηση του MySQL server απαιτείται η εγκατάσταση του πακέτου XAMPP ή εγκατάσταση του πακέτου εργαλείων από το επίσημο site της MySQL (https://dev.mysql.com/downloads/installer/).

Για την διαχείριση του server και την εισαγωγή των δεδομένων και στις δύο περιπτώσεις πρέπει μέσω του terminal να οδηγηθούμε στον φάκελο ο οποίος περιέχει το αρχείο mysql.exe και έπειτα εκτελούμε την εντολή **mysql -u root -p** όπου θεωρούμε ότι το username είναι by default το 'root και η ip του server είναι η 127.0.0.1 (loopbuck ip). Έπειτα αφού έχουμε συνδεθεί τρέχουμε την εντολή **sourse file_path** όπου file path η διαδρομή του αρχείου δεδομένων dummy_data. Εναλλακτικός τρόπος σύνδεση της βάσεις είναι η εγκατάσταση του client προγράμματος από το πακέτο εργαλείων της MySQL MySQL WORKBENCH το οποίο ανιχνεύει αυτόματα τον server στο port 3306 και ο χρήστης είναι έτοιμος να φορτώσει τα απαραίτητα αρχεία στον server.

7. Διεκπεραίωση Εργασίας

Για την υλοποίηση και διεκπεραίωση της εργασίας προκειμένου όλα τα μέλη της ομάδας να μπορούν να δουλεύουν ταυτόχρονα και σε ένα ενιαίο server εγκαταστάθηκε σε πλατφόρμα Raspberry pi 4 Ubuntu 25.04 (GNU/Linux 6.14.0-1005-raspi aarch64) όπου κατεβάσαμε τα απαραίτητα εργαλεία της MySQL MariaDB (server version → 11.4.5-MariaDB-1 Ubuntu 25.04). Για να έχουν όλα τα μέλη της ομάδας πρόσβαση στον server είναι απαραίτητη χρήση ενός VPN (στην δική μας περίπτωση χρησιμοποιήθηκε το VPN wireguard) το οποίο μπορεί να εγκατασταθεί ως εφαρμογή (wireguard server) στο ίδιο board. Έπειτα κάθε μέλος της ομάδας έχει το δικό του ξεχωριστό ‘κλειδί’ το οποίο αρχικοποιεί το **tunel** ανάμεσα στον τερματικό και στον server.
Επιπλέον λόγω του server μπορέσαμε και δημιουργήσαμε script το οποίο κάνει αυτόματα **backup** το DDL της βάσης σε συνδυασμό με τον scheduler του linux σε μία συγκεκριμένη ώρα της ημέρας.
παρακάτω παρατίθεται το script:

```
# where to backup
```

```
dest="/home/user/sambashare"

# create backup folder if not exist
mkdir -p $dest

# create archive filenames
day=$(date +%y-%m-%d)
hostname=$(hostname -s)
archive_file="$hostname-$day.tar"
mysql_file="$hostname-mysql-$day.tar"

# print start status message
echo "Backing up $backup_files to $dest/$archive_file ..."
echo "Backing up $backup_databases to $dest/$mysql_file ..."

# database dump in temp file
mysqldump --user=root --password=Dionisis@1 --routines
--triggers --single-transacti> "$dest/festival_backup.sql"
# pack the sql dump with tar and remove dump
tar -czf "$dest/$mysql_file" -C "$dest" "festival_backup.sql"
rm $dest/festival_backup.sql

# print end status message
echo "Backup SUCCESS"

# echo generated files
ls -lh $dest
```