ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2024-2025

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ

ΟΜΑΔΑ 1

ΓΕΩΡΓΙΟΣ ΠΑΛΛΗΣ 03122144

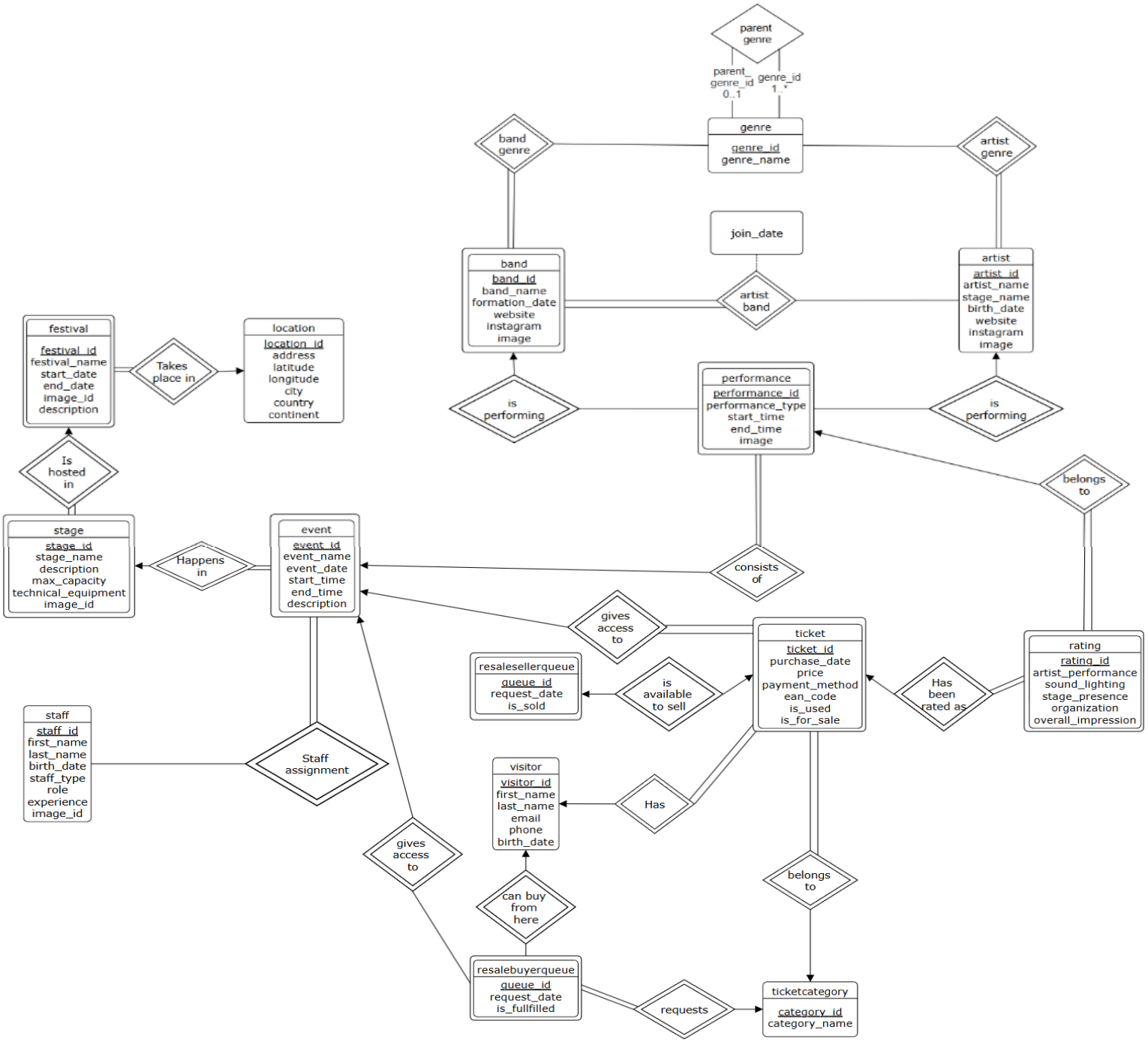
ΓΡΗΓΟΡΙΟΣ ΣΤΑΜΑΤΟΠΟΥΛΟΣ 03122039

ΝΙΚΟΛΑΟΣ ΔΙΟΝΥΣΙΟΣ ΦΡΑΓΚΟΣ 03122028

Περιεχόμενα:

Α. Σχεδιασμός και Υλοποίηση Βάσης Δεδομένων

1. ER Διάγραμμα



Στο παρόν σημείο θα αναλύσουμε τη λογική πίσω από τον σχεδιασμό του διαγράμματος Οντοτήτων-Σχέσεων (ER Diagram):

- Αρχικά, υλοποιήσαμε το **entity Location** το οποίο περιέχει τις τοποθεσίες των φεστιβάλ. Ως Primary key θέσαμε το location_id, το οποίο υπογραμμίζουμε, και το entity περιέχει τα παρακάτω attributes: address latitude longitude city country continent, δηλαδή την διεύθυνση του location, τις γεωγραφικές συντεταγμένες, τη πόλη, τη χώρα και την ήπειρο.

- Έπειτα, υλοποιήσαμε το **entity Festival**, το οποίο περιέχει τα φεστιβάλ που θα πραγματοποιηθούν, Ως Primary key θέσαμε το festival_id και περιέχονται τα παρακάτω_attributes:

    - festival_name, start_date, end_date, image_id, description

  δηλαδή το όνομα του κάθε festival, οι ημερομηνίες διεξαγωγής του, το id της φωτογραφίας του και μια μικρή περιγραφή για αυτό.

  Η σχέση που συνδέει τα **entities festival και location** αποτελεί σχέση **many to one**, καθώς κάθε έτος το festival διεξάγεται σε μία μοναδική τοποθεσία, ενώ ακόμη η τοποθεσία σχετίζεται με πολλά festival, με περιορισμό να μην είναι η ίδια σε δύο διαδοχικά έτη. Έτσι, έχουμε ακόμη πως η σχέση festival - location έχει **total participation**, καθώς τουλάχιστον ένα φεστιβάλ σχετίζεται με ένα location, άρα το festival με τη **σχέση takes place** in ενώνεται με **διπλή γραμμή**.

- Παρόμοια, υλοποιούμε το **entity Stage**, το οποίο έχει ως primary key το stage_id και ως άλλα attributes τα:

    - stage_name, description, max_capacity, technical_equipment, image_id

  και σχετίζεται με το festival με τη σχέση “is hosted in”. Η σχέση αυτή είναι πάλι many to one, καθώς ένα stage σχετίζεται με ένα ακριβώς festival, άρα θέλει και διπλή γραμμή από το stage στο relationship set “is hosted in”, ενώ ένα festival περιέχει πολλά stages, ανάλογα για πόσες μέρες διεξάγεται.

- Στην συνέχεια, υλοποιήσαμε το **entity Event**, όμοια με το stage, δηλαδή περιέχει ένα primary key το event_id, και άλλα attributes που περιγράφουν την κάθε παράσταση. Η σχέση event - stage είναι πάλι many to one, καθώς ένα event διεξάγεται αποκλειστικά

σε μία και μοναδική σκηνή - stage (επομένως και η διπλή γραμμή), ενώ μία σκηνή μπορεί να περιέχει πολλά events.

- Επιπλέον, δημιουργήσαμε το **entity Staff**, το οποίο μέσω του staff assignment ενώνεται με το entity event. Το staff έχει ως primary key το id του κάθε προσωπικού, και άλλα attributes, όπως name, birth_date, staff_type, role, experience τα οποία περιγράφουν το κάθε staff. Η σχέση του staff με το event είναι many to many, καθώς ένας άνθρωπος του προσωπικού μπορεί να βρίσκεται σε πολλά events, δεδομένου ότι δεν συμβαίνουν την ίδια ώρα, ενώ ένα event κάνει assign πολλά άτομα για προσωπικό. Μάλιστα, θέλει διπλή γραμμή στο event-staff assignment καθώς χρειάζεται έναν ελάχιστο αριθμό από προσωπικό για να υπάρξει το event, άρα total participation.

- Ακόμη, υλοποιήσαμε το **entity Performance**, το οποίο μέσω του consists of συνδέεται με το event. Περιέχει το primary key performance_id, καθώς και τα υπόλοιπα attributes:
    - performance_type, start και end time, image

  Η σχέση performance - event αποτελεί many to one, καθώς ένα event περιέχει πολλά performances, ενώ ένα performance σχετίζεται με ένα αποκλειστικά event για να υπάρξει, επομένως χρειάζεται και διπλή γραμμή.

- Από το performance δημιουργήσαμε στη συνέχεια τα εντελώς όμοια **entities Band και Artist**. Και τα δύο έχουν ως primary keys τα χαρακτηριστικά id τους (band_id, artist_id), ενώ για attributes περιέχουν στοιχεία που τα χαρακτηρίζουν:
    - band_name, formation_date, website, instagram, image για το band
    - artist_name, stage_name, birth_data, website, instagram, image για το artist

  Οι σχέσεις που συνδέουν performance - band , performance - artist είναι όμοιες και many to one, καθώς μία μπάντα/καλλιτέχνης μπορεί να λαμβάνει μέρος σε πολλά performances, ενώ ένα performance γίνεται/χαρακτηρίζεται από μία μπάντα ή ένα καλλιτέχνη.

- Για τη σχέση μεταξύ **band - artist** υλοποιήσαμε το **relationship set "artistband"**, και η σχέση αυτή είναι **many to many**, καθώς ένας καλλιτέχνης μπορεί να είναι μέλος πολλών band, και μια μπάντα μπορεί να αποτελείται από πολλούς καλλιτέχνες. Ακόμη,

από το band → artistband χρειάζεται **διπλή γραμμή**, καθώς για να υπάρξει μπάντα χρειάζεται η τουλάχιστον ένας καλλιτέχνη, άρα total participation, ενώ κάθε καλλιτέχνης δεν χρειάζεται μπάντα. Το relationship set έχει ακόμη attribute formation_date, για την ημερομηνία ένωσης της κάθε μπάντας.

- Στη συνέχεια, για τη διευκρίνιση του μουσικού είδους του κάθε καλλιτέχνη/μπάντας δημιουργήσαμε το **entity Genre**, με primary key το genre_id και άλλα attributes το όνομα του είδους (genre_name). Οι καλλιτέχνες και οι μπάντες σχετίζονται με αυτό μέσω των σχέσεων artist genre και band genre αντίστοιχα, και οι σχέσεις αυτές είναι **many to many** (καλλιτέχνης → πολλά είδη, ένα είδος → πολλοί καλλιτέχνες) και μάλιστα χρειάζεται διπλή γραμμή από τον artist / band προς το relationship set του καθενός καθώς χρειάζονται τουλάχιστον ένα είδος μουσικής.

  Σχετικά με το genre, δημιουργήσαμε τη **σχέση "parent genre"**, η οποία ενώνει ένα genre με κάποιο υποείδος του. Είναι σχέση **one to many**, καθώς ένα είδος μπορεί να έχει ένα ή κανένα είδος "γονέα", ενώ ο "γονέας" να έχει ένα ή και παραπάνω υποείδη.

- Έχοντας τελειώσει με το κομμάτι του performance, ασχοληθήκαμε και υλοποιήσαμε το **entity Ticket.** Το ticket περιέχει ως primary key το ticket_id και τα υπόλοιπα attributes αφορούν χαρακτηριστικά του κάθε εισιτηρίου, όπως

  - purchase_date, price, payment_method, ean_code, is_used, is_for_sale.

  Σχετίζεται άμεσα με το **entity Event** μέσω του relationship set **"gives access to"**, για να εξηγήσει τι σχέση έχουν οι παραστάσεις με τα tickets. Αναλυτικότερα, η σχέση ticket-event είναι **many to one**, καθώς ένα event περιέχει πολλά ticket, ενώ ένα εισιτήριο αφορά αποκλειστικά ένα event. Μάλιστα, χρειάζεται ένα ακριβώς event για να υπάρξει, επομένως έχουμε total participation του ticket.

- Τα performances δέχονται κριτικές δεδομένου ότι αυτός που κάνει τη κριτική έχει ενεργοποιημένο ticket για το event που περιέχεται η παράσταση που τον αφορά. Για αυτό, δημιουργήσαμε το **entity Rating**, το οποίο έχει το δικό του primary key rating_id, και άλλα attributes που χαρακτηρίζουν την επίδοση των επιμέρων χαρακτηριστικών της παράστασης, όπως τα:

  - artist_band_performance, sound_lighting, stage_presence, organization, overall_impression

- Το Rating σχετίζεται με το Ticket και το Performance μέσω **many to one** σχέσεων με ονόματα **"has rated as**" και **"belongs to"** αντίστοιχα. Οι σχέσεις είναι έτσι καθώς ένα rating αφορά αποκλειστικά ένα performance και ένα ticket, ενώ ένα ticket μπορεί να περιέχει πολλά ratings (σε ένα event έχουμε πολλές παραστάσεις) και ένα performance πολλά rating ανάλογα με τους ποιους από αυτούς που παρευρέθηκαν αποφάσισαν να κάνουν κριτική. Μάλιστα, το κάθε rating πρέπει να έχει ένα σχετικό εισιτήριο και αντίστοιχα σχετική παράσταση αλλιώς δεν μπορεί να υπάρξει, άρα χρειάζονται και **διπλές γραμμές.**

- Επιπρόσθετα, δημιουργήσαμε την **οντότητα Visitor**, η οποία έχει primary key το visitor_id, και τα υπόλοιπα attributes αποτελούν το ονοματεπώνυμό του, το email του, το κινητό του, και την ημερομηνία γέννησής του. Σχετίζεται με το ticket μέσω του **relationship set "Has"** και η σχέση τους είναι one to many, καθώς ένα ticket αφορά έναν αποκλειστικά επισκέπτη και ένας επισκέπτης μπορεί να έχει πολλά εισιτήρια. Μάλιστα για να υπάρχει το εισιτήριο πρέπει να βάλουμε και την διπλή γραμμή (**total participation**), καθώς χρειάζεται απαραίτητα ένα visitor.

- Το ticket ακόμη χαρακτηρίζεται από τρεις κατηγορίες εισόδου, για τις οποίες φτιάξαμε την οντότητα **ticketcategory.** Αυτή αποτελείται από το primary key category_id και το attribute category_name, το οποίο θα χαρακτηρίζεται από τα : γενική είσοδος, VIP, backstage χωρίς όμως τη χρήση enums, όπως θα δούμε στη συνέχεια. Το ticketcategory σχετίζεται με το ticket μέσω του "**belongs to"** relationship set και η σχέση τους είναι **one to many:** Το κάθε εισιτήριο έχει μία κατηγορία, ενώ μία κατηγορία έχει πολλά εισιτήρια. Απαραίτητη είναι μάλιστα η **διπλή γραμμή** στο ticket, καθώς είναι υποχρεωτικό για ένα εισιτήριο να έχει κατηγορία.

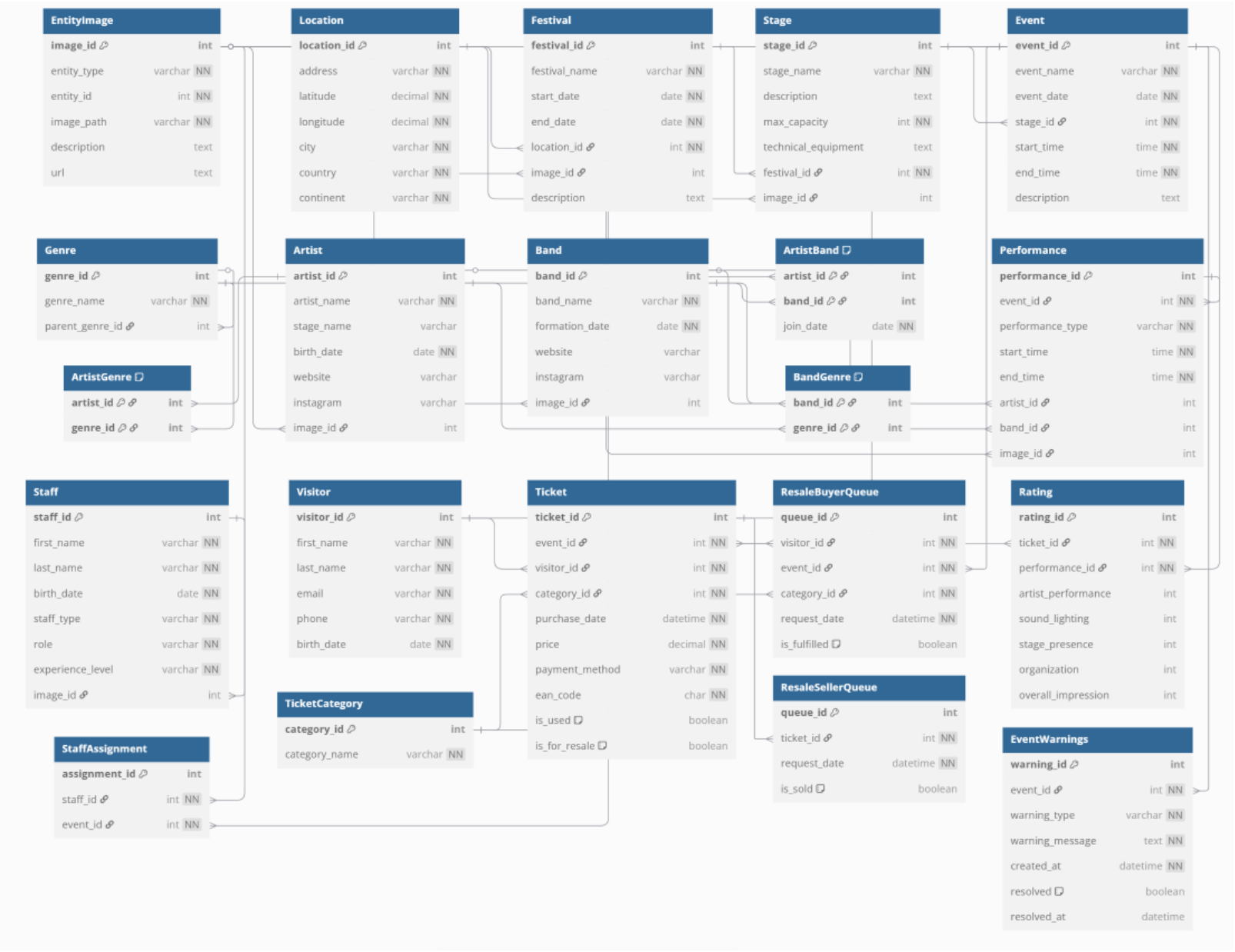
- Για την γραμμή πώλησης εισιτηρίων όταν μια παράσταση του φεστιβάλ εξαντλεί τα εισιτήρια της υλοποιήσαμε τα entities **resalebuyerqueue** και **resalesellerqueue** για το διαχωρισμό των θέσεων των αγοραστών και πωλητών αντίστοιχα σε ένα σύστημα FIFO.

  Συγκεκριμένα, το entity **Resalesellerqueue** αποτελείται από ένα primary key queue_id, και ακόμη από άλλα attributes request_date και is_sold, τα οποία αφορούν πληροφορίες για τις οποίες θα εξαρτηθεί η ουρά προτεραιότητας όταν υπάρξει ανάγκη. Σχετίζεται με το ticket μέσω του **relationship set "is available to sell"** σε μία σχέση **one to one,** καθώς ένα ticket μπορεί να μπει σε μία μονάχα θέση στην ουρά πώλησης, ενώ η θέση σε μια ουρά πώλησης αφορά ένα ticket.

- Τέλος, το entity **Resalebuyerqueue** αποτελεί τη θέση στην ουρά πώλησης. Αποτελείται από το primary key **queue_id** και τα attributes request_date και is fulfilled, έτσι ώστε να υπάρχει τάξη στην ουρά αγοράς και η θέση με index queue_id να αποχωρεί όταν δοθεί εισιτήριο.
  Το entity αυτό σχετίζεται άμεσα με:
    - Το **ticketcategory** μέσω του relationship set **"requests"** , σε μία **many to one σχέση**, καθώς ένας στην ουρά αγοράς ζητάει εισιτήριο μοναδικής κατηγορίας, ενώ μία κατηγορία εισιτηρίου αφορά πολλούς στην ουρά. Επιπλέον, η θέση στην ουρά χρειάζεται απαραίτητα κατηγορία εισιτηρίου, άρα χρειάζεται να μπει διπλή γραμμή.
    - Το **visitor** μέσω του relationship set **"can buy from here"** , σε μία **many to one σχέση**, καθώς ένας visitor έχει καμία έως πολλές θέσεις στην ουρά, ενώ μία θέση στην ουρά αφορά έναν visitor.
    - Το **event** μέσω του relationship set **"gives access to"** , σε μία **many to one σχέση**, καθώς ένα event αφορά καμία μέχρι και πολλές θέσεις στην ουρά, ενώ μία θέση στην ουρά αφορά ένα event.

2. Υλοποίηση Βάσης

α. Σχεσιακό Διάγραμμα



β. Ανάπτυξη Βάσης – Επεξήγηση DDL script

```sql
-- Database creation
DROP DATABASE IF EXISTS pulse_university;
CREATE DATABASE pulse_university;
USE pulse_university;

-- Images for various entities table
CREATE TABLE EntityImage (
    image_id INT AUTO_INCREMENT PRIMARY KEY,
    entity_type VARCHAR(50) NOT NULL,
    entity_id INT NOT NULL,
    image_path VARCHAR(255) NOT NULL,
    description TEXT,
    url TEXT,
    INDEX (entity_type, entity_id)
);

-- Location table
CREATE TABLE Location (
    location_id INT AUTO_INCREMENT PRIMARY KEY,
    address VARCHAR(255) NOT NULL,
    latitude DECIMAL(10, 8) NOT NULL,
    longitude DECIMAL(11, 8) NOT NULL,
    city VARCHAR(100) NOT NULL,
    country VARCHAR(100) NOT NULL,
    continent VARCHAR(50) NOT NULL,
    CHECK (latitude BETWEEN -90 AND 90),
    CHECK (longitude BETWEEN -180 AND 180)
);

-- Festival table
CREATE TABLE Festival (
    festival_id INT AUTO_INCREMENT PRIMARY KEY,
    festival_name VARCHAR(100) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    location_id INT NOT NULL,
    image_id INT NULL,
    description TEXT,
    FOREIGN KEY (image_id) REFERENCES EntityImage(image_id) ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (location_id) REFERENCES Location(location_id) ON DELETE CASCADE ON UPDATE CASCADE,
    CHECK (end_date >= start_date)
);
```

```sql
-- Stage/Venue table
CREATE TABLE Stage (
    stage_id INT AUTO_INCREMENT PRIMARY KEY,
    stage_name VARCHAR(100) NOT NULL,
    description TEXT,
    max_capacity INT NOT NULL,
    technical_equipment TEXT,
    festival_id INT NOT NULL,
    image_id INT NULL,
    FOREIGN KEY (image_id) REFERENCES EntityImage(image_id) ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (festival_id) REFERENCES Festival(festival_id) ON DELETE CASCADE ON UPDATE CASCADE,
    CHECK (max_capacity > 0)
);

-- Event table
CREATE TABLE Event (
    event_id INT AUTO_INCREMENT PRIMARY KEY,
    event_name VARCHAR(255) NOT NULL,
    event_date DATE NOT NULL,
    stage_id INT NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    description TEXT,
    FOREIGN KEY (stage_id) REFERENCES Stage(stage_id) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (stage_id, event_date, start_time, end_time)
);

-- Musical Genre table
CREATE TABLE Genre (
    genre_id INT AUTO_INCREMENT PRIMARY KEY,
    genre_name VARCHAR(100) NOT NULL,
    parent_genre_id INT NULL,
    FOREIGN KEY (parent_genre_id) REFERENCES Genre(genre_id) ON DELETE SET NULL ON UPDATE CASCADE,
    UNIQUE (genre_name)
);

-- Artist table
CREATE TABLE Artist (
    artist_id INT AUTO_INCREMENT PRIMARY KEY,
    artist_name VARCHAR(255) NOT NULL,
    stage_name VARCHAR(255) NULL,
    birth_date DATE NOT NULL,
    website VARCHAR(255) NULL,
    instagram VARCHAR(255) NULL,
    image_id INT NULL,
    FOREIGN KEY (image_id) REFERENCES EntityImage(image_id) ON DELETE SET NULL ON UPDATE CASCADE,
    UNIQUE (artist_name, birth_date)
);
```

```sql
-- Band table
CREATE TABLE Band (
    band_id INT AUTO_INCREMENT PRIMARY KEY,
    band_name VARCHAR(255) NOT NULL,
    formation_date DATE NOT NULL,
    website VARCHAR(255) NULL,
    instagram VARCHAR(255) NULL,
    image_id INT NULL,
    FOREIGN KEY (image_id) REFERENCES EntityImage(image_id) ON DELETE SET NULL ON UPDATE CASCADE,
    UNIQUE (band_name)
);

-- Artist-Band relationship (Many-to-Many)
CREATE TABLE ArtistBand (
    artist_id INT NOT NULL,
    band_id INT NOT NULL,
    join_date DATE NOT NULL,
    PRIMARY KEY (artist_id, band_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (band_id) REFERENCES Band(band_id) ON DELETE CASCADE ON UPDATE CASCADE
);

-- Artist-Genre relationship (Many-to-Many)
CREATE TABLE ArtistGenre (
    artist_id INT NOT NULL,
    genre_id INT NOT NULL,
    PRIMARY KEY (artist_id, genre_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id) ON DELETE CASCADE ON UPDATE CASCADE
);

-- Band-Genre relationship (Many-to-Many)
CREATE TABLE BandGenre (
    band_id INT NOT NULL,
    genre_id INT NOT NULL,
    PRIMARY KEY (band_id, genre_id),
    FOREIGN KEY (band_id) REFERENCES Band(band_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id) ON DELETE CASCADE ON UPDATE CASCADE
);
 -- Performance table
 CREATE TABLE Performance (
     performance_id INT AUTO_INCREMENT PRIMARY KEY,
     event_id INT NOT NULL,
     performance_type VARCHAR(50) NOT NULL CHECK (performance_type IN ('warm up', 'headline', 'special guest', 'regular')),
     start_time TIME NOT NULL,
     end_time TIME NOT NULL,
     artist_id INT NULL,
     band_id INT NULL,
     image_id INT NULL,
     FOREIGN KEY (event_id) REFERENCES Event(event_id) ON DELETE CASCADE ON UPDATE CASCADE,
     FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE SET NULL ON UPDATE CASCADE,
     FOREIGN KEY (band_id) REFERENCES Band(band_id) ON DELETE SET NULL ON UPDATE CASCADE,
     FOREIGN KEY (image_id) REFERENCES EntityImage(image_id) ON DELETE SET NULL ON UPDATE CASCADE,
     CHECK ((artist_id IS NULL AND band_id IS NOT NULL) OR (artist_id IS NOT NULL AND band_id IS NULL)),
     CHECK (TIMEDIFF(end_time, start_time) <= '03:00:00')
 );
```

```sql
-- Staff table
CREATE TABLE Staff (
    staff_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    birth_date DATE NOT NULL,
    staff_type VARCHAR(100) NOT NULL,
    role VARCHAR(100) NOT NULL,
    experience_level VARCHAR(100) NOT NULL,
    image_id INT NULL,
    FOREIGN KEY (image_id) REFERENCES EntityImage(image_id) ON DELETE SET NULL ON UPDATE CASCADE
);

-- Staff Assignment table
CREATE TABLE StaffAssignment (
    assignment_id INT AUTO_INCREMENT PRIMARY KEY,
    staff_id INT NOT NULL,
    event_id INT NOT NULL,
    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (event_id) REFERENCES Event(event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (staff_id, event_id)
);

-- Visitor table
CREATE TABLE Visitor (
    visitor_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    birth_date DATE NOT NULL,
    UNIQUE (email)
);

-- Ticket Category table
CREATE TABLE TicketCategory (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(50) NOT NULL CHECK (category_name IN ('general', 'vip', 'backstage')),
    UNIQUE (category_name)
);
-- Ticket table
CREATE TABLE Ticket (
    ticket_id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT NOT NULL,
    visitor_id INT NOT NULL,
    category_id INT NOT NULL,
    purchase_date DATETIME NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    payment_method VARCHAR(100) NOT NULL,
    ean_code CHAR(13) NOT NULL,
    is_used BOOLEAN DEFAULT FALSE,
    is_for_resale BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (event_id) REFERENCES Event(event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (visitor_id) REFERENCES Visitor(visitor_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (category_id) REFERENCES TicketCategory(category_id) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (ean_code),
    CHECK (price > 0)
);
```

```sql
-- Resale Queue (for buyers)
CREATE TABLE ResaleBuyerQueue (
    queue_id INT AUTO_INCREMENT PRIMARY KEY,
    visitor_id INT NOT NULL,
    event_id INT NOT NULL,
    category_id INT NOT NULL,
    request_date DATETIME NOT NULL,
    is_fulfilled BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (visitor_id) REFERENCES Visitor(visitor_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (event_id) REFERENCES Event(event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (category_id) REFERENCES TicketCategory(category_id) ON DELETE CASCADE ON UPDATE CASCADE
);

-- Resale Queue (for sellers)
CREATE TABLE ResaleSellerQueue (
    queue_id INT AUTO_INCREMENT PRIMARY KEY,
    ticket_id INT NOT NULL,
    request_date DATETIME NOT NULL,
    is_sold BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (ticket_id)
);

-- Rating table
CREATE TABLE Rating (
    rating_id INT AUTO_INCREMENT PRIMARY KEY,
    ticket_id INT NOT NULL,
    performance_id INT NOT NULL,
    artist_performance INT CHECK (artist_performance BETWEEN 1 AND 5),
    sound_lighting INT CHECK (sound_lighting BETWEEN 1 AND 5),
    stage_presence INT CHECK (stage_presence BETWEEN 1 AND 5),
    organization INT CHECK (organization BETWEEN 1 AND 5),
    overall_impression INT CHECK (overall_impression BETWEEN 1 AND 5),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (performance_id) REFERENCES Performance(performance_id) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (ticket_id, performance_id)
);

CREATE TABLE EventWarnings (
    warning_id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT NOT NULL,
    warning_type VARCHAR(100) NOT NULL,
    warning_message TEXT NOT NULL,
    created_at DATETIME NOT NULL,
    resolved BOOLEAN DEFAULT FALSE,
    resolved_at DATETIME,
    FOREIGN KEY (event_id) REFERENCES Event(event_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

i. Πίνακες

Στο μοντέλο δεδομένων του Pulse University, η πληροφορία δομείται γύρω από ένα σύνολο οντοτήτων, οι οποίες διαθέτουν τα απαραίτητα primary keys για τον μοναδικό προσδιορισμό των αντικειμένων τους, καθώς και foreign keys για την αποδοτική σύνδεση μεταξύ οντοτήτων:

1. **EntityImage**: Η EntityImage αποτελεί μια οντότητα συλλογής εικόνων, για την αποθήκευση εικόνων με λεκτική περιγραφή των καλλιτεχνών, των φεστιβάλ, του εξοπλισμού, κτλ. Έχει το primary key image_id, το οποίο ορίζουμε να είναι AUTO_INCREMENT για ευκολότερη ταυτοποίηση κάθε εικόνας, και διαθέτει ως attributes τον τύπο της οντότητας που χαρακτηρίζει η εικόνα καθώς και το primary key της. Αυτά τα attributes, όπως και το image_path, που εκφράζει τη διαδρομή του αρχείου, είναι υποχρεωτικό να υφίστανται (NOT NULL) ώστε να υπάρχει το αντικείμενο της οντότητας EntityImage. Τέλος, έχει περιγραφή description και σύνδεσμο url και έχουμε ορίσει index (entity_type, entity_id) για τη διευκόλυνση πιθανών queries. Αρκετές από τις οντότητες της βάσης μας έχουν ως foreign key το image_id ώστε να αντιστοιχίζονται σε αντικείμενα αυτών των οντοτήτων.

2. **Location**: Η οντότητα Location αποθηκεύει γεωγραφικά δεδομένα κάθε τόπου διεξαγωγής φεστιβάλ. Έχει ως primary key το location_id με αυτόματη αύξηση για ευκολία, και attributes που αφορούν την ακριβή τοποθεσία του φεστιβάλ και είναι απαιτούμενα για το αντικείμενο, όπως ήπειρο, χώρα, πόλη, διεύθυνση, γεωχωρικές συντεταγμένες (latitude, longitude), με τον κατάλληλο έλεγχο ώστε οι συντεταγμένες να είναι έγκυρες.

3. **Festival**: Η Festival αναπαριστά κάθε ετήσιο φεστιβάλ με το festival_id ως primary key (AUTO_INCREMENT) και ορισμένα attributes που δίνουν σημαντικές πληροφορίες για αυτό, όπως festival_name και start-end dates. Αξιοσημείωτα σε αυτή την οντότητα είναι τα foreign keys location_id, αφού κάθε φεστιβάλ πρέπει να αντιστοιχίζεται σε τοποθεσία, και image_id. Επίσης προφανώς πρέπει να έχουμε τον περιορισμό end_date >= start_date.

4. **Stage**: Η Stage μοντελοποιεί κάθε κτίριο/μουσική σκηνή όπου διεξάγονται παραστάσεις, με μοναδική ταυτοποίηση μέσω του primary key stage_id, το οποίο αυηξάνουμε κατά την εισαγωγή αντικειμένων στη βάση. Διαθέτει attributes, απαραίτητα για τον προσδιορισμό του stage, όπως stage_name, max_capacity (έλεγχος

αν max_capacity > 0), technical_equipment, και foreign key festival_id αφού πρέπει να αντιστοιχίζεται σε φεστιβάλ.

5. **Event**: Η Event οντότητα συμβολίζει κάθε παράσταση που πραγματοποιείται στα φεστιβάλ, με primary key το event_id (AUTO_INCREMENT) και, ομοίως με παραπάνω, τα απαιτούμενα attributes για τις πληροφορίες της, καθώς και foreign key stage_id. Αξίζει να τονίσουμε τον περιορισμό UNIQUE (stage_id, event_date, start_time, end_time), διότι κάθε σκηνή μπορεί να φιλοξενεί μόνο μία παράσταση την ίδια στιγμή.

6. **Genre**: Η οντότητα Genre περιγράφει τα μουσικά είδη και υποείδη, παρέχοντας ιεραρχική κατηγοριοποίηση. Έχει ως primary key το genre_id με αυτόματη αύξηση και attributes genre_name (το επιλέξαμε ως μοναδικό, ώστε να μην υπάρχουν διπλές καταχωρίσεις καθώς δεν γίνεται δύο είδη να έχουν το ίδιο όνομα) και parent_genre_id, το οποίο μας διευκολύνει για την ιεραρχική δομή που προαναφέραμε αφού λειτουργεί ως foreign key για τα μουσικά υποείδη που πρέπει να αντιστοιχίζονται σε ένα είδος.

7. **Artist**: Αντιπροσωπεύει καλλιτέχνες, διαθέτοντας ως attributes τα βασικά τους στοιχεία (artist_name, birth date, stage_name, website, Instagram). Έχει ως primary key το artist_id με AUTO_INCREMENT, foreign key σε EntityImage ώστε να αντιστοιχίζονται οι φωτογραφίες σε καλλιτέχνες, και ορίσαμε τον συνδυασμό (artist_name, birth_date) μοναδικό την πιο εύκολη αποφυγή διπλοεγγραφών.

8. **Band**: Ομοίως με την οντότητα Artist, η Band έχει το μοναδικό band_id το οποίο αυξάνουμε σε κάθε εισαγωγή και παρόμοια attributes, με επιπλέον στοιχεία το band_name, το οποίο θεωρούμε μοναδικό για κάθε μπάντα, και το formation_date.

Για τις οντότητες Artist και Band δημιουργήσαμε τρεις βοηθητικές οντότητες, την ArtistBand, την ArtistGenre και την BandGenre, οι οποίες μας διευκολύνουν για τη σχέση που υπάρχει μεταξύ των καλλιτεχνών και των συγκροτημάτων και τη σύνδεσή τους με τα μουσικά είδη:

9. **ArtistBand**: Περιγράφει τη συμμετοχή καλλιτεχνών σε συγκροτήματα και την ημερομηνία εισόδου, έχοντας ως σύνθετο primary key το ζευγάρι (artist_id, band_id), που είναι μοναδικό και περιγράφει επαρκώς τη λειτουργία αυτής της οντότητας, και ως foreign keys τα artist_id, band_id.

10. **ArtistGenre**: Αυτή η οντότητα συσχετίζει τους καλλιτέχνες με τα μουσικά είδη με τα οποία ασχολούνται, ώστε κάθε καλλιτέχνης να μπορεί να έχει παραπάνω από ένα. Έχει σύνθετο primary key (artist_id, genre_id) για τον προσδιορισμό των ειδών κάθε καλλιτέχνη, και τα αντίστοιχα foreign keys.

11. **BandGenre**: Ομοίως με την παραπάνω οντότητα, εκφράζει τα μουσικά είδη κάθε συγκροτήματος, με primary key (band_id, genre_id).

12. **Performance**: Η οντότητα Performance αντιπροσωπεύει μία μεμονωμένη εμφάνιση ενός καλλιτέχνη ή συγκροτήματος στο πλαίσιο ενός event, σε συγκεκριμένη σκηνή και χρονικό διάστημα. Διαθέτει ως primary key το performance_id με αυτόματη αύξηση, μοναδικό για κάθε performance, και τα απαραίτητα attributes (ορισμένα από αυτά foreign keys, συγκεκριμένα τα event_id, artist_id, band_id, image_id), τα οποία διευκρινίζουν τον χρόνο και τον καλλιτέχνη/συγκρότημα που ερμηνεύει. Παράλληλα, με τους ελέγχους για το performance type ώστε να είναι κάποιο από τα ('warm up', 'headline', 'special guest', 'regular'), για την εμφάνιση είτε μεμονωμένου καλλιτέχνη είτε συγκροτήματος, και για τη διάρκεια του performance, εισάγουμε δεδομένα για αυτή την οντότητα στη βάση μας με ασφάλεια.

13. **Staff**: Η Staff αναπαριστά, με primary key το staff_id, το προσωπικό που εργάζεται σε διάφορους ρόλους κατά τη διάρκεια των φεστιβάλ, περιέχοντας σημαντικές προσωπικές πληροφορίες όπως το όνομα, ημερομηνία γέννησης (first_name, last_name, birth_date), καθώς και πληροφορίες για την εμπειρία τους και τον ρόλο τους στο φεστιβάλ (experience_level, staff_type, role).

14. **StaffAssignment**: Η οντότητα StaffAssignment αποτελεί βοηθητική οντότητα που αναθέτει το προσωπικό στις παραστάσεις. Έχει primary key με AUTO_INCREMENT το assignment_id και ως foreign keys τα staff_id, event_id για την ανάθεση του διαθέσιμου προσωπικού στις παραστάσεις. Για λόγους ευκολίας εισαγωγής δεδομένων απαιτούμε το ζευγάρι (staff_id, event_id) να είναι μοναδικό για να μην προκύψει κάποια διπλοεγγραφή.

15. **Visitor**: Η Visitor εκφράζει τους επισκέπτες του φεστιβάλ, οι οποίοι αγοράζουν εισιτήρια και έχουν τη δυνατότητα παρακολούθησης πολλαπλών εμφανίσεων. Έχει ως

primary key το visitor_id με αυτόματη αύξηση κατά την εισαγωγή των δεδομένων και περιλαμβάνει προσωπικά στοιχεία κάθε ατόμου (first_name, last_name, email, phone, birth_date) με την απαίτηση το email να είναι μοναδικό, καθώς δεν γίνεται να είναι ίδιες δύο διευθύνσεις email.

16. **TicketCategory**: Η οντότητα TicketCategory καθορίζει τις διακριτές κατηγορίες εισιτηρίων που διατίθενται για κάθε παράσταση, παρέχοντας ευκολία στην κατηγοριοποίηση των εισιτηρίων (χρήσιμο και για την ουρά μεταπώλησης), με primary key category_id και μοναδικό όνομα ένα εκ των οποίων ('general', 'vip', 'backstage').

17. **Ticket**: Αυτή η οντότητα αναπαριστά μοναδικά εισιτήρια που μπορεί να αγοραστεί ή μεταπωληθεί από επισκέπτη, με primary key το ticket_id αλλά και μοναδικό αναγνωριστικό τον κώδικα EAN-13 (ean_code). Διαθέτει τις απαραίτητες πληροφορίες για την παράσταση , τον κάτοχο και την κατηγορία, για αυτό έχει τα αντίστοιχα foreign keys (event_id, visitor_id, category_id). Επιπροσθέτως, εξίσου σημαντικές είναι και οι πληροφορίες για την αγορά (price με έλεγχο ώστε να είναι θετικός αριθμός, payment_method) και για την κατάσταση του εισιτηρίου (is_used, is_for_resale).

Για λόγους ευκολίας υλοποίησης της ουράς μεταπώλησης, αφού δεν διαθέτουμε λίστες ή ουρές, τη χωρίσαμε σε ResaleBuyerQueue και ResaleSellerQueue.

18. **ResaleBuyerQueue**: Εκφράζει τους επισκέπτες που έχουν δηλώσει ενδιαφέρον να αγοράσουν εισιτήριο από μεταπώληση συγκεκριμένης κατηγορίας, με primary_key το queue_id (AUTO_INCREMENT), που δηλώνει τη θέση του επισκέπτη στην ουρά μεταπώλησης και foreign keys αναγκαία για την αγορά, όπως visitor_id, event_id, category_id. Διαθέτει και κάποιες τεχνικές λεπτομέρειες, συγκεκριμένα το request_date και το is_fulfilled για την κατάσταση του εισιτηρίου.

19. **ResaleSellerQueue**: Η οντότητα καταγράφει τα εισιτήρια που έχουν προσφερθεί για μεταπώληση από τους ιδιοκτήτες τους, έχοντας primary_key το queue_id, που δηλώνει τη θέση του εισιτηρίου στην ουρά μεταπώλησης αλλά και μοναδικό το foreign key ticket_id, αφού δεν μπορεί κάποιο εισιτήριο να υπάρξει 2 φορές στην ουρά μεταπώλησης. Επίσης, διαθέτει και τεχνικές λεπτομέρειες, συγκεκριμένα το request_date και το is_sold για την κατάσταση του εισιτηρίου.

20. **Rating**: Η οντότητα Rating εκφράζει μία αξιολόγηση που δίνει ένας επισκέπτης για την ερμηνεία ενός καλλιτέχνη ή συγκροτήματος αλλά και γενικότερα για την οργάνωση και τα εφέ της εμφάνισης, έχοντας ως primary key το rating_id, με αυτόματη αύξηση κατά την εισαγωγή δεδομένων. Συγκεκριμένα, βαθμολογεί με βάση την κλίμακα Likert, από 1 έως 5 την ερμηνεία του καλλιτέχνη (artist_performance), τον ήχο και τον φωτισμό (sound_lighting), τη σκηνική παρουσία (stage_presence), την οργάνωση (organization) και τη συνολική εντύπωση (overall_impression). Για να είναι έγκυρη η αξιολόγηση, πρέπει το ζευγάρι (ticket_id, performance_id) να είναι μοναδικό, ώστε να αντιστοιχεί 1 αξιολόγηση ανά εισιτήριο για κάθε εμφάνιση.

21. **EventWarnings**: Αντιπροσωπεύει ειδοποιήσεις και προβλήματα που σχετίζονται με μια παράσταση. Έχει primary key το warning_id (AUTO_INCREMENT), σχετίζεται με την παράσταση μέσω του foreign key event_id και διαθέτει τις απαραίτητες πληροφορίες για το είδος αυτών των ειδοποιήσεων (warning_type, warning_message), τη χρονική τους διάσταση (created_at) και την κατάσταση στην οποία βρίσκονται, δηλαδή αν έχουν επιλυθεί (resolved, resolved_at).

Σε όλους τους πίνακες που περιέχουν foreign keys, έχουμε βάλει περιορισμό είτε ON DELETE CASCADE ON UPDATE CASCADE είτε ON DELETE SET NULL ON UPDATE CASCADE, ώστε να είναι η βάση μας ασφαλής σε περίπτωση διαγραφών και να γίνεται ενημέρωση των child tables σε περίπτωση ενημέρωσης των parent tables.

ii. Περιορισμοί

Τα constraints είναι χρήσιμα στην δημιουργία μιας βάσης δεδομένων, καθώς θέτουν κάποια όρια στα σύνολα τιμών των attributes, τέτοια ώστε να αντικατοπτρίζουν την λογική την οποία θέλουμε να εξυπηρετεί η βάση. Αυτοί οι περιορισμοί χωρίζονται στις ακόλουθες κατηγορίες:

Κλειδιά

PRIMARY KEY: Με αυτόν τον όρο αποκαλούμε την ή τις στήλες οι οποίες απαιτούνται, για να εξασφαλιστεί μοναδικότητα των στοιχείων ενός πίνακα. Στις περισσότερες οντότητες υπάρχει ξεχωριστό attribute το οποίο αναλαμβάνει αυτό τον ρόλο, καθώς η δυνατότητα να μπορούμε να διαχωρίσουμε το κάθε στοιχείο του πίνακα είναι καθοριστικής σημασίας.

FOREIGN KEY: Αυτός ο περιορισμός θέτει σε μια στήλη του πίνακα στοιχεία του Primary Key ενός άλλου πίνακα. Η αξία ενός τέτοιου εργαλείου αναδεικνύεται πρακτικά στην χρήση procedures, triggers αλλά και queries, αφού μας δίνει την δυνατότητα να συσχετίσουμε ένα στοιχείο του αρχικού πίνακα με κάποιο ενός άλλου, με μοναδικό τρόπο.

UNIQUE KEY: Ο περιορισμός αυτός εξασφαλίζει πως τα στοιχεία μιας στήλης θα είναι μοναδικά, δεν θα μπορούν να υπάρχουν διπλότυπα. Μερικά πεδία όπως τα band_name του Band, email του Visitor ή το ean του Ticket πρέπει να είναι μοναδικά εκ φύσεως, οπότε, με τη βοήθεια του constraint αυτού, μπορούμε να το εγγυηθούμε αυτό.

Περιορισμοί Αναφορικής Ακεραιότητας

ON DELETE CASCADE: Αυτή η έκφραση σημαίνει πως, όταν συμβεί κάποια διαγραφή ενός στοιχείου στον πίνακα στον οποίο ανήκει αυτή η εντολή, διαγράφονται αυτόματα και όλα τα σχετικά στοιχεία στους θυγατρικούς πίνακες που εξαρτώνται από το γονικό μέσω foreign key. Αυτό συμβαίνει ώστε να διατηρείται η ακεραιότητα των δεδομένων σε όλη την έκταση της βάσης δεδομένων.

ON UPDATE CASCADE: Όπως και προηγουμένως, για τον ίδιο λόγο, ενημερώνονται όλες οι σχετικές εγγραφές σε πίνακες συνδεδεμένους με foreign keys με κάποιον γονικό, όταν πραγματοποιείται κάποια ενημέρωση σε αυτόν.

ON DELETE SET NULL: Η λειτουργία αυτή μοιάζει με το ON DELETE CASCADE, μόνο που σε αυτή τη περίπτωση δεν θέλουμε να ξεφορτωθούμε εντελώς τις εμφανίσεις του στοιχείου το οποίο διαγράφτηκε, από τους θυγατρικούς πίνακες, αρκούμαστε απλά στην ενημέρωση τους με την τιμή NULL. Αυτό συμβαίνει όταν το στοιχείο συνεχίζει να έχει νόημα και χωρίς τον "γονέα".

Περιορισμοί Ακεραιότητας Πεδίου Τιμών

NOT NULL: Με αυτόν τον περιορισμό εξασφαλίζουμε ότι κάποιο attribute δεν θα μπορεί να έχει κενές τιμές. Εμφανίζεται εκτενώς στην βάση δεδομένων μας, καθώς για πολλά από τα πεδία είναι απαραίτητη αυτή η συνθήκη ώστε να μην υπάρχουν κενά πληροφορίας.

DEFAULT: Μπορούμε να θέσουμε μια αρχική τιμή για τις τιμές ενός attribute, μέσω της λειτουργίας του default. Κάτι τέτοιο είναι ιδιαίτερα βοηθητικό σε Boolean τιμές, καθώς συχνά χρειαζόμαστε να βρίσκονται στην False κατάσταση και ύστερα από κάποια ενέργεια να ενεργοποιούνται και γίνονται True. Χαρακτηριστικό παράδειγμα αποτελεί η κατάσταση στην οποία βρίσκεται ένα εισιτήριο πριν και αφού χρησιμοποιηθεί, καθώς την στιγμή που θα αγοραστεί και θα προστεθεί στην βάση δεδομένων, σίγουρα δεν θα είναι χρησιμοποιημένο, οπότε αρχικοποιείται σε False.

CHECK: Υπάρχει και η δυνατότητα να περιορίσουμε τις τιμές μιας στήλης αναγκάζοντάς τες να τηρούν μια συνθήκη, δίχως την οποία θα χανόταν η λογική συνέπεια των δεδομένων. Παράδειγμα αυτού αποτελεί η μέγιστη χωρητικότητα μιας σκηνής, η οποία θα πρέπει να είναι σίγουρα κάποιος ακέραιος μεγαλύτερος του μηδενός.

Περιορισμοί Οριζόμενοι από τον Χρήστη

CHECK (Σε Επίπεδο Πίνακα): Είναι ίδιος με τον απλό περιορισμό CHECK στην ουσία του, με την διαφορά πως εδώ, η συνθήκη που επιβάλουμε στον πίνακα, συμπεριλαμβάνει την σύγκριση δύο διαφορετικών attributes του πίνακα. Επί παραδείγματι, η ημερομηνία εκκίνησης start_date του Festival πρέπει να είναι μικρότερη από την ημερομηνία λήξης end_date.

Triggers: Αποτελούν έναν τρόπο περιορισμού ο οποίος ενεργοποιείται αυτόματα μετά από κάποια ορισμένη ενέργεια η οποία πυροδοτεί τον έλεγχο και την επιβολή του περιορισμού, και συνήθως είναι αρκετά πιο σύνθετος για να εκφραστεί με μια απλή σύγκριση. Εκτενέστερη ανάλυση των triggers παρουσιάζεται στην αντίστοιχη ενότητα.

iii. Ευρετήρια – Indexes

```sql
-- Create indexes for the most frequently used queries

-- For festival revenue reports
CREATE INDEX idx_ticket_event_category ON Ticket (event_id, category_id);
CREATE INDEX idx_ticket_payment ON Ticket (payment_method);
CREATE INDEX idx_event_festival ON Event (stage_id);
CREATE INDEX idx_stage_festival ON Stage (festival_id);

-- For artist performance searches
CREATE INDEX idx_performance_artist ON Performance (artist_id);
CREATE INDEX idx_performance_band ON Performance (band_id);
CREATE INDEX idx_performance_type ON Performance (performance_type);
CREATE INDEX idx_performance_event ON Performance (event_id);

-- For visitor-related queries
CREATE INDEX idx_ticket_visitor ON Ticket (visitor_id);
CREATE INDEX idx_rating_ticket ON Rating (ticket_id);

-- For location-based queries
CREATE INDEX idx_festival_location ON Festival (location_id);
CREATE INDEX idx_location_continent ON Location (continent);

-- For staff queries
CREATE INDEX idx_staff_type ON Staff (staff_type);
CREATE INDEX idx_staff_experience ON Staff (experience_level);

-- For date-based queries
CREATE INDEX idx_festival_dates ON Festival (start_date, end_date);
CREATE INDEX idx_event_date ON Event (event_date);

-- Add this index to optimize Query 5 (young artists)
CREATE INDEX idx_artist_birth_date ON Artist (birth_date);
CREATE INDEX idx_rating_performance ON Rating (performance_id);
```

Για την αναφορά σε ένα στοιχείο της βάσης δεδομένων, πρέπει να προσπελαστούν πλήθος πινάκων, με χρήση κατάλληλων πράξεων ένωσης αυτών, κάτι που μπορεί να γίνει αρκετά κοστοβόρο σε μεγάλης κλίμακας βάσεις. Για τον λόγο αυτό ενδείκνυται η χρήση δεικτών, οι οποίοι μπορούν να επιταχύνουν σημαντικά την αναζήτηση της κατάλληλης γραμμής, αρκεί να τοποθετηθούν σε κατάλληλες στήλες. Πιο συγκεκριμένα, όλα τα primary keys έχουν index, καθώς και οι στήλες που έχουν δεχτεί foreign key constraint, αφού αυτές είναι σίγουρο ότι θα χρησιμοποιηθούν περισσότερο από όλες. Επιπλέον, έχουν προστεθεί κι άλλοι δείκτες με κριτήριο τις πιο πολυχρησιμοποιούμενες στήλες, όπως είναι αυτές οι οποίες εμφανίζονται στα queries, προσέχοντας ταυτόχρονα να μην δημιουργήσουμε υπερβολικά πολλούς. Κάτι τέτοιο θα μπορούσε να επηρεάσει αρνητικά την βάση, πρώτον λόγω του αυξημένου μεγέθους που θα απαιτεί, και δεύτερον λόγω της παραπάνω επεξεργαστικής δύναμης που θα απαιτείται όταν θα γίνονται προσθήκες ή ενημερώσεις σε στοιχεία της.

iv. Triggers

Τα Triggers αποτελούν έναν τρόπο να αυτοματοποιηθεί η εκτέλεση κάποιας ενέργειας, με έναυσμα κάποια άλλη συγκεκριμένη ενέργεια. Στην βάση μας έχει γίνει χρήση μερικών triggers, τα οποία ελέγχουν την συνέπεια ορισμένων τιμών ύστερα από προσθήκες και αλλαγές σε κρίσιμα πεδία. Γενικά, τα triggers ακολουθούν της εξής λογική, συντακτικά:

Αρχικά, αλλάζουμε το delimeter σε "//". Το delimeter οριοθετεί τα statements της SQL και είναι by default καθορισμένο ως ";". Αλλάζοντάς το προσωρινά σε "//", μπορούμε να διαχειριστούμε τα statements που θα έχει μέσα το trigger μας ως ένα ενιαίο, μεγαλύτερο statement. Η έκφραση FOR EACH ROW συνεπάγεται με εφαρμογή των ακόλουθων ενεργειών σε κάθε γραμμή του πίνακα. Έπειτα, δημιουργούμε boolean συνθήκες που ελέγχουν αν τα δεδομένα μας θα απορριφθούν λόγω μη τήρησης των constraints. Με το trigger μας, πριν κάνουμε insertion στον πίνακα table, ελέγχουμε αν ισχύουν οι συνθήκες αυτές (IF condition THEN …), ώστε να απορρίψουμε το insertion. Αυτό συμβαίνει με την χρήση του SIGNAL SQLSTATE '45000', το οποίο δημιουργεί ένα τεχνητό σφάλμα, με τον κωδικό 45000 να υποδηλώνει σφάλμα ορισμένο από τον χρήστη, και ταυτόχρονα τυπώνεται κάποιο custom μήνυμα σφάλματος. Τέλος, επαναφέρουμε το delimeter σε ";".

- Διασφάλιση διαλείμματος μεταξύ εμφανίσεων (5 – 30 λεπτά)

Πριν από κάθε προσθήκη στον πίνακα Performance κάνουμε τον ακόλουθο έλεγχο. Αρχικά ελέγχουμε αν υπάρχει προηγούμενη εμφάνιση και αποθηκεύουμε την end_time της. Σε περίπτωση που υπάρχει και η καινούρια εμφάνιση που θέλουμε να προσθέσουμε δεν είναι η πρώτη, υπολογίζουμε την διαφορά της ώρας λήξης της προηγούμενης με την start_time της καινούριας και ελέγχουμε αν πληρεί τις προϋποθέσεις. Σε περίπτωση σφάλματος, τυπώνεται αντίστοιχο μήνυμα στην οθόνη. Σε αντίθετη περίπτωση, προχωράμε στην εκτέλεση της προσθήκης καινούριας εμφάνισης.

```
-- Trigger to ensure break times between performances (5-30 min)
DELIMITER //
CREATE TRIGGER check_performance_break_time BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
    DECLARE prev_end TIME;

    SELECT MAX(end_time) INTO prev_end
    FROM Performance
    WHERE event_id = NEW.event_id AND end_time < NEW.start_time;

    IF prev_end IS NOT NULL THEN
        IF TIMEDIFF(NEW.start_time, prev_end) < '00:05:00' OR
           TIMEDIFF(NEW.start_time, prev_end) > '00:30:00' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Break between performances must be between 5 and 30 minutes';
        END IF;
    END IF;
END//
DELIMITER ;
```

- Έλεγχος του αριθμού των VIP εισιτηρίων (10% της χωρητικότητας)

Πριν από προσθήκη VIP εισιτηρίου: Αρχικά αποθηκεύουμε στην προσωρινή μεταβλητή vip_category_id το id των VIP εισιτηρίων. Εάν το εισιτήριο που θέλουμε να προσθέσουμε είναι VIP, βρίσκουμε το stage_id και την χωρητικότητα της σκηνής στην οποία αντιστοιχεί. Στη συνέχεια υπολογίζουμε το πόσα VIP εισιτήρια έχουν ήδη πωληθεί για αυτό το event και αν με τη προσθήκη του καινούριου θα ξεπεραστεί το όριο που έχουμε θέσει.

```
-- Trigger to check maximum VIP tickets (10% of capacity)
DELIMITER //
CREATE TRIGGER check_vip_tickets BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
    DECLARE vip_category_id INT;
    DECLARE stage_capacity INT;
    DECLARE current_vip_count INT;
    DECLARE event_stage_id INT;

    -- Get the VIP category ID
    SELECT category_id INTO vip_category_id FROM TicketCategory WHERE category_name = 'vip';

    IF NEW.category_id = vip_category_id THEN
        -- Get the stage ID for this event
        SELECT s.stage_id, s.max_capacity INTO event_stage_id, stage_capacity
        FROM Event e
        JOIN Stage s ON e.stage_id = s.stage_id
        WHERE e.event_id = NEW.event_id;

        -- Count current VIP tickets
        SELECT COUNT(*) INTO current_vip_count
        FROM Ticket
        WHERE event_id = NEW.event_id AND category_id = vip_category_id;

        -- Check if adding one more VIP ticket would exceed 10%
        IF (current_vip_count + 1) > (stage_capacity * 0.1) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'VIP tickets cannot exceed 10% of stage capacity';
        END IF;
    END IF;
END//
DELIMITER ;
```

- Έλεγχος των εισιτηρίων σε σχέση με την χωρητικότητα της σκηνής

Πριν από προσθήκη εισιτηρίου: Πρώτα βρίσκουμε την χωρητικότητα της σκηνής στην οποία αντιστοιχεί αυτό και την αποθηκεύουμε στην προσωρινή μεταβλητή stage_capacity. Ύστερα υπολογίζουμε τον αριθμό των εισιτηρίων και τον αποθηκεύουμε στο current_ticket_count. Αν με αύξηση αυτού του αριθμού υπερβαίνεται το stage_capacity, τότε εμποδίζεται η προσθήκη εισιτηρίου.

```
-- Trigger to check total tickets vs capacity
DELIMITER //
CREATE TRIGGER check_capacity BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
    DECLARE stage_capacity INT;
    DECLARE current_ticket_count INT;

    -- Get the stage capacity for this event
    SELECT s.max_capacity INTO stage_capacity
    FROM Event e
    JOIN Stage s ON e.stage_id = s.stage_id
    WHERE e.event_id = NEW.event_id;

    -- Count current tickets
    SELECT COUNT(*) INTO current_ticket_count
    FROM Ticket
    WHERE event_id = NEW.event_id;

    -- Check if adding one more ticket would exceed capacity
    IF (current_ticket_count + 1) > stage_capacity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot sell more tickets than stage capacity';
    END IF;
END//
DELIMITER ;
```

- Αποφυγή ύπαρξης διπλότυπου εισιτηρίου για τον ίδιο επισκέπτη, την ίδια μέρα και την ίδια παράσταση

Πριν από προσθήκη εισιτηρίου: Η ημερομηνία της παράστασης αποθηκεύεται στο event_date. Ελέγχουμε αν υπάρχει ήδη εισιτήριο για αυτή τη παράσταση από τον καινούριο επισκέπτη και, αν υπάρχει, εμποδίζουμε την καινούρια αγορά.

```
-- Trigger to prevent duplicate tickets for same visitor, day and event
DELIMITER //
CREATE TRIGGER check_duplicate_ticket BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
    DECLARE ticket_count INT;
    DECLARE event_date DATE;

    -- Get the event date
    SELECT event_date INTO event_date
    FROM Event
    WHERE event_id = NEW.event_id;

    -- Check if visitor already has a ticket for an event on the same day
    SELECT COUNT(*) INTO ticket_count
    FROM Ticket t
    JOIN Event e ON t.event_id = e.event_id
    WHERE t.visitor_id = NEW.visitor_id
    AND e.event_date = event_date
    AND t.event_id = NEW.event_id;

    IF ticket_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Visitor already has a ticket for this event on this day';
    END IF;
END//
DELIMITER ;
```

- Αποφυγή ταυτόχρονης συμμετοχής καλλιτέχνη/μπάντας σε δύο σκηνές

Πριν από προσθήκη εμφάνισης: Αποθηκεύουμε την ημερομηνία της εμφάνισης στο event_date. Εάν πρόκειται για καλλιτέχνη, ελέγχουμε την ύπαρξη άλλων εμφανίσεων του ίδιου καλλιτέχνη οι οποίες να πραγματοποιούνται σε ώρα που επικαλύπτει αυτήν της νέας εμφάνισης, και ενημερώνουμε κατάλληλα το conflict_count. Αντίστοιχα προσεγγίζουμε την περίπτωση μιας νέας μπάντας.

```
-- Trigger to check artist/band can't perform at two stages simultaneously
DELIMITER //
CREATE TRIGGER check_artist_simultaneous_performances BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
    DECLARE event_date DATE;
    DECLARE conflict_count INT DEFAULT 0;

    -- Get event date
    SELECT event_date INTO event_date FROM Event WHERE event_id = NEW.event_id;

    -- Check for artist conflicts
    IF NEW.artist_id IS NOT NULL THEN
        SELECT COUNT(*) INTO conflict_count
        FROM Performance p
        JOIN Event e ON p.event_id = e.event_id
        WHERE p.artist_id = NEW.artist_id
        AND e.event_date = 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));
    ELSE
        -- Check for band conflicts
        SELECT COUNT(*) INTO conflict_count
        FROM Performance p
        JOIN Event e ON p.event_id = e.event_id
        WHERE p.band_id = NEW.band_id
        AND e.event_date = 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));
    END IF;

    IF conflict_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist/Band cannot perform at two stages simultaneously';
    END IF;
END//
DELIMITER ;
```

- Αποτροπή συμμετοχής καλλιτέχνη/μπάντας για πάνω από 3 συνεχή έτη.

Πριν από προσθήκη εμφάνισης: Πρώτα αποθηκεύουμε στο festival_year την ημερομηνία του festival στο οποίο ανήκει η νέα εμφάνιση. Στη συνέχεια βρίσκουμε όλα τα event που συμμετείχε ο καλλιτέχνης, στα οποία η χρονιά είναι διαφορετική και ταυτόχρονα συμπίπτει με τις 3 προηγούμενες χρονιές, αποθηκεύοντας τον αριθμό αυτό στο consecutive_years. Αν αυτός ο αριθμός είναι τρία, σημαίνει πως έχει ήδη συμμετάσχει το μέγιστο των επιτρεπτών φορών, αρά αποτρέπεται η προσθήκη μιας ακόμα εμφάνισής του. Αντίστοιχα προσεγγίζεται και η περίπτωση μιας μπάντας.

```
-- Trigger to ensure an artist/band doesn't perform for more than 3 consecutive years
DELIMITER //
CREATE TRIGGER check_artist_consecutive_years BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
    DECLARE consecutive_years INT DEFAULT 0;
    DECLARE festival_year INT;

    -- Get the festival year
    SELECT YEAR(event_date) INTO festival_year
    FROM Event
    WHERE event_id = NEW.event_id;

    -- Check for artist
    IF NEW.artist_id IS NOT NULL THEN
        SELECT COUNT(DISTINCT YEAR(e.event_date)) INTO consecutive_years
        FROM Performance p
        JOIN Event e ON p.event_id = e.event_id
        WHERE p.artist_id = NEW.artist_id
        AND YEAR(e.event_date) BETWEEN (festival_year - 3) AND (festival_year - 1);
    ELSE
        -- Check for band
        SELECT COUNT(DISTINCT YEAR(e.event_date)) INTO consecutive_years
        FROM Performance p
        JOIN Event e ON p.event_id = e.event_id
        WHERE p.band_id = NEW.band_id
        AND YEAR(e.event_date) BETWEEN (festival_year - 3) AND (festival_year - 1);
    END IF;

    IF consecutive_years >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist/Band cannot perform for more than 3 consecutive years';
    END IF;
END//
DELIMITER ;
```

- Έλεγχος του αριθμού του προσωπικού ασφαλείας να είναι τουλάχιστον το 5% της χωρητικότητας.

Μετά από προσθήκη event: Θέτουμε στο stage_capacity την χωρητικότητα της σκηνής, και έπειτα υπολογίζουμε το required_staff που θα απαιτείται για τη συγκεκριμένη σκηνή. Μετράμε τον αριθμό του προσωπικού ασφαλείας που έχει ανατεθεί σε αυτό το event μέσα από τον πίνακα του StaffAssignment, και τον αποθηκεύουμε στο security_count. Αν είναι μικρότερος από το required_staff, τυπώνεται ένα μήνυμα στον βοηθητικό πίνακα EventWarnings ώστε να το λάβουμε υπόψιν.

```
-- Trigger to ensure security staff is at least 5% of capacity
DELIMITER //
CREATE TRIGGER check_security_staff AFTER INSERT ON Event
FOR EACH ROW
BEGIN
    DECLARE stage_capacity INT;
    DECLARE security_count INT DEFAULT 0;
    DECLARE required_staff INT;

    -- Get stage capacity
    SELECT max_capacity INTO stage_capacity
    FROM Stage
    WHERE stage_id = NEW.stage_id;

    -- Calculate required security staff (at least 5% of capacity)
    SET required_staff = CEILING(stage_capacity * 0.05);

    -- Count security staff for this event
    SELECT COUNT(*) INTO security_count
    FROM StaffAssignment sa
    JOIN Staff s ON sa.staff_id = s.staff_id
    WHERE sa.event_id = NEW.event_id AND s.staff_type = 'security';

    -- Insert warning into a log table if security staff is insufficient
    IF security_count < required_staff THEN
        INSERT INTO EventWarnings (event_id, warning_type, warning_message, created_at)
        VALUES (NEW.event_id, 'SECURITY', CONCAT('Security staff insufficient: ', security_count, ' of ', required_staff, ' required'), NOW());
    END IF;
END//
DELIMITER ;
```

- Έλεγχος του αριθμού βοηθητικού προσωπικού να είναι τουλάχιστον το 2% της χωρητικότητας.

Λειτουργεί με αντίστοιχο τρόπο με το προηγούμενο trigger, με τις διαφορές πως αυτό ασχολείται με το staff_type = support και πως το όριο ανοχής είναι 2% της χωρητικότητας της σκηνής.

```
-- Trigger to ensure support staff is at least 2% of capacity
DELIMITER //
CREATE TRIGGER check_support_staff AFTER INSERT ON Event
FOR EACH ROW
BEGIN
    DECLARE stage_capacity INT;
    DECLARE support_count INT DEFAULT 0;
    DECLARE required_staff INT;

    -- Get stage capacity
    SELECT max_capacity INTO stage_capacity
    FROM Stage
    WHERE stage_id = NEW.stage_id;

    -- Calculate required security staff (at least 5% of capacity)
    SET required_staff = CEILING(stage_capacity * 0.02);

    -- Count security staff for this event
    SELECT COUNT(*) INTO support_count
    FROM StaffAssignment sa
    JOIN Staff s ON sa.staff_id = s.staff_id
    WHERE sa.event_id = NEW.event_id AND s.staff_type = 'support';

    -- Insert warning into a log table if security staff is insufficient
    IF support_count < required_staff THEN
        INSERT INTO EventWarnings (event_id, warning_type, warning_message, created_at)
        VALUES (NEW.event_id, 'SUPPORT', CONCAT('Support staff insufficient: ', support_count, ' of ', required_staff, ' required'), NOW());
    END IF;
END//
DELIMITER ;
```

- Διαχείριση εισιτηρίου μετά από μεταπώληση.

Μετά από προσθήκη εισιτηρίου: Ελέγχουμε εάν πριν και μετά την προσθήκη του εισιτηρίου εάν το πεδίο is_for_resale από False έγινε True. Σε αυτή την περίπτωση υπάρχει πράγματι ένα νέο εισιτήριο για μεταπώληση, το οποίο και μεταφέρουμε στην ResaleSellerQueue. Ως επιπλέον ενέργεια ελέγχουμε και αν υπάρχει ήδη κάποιος πιθανός αγοραστής μέσω της διαδικασίας ProcessResaleQueue που εξηγείται αργότερα.

```
-- Trigger to process ticket resale when a match is found
DELIMITER //
CREATE TRIGGER process_ticket_resale AFTER UPDATE ON Ticket
FOR EACH ROW
BEGIN
    -- If a ticket was flagged for resale
    IF NEW.is_for_resale = TRUE AND OLD.is_for_resale = FALSE THEN
        INSERT INTO ResaleSellerQueue (ticket_id, request_date, is_sold)
        VALUES (NEW.ticket_id, NOW(), FALSE);

        -- Check if there are buyers waiting for this ticket category and event
        CALL ProcessResaleQueue(NEW.ticket_id);
    END IF;
END//
DELIMITER ;
```

- Έλεγχος για την συνεισφορά μόνο των κριτικών των χρηστών που έχουν ενεργό εισιτήριο.

Πριν από προσθήκη κριτικής: Πρώτα ελέγχουμε εάν το εισιτήριο έχει χρησιμοποιηθεί, αποθηκεύοντας την πληροφορία αυτή στο is_used. Άμα είναι ανενεργό, η κριτική ακυρώνεται. Σε αντίθετη περίπτωση ελέγχουμε και το αν αντιστοιχεί η παράσταση του εισιτηρίου με κάποια παράσταση και, αν όλες οι συνθήκες είναι αληθείς, προχωράμε στην προσθήκη της κριτικής.

```
-- Trigger to check ratings only from ticket users
DELIMITER //
CREATE TRIGGER check_rating_validity BEFORE INSERT ON Rating
FOR EACH ROW
BEGIN
    DECLARE is_ticket_used BOOLEAN;
    DECLARE correct_event BOOLEAN DEFAULT FALSE;

    -- Check if ticket is used
    SELECT is_used INTO is_ticket_used
    FROM Ticket
    WHERE ticket_id = NEW.ticket_id;

    IF NOT is_ticket_used THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Only used tickets can submit ratings';
    END IF;

    -- Check if performance belongs to the ticket's event
    SELECT COUNT(*) > 0 INTO correct_event
    FROM Performance p
    JOIN Ticket t ON p.event_id = t.event_id
    WHERE p.performance_id = NEW.performance_id AND t.ticket_id = NEW.ticket_id;

    IF NOT correct_event THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Rating must be for a performance the visitor attended';
    END IF;
END//
DELIMITER ;
```

3. Εισαγωγή Δεδομένων στη Βάση

α. DML script

Dummy Data και load script

Με σκοπό την εύρυθμη λειτουργία της βάσης μας, κατασκευάσαμε το **"load.sql"** αρχείο για να γεμίσουμε με δεδομένα τη βάση μας. Ουσιαστικά, το **"create.py"** αρχείο στο φάκελο code δημιουργεί τυχαία δεδομένα με τη χρήση της βιβλιοθήκης **Faker** της Python και παράλληλα τα εισάγει σε μορφή "INSERT(data) ( );" σε νέο αρχείο "load.sql". Τα δεδομένα αυτά περιέχουν περιορισμούς δοσμένους από την εκφώνηση, όπως 50 καλλιτέχνες / συγκροτήματα, 30 σκηνές, 100 εμφανίσεις, 10 φεστιβάλ (εκ των οποίων 2 μελλοντικά) και 200 εισιτήρια. Το script της python δημιουργεί την πλειονότητα των δεδομένων για τα επιμέρους tables, παρόλα αυτά στο τέλος του αρχείου load.sql προσθέσαμε μερικά procedures της sql για να εισάγουμε χωρίς σφάλμα δεδομένα για εισιτήρια και βαθμολογήσεις. Τέλος, μεταβάλλαμε ορισμένα από τα δεδομένα στο κώδικα που προκλήθηκε από το script έτσι ώστε να φέρουν αποτελέσματα όλα τα queries που μας ζητούνται, αλλά και για να επιβεβαιώνεται η ορθότητα της βάσης.

β. Procedures

Στη βάση δεδομένων μας χρησιμοποιήσαμε ορισμένα Procedures, προκειμένου να αυτοματοποιήσουμε σύνθετες και επαναλαμβανόμενες διαδικασίες, όπως η δημιουργία εισιτηρίων και αξιολογήσεων. Ακολουθεί συνοπτική επεξήγηση των procedures που ενσωματώσαμε στο σύστημά μας.

- **ProcessResaleQueue**

Υλοποιεί τη διαχείριση μεταπώλησης εισιτηρίων, αντιστοιχίζοντας κάθε εισιτήριο που προσφέρεται για μεταπώληση στον πρώτο διαθέσιμο αγοραστή στην ουρά (FIFO) που ζητάει εισιτήριο για την ίδια εκδήλωση και κατηγορία. Ενημερώνει αυτόματα τον νέο κάτοχο του εισιτηρίου, ολοκληρώνει την αίτηση του αγοραστή και κλείνει την καταχώρηση του πωλητή.

Αρχικά δηλώνουμε τις απαιτούμενες μεταβλητές για να τις αποθηκεύσουμε προσωρινά, στις οποίες έπειτα περνάμε τα στοιχεία του εισιτηρίου που πρόκειται να μεταπωληθεί και βρίσκουμε τον πρώτο χρονικά επισκέπτη που πληροί τις προϋποθέσεις, όπως ίδια παράσταση και κατηγορία εισιτηρίου, ταξινομώντας σε αύξουσα σειρά request_date.

Αν βρει διαθέσιμο αγοραστή, τότε αλλάζει τον κάτοχο του εισιτηρίου, δηλώνει πως πια το εισιτήριο δεν είναι προς μεταπώληση και σημειώνει πως η αίτηση του αγοραστή έχει εξυπηρετηθεί.

```
-- Procedure to process the resale queue (FIFO)
DELIMITER //
CREATE PROCEDURE ProcessResaleQueue(IN p_ticket_id INT)
BEGIN
    DECLARE v_event_id INT;
    DECLARE v_category_id INT;
    DECLARE v_buyer_id INT;
    DECLARE v_buyer_queue_id INT;
    DECLARE v_price DECIMAL(10,2);
    DECLARE v_seller_id INT;

    -- Get ticket details
    SELECT event_id, category_id, price, visitor_id INTO v_event_id, v_category_id, v_price, v_seller_id
    FROM Ticket
    WHERE ticket_id = p_ticket_id;

    -- Find the earliest buyer in the queue matching event and category
    SELECT queue_id, visitor_id INTO v_buyer_queue_id, v_buyer_id
    FROM ResaleBuyerQueue
    WHERE event_id = v_event_id
    AND category_id = v_category_id
    AND is_fulfilled = FALSE
    ORDER BY request_date ASC
    LIMIT 1;

    -- If we found a buyer
    IF v_buyer_id IS NOT NULL THEN
        -- Update the ticket owner
        UPDATE Ticket
        SET visitor_id = v_buyer_id,
            is_for_resale = FALSE
        WHERE ticket_id = p_ticket_id;

        -- Mark buyer's request as fulfilled
        UPDATE ResaleBuyerQueue
        SET is_fulfilled = TRUE
        WHERE queue_id = v_buyer_queue_id;

        -- Mark seller's request as completed
        UPDATE ResaleSellerQueue
        SET is_sold = TRUE
        WHERE ticket_id = p_ticket_id;
    END IF;
END//
DELIMITER ;
```

- **AssignStaffToEvents**:

Αυτή η διαδικασία αναλαμβάνει αυτόματα την ανάθεση προσωπικού (ασφάλειας, υποστήριξης και τεχνικού) σε κάθε εκδήλωση, με βάση το ποσοστό της χωρητικότητας που απαιτείται για κάθε τύπο. Ελέγχει πόσο προσωπικό έχει ήδη ανατεθεί και προσθέτει επιπλέον μόνο αν χρειάζεται, εξασφαλίζοντας ότι κάθε εκδήλωση θα καλυφθεί επαρκώς.

Αρχικά δημιουργεί κέρσορα για να διατρέξει όλες τις παραστάσεις και τις αντίστοιχές τους χωρητικότητες από τους πίνακες Event και Stage (σημαντική είναι η χρήση της σημαίας done που δηλώνει πότε ο κέρσορας έχει τελειώσει με τις παραστάσεις) και ορίζει ένα χειριστή για την περίπτωση που δεν βρεθούν άλλες εγγραφές κατά τη διάρκεια της εκτέλεσης του κέρσορα, θέτοντας την τιμή της σημαίας done σε TRUE.

Δημιουργώντας ένα βρόχο που θα επαναλαμβάνεται για κάθε παράσταση που επιστρέφει ο κέρσορας, διαβάζει τις πληροφορίες της και τις αποθηκεύει στις μεταβλητές που είχαν γίνει declared στην αρχή.

Έπειτα, ορίζει τον απαιτούμενο αριθμό προσωπικού για κάθε τύπο, στρογγυλοποιώντας προς τα πάνω (CEILING) και ελέγχει πόσοι υπάλληλοι έχουν ήδη ανατεθεί, υπολογίζοντας πόσοι επιπλέον υπαλλήλους πρέπει να προστεθούν σε κάθε τύπο προσωπικού.

Τέλος, ακολουθώντας την ίδια διαδικασία για κάθε τύπο προσωπικού, εισάγει τους πρώτους διαθέσιμους υπαλλήλους που δεν έχουν ήδη ανατεθεί στην παράσταση, μέχρι να καλυφθεί η απαιτούμενη ποσότητα.

```sql
-- Assign Staff to Events
-- Create a procedure to automatically assign staff to events based on the requirements
DELIMITER //

CREATE PROCEDURE AssignStaffToEvents()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE event_id_var INT;
    DECLARE capacity_var INT;
    DECLARE required_security INT;
    DECLARE required_support INT;
    DECLARE required_technical INT;
    DECLARE current_security INT;
    DECLARE current_support INT;
    DECLARE current_technical INT;
    DECLARE security_to_add INT;
    DECLARE support_to_add INT;
    DECLARE technical_to_add INT;
    DECLARE staff_counter INT;
    -- Cursor for events
    DECLARE event_cursor CURSOR FOR
        SELECT e.event_id, s.max_capacity
        FROM Event e
        JOIN Stage s ON e.stage_id = s.stage_id
        ORDER BY e.event_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN event_cursor;

    read_loop: LOOP
        FETCH event_cursor INTO event_id_var, capacity_var;

        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Calculate required staff
        SET required_security = CEILING(capacity_var * 0.05);
        SET required_support = CEILING(capacity_var * 0.02);
        SET required_technical = CEILING(capacity_var * 0.03); -- Assume 3% technical staff requirement

        -- Get current staff count
        SELECT COUNT(*) INTO current_security
        FROM StaffAssignment sa
        JOIN Staff s ON sa.staff_id = s.staff_id
        WHERE sa.event_id = event_id_var AND s.staff_type = 'security';

        SELECT COUNT(*) INTO current_support
        FROM StaffAssignment sa
        JOIN Staff s ON sa.staff_id = s.staff_id
        WHERE sa.event_id = event_id_var AND s.staff_type = 'support';

        SELECT COUNT(*) INTO current_technical
        FROM StaffAssignment sa
        JOIN Staff s ON sa.staff_id = s.staff_id
        WHERE sa.event_id = event_id_var AND s.staff_type = 'technical';

        -- Calculate how many staff to add
        SET security_to_add = GREATEST(0, required_security - current_security);
        SET support_to_add = GREATEST(0, required_support - current_support);
        SET technical_to_add = GREATEST(0, required_technical - current_technical);
```

```
        -- Add security staff
        IF security_to_add > 0 THEN
            SET staff_counter = 0;

            INSERT INTO StaffAssignment (staff_id, event_id)
            SELECT s.staff_id, event_id_var
            FROM Staff s
            WHERE s.staff_type = 'security'
            AND NOT EXISTS (SELECT 1 FROM StaffAssignment sa WHERE sa.staff_id = s.staff_id AND sa.event_id = event_id_var)
            LIMIT security_to_add;
        END IF;

        -- Add support staff
        IF support_to_add > 0 THEN
            SET staff_counter = 0;

            INSERT INTO StaffAssignment (staff_id, event_id)
            SELECT s.staff_id, event_id_var
            FROM Staff s
            WHERE s.staff_type = 'support'
            AND NOT EXISTS (SELECT 1 FROM StaffAssignment sa WHERE sa.staff_id = s.staff_id AND sa.event_id = event_id_var)
            LIMIT support_to_add;
        END IF;

        -- Add technical staff
        IF technical_to_add > 0 THEN
            SET staff_counter = 0;

            INSERT INTO StaffAssignment (staff_id, event_id)
            SELECT s.staff_id, event_id_var
            FROM Staff s
            WHERE s.staff_type = 'technical'
            AND NOT EXISTS (SELECT 1 FROM StaffAssignment sa WHERE sa.staff_id = s.staff_id AND sa.event_id = event_id_var)
            LIMIT technical_to_add;
        END IF;
    END LOOP;

    CLOSE event_cursor;
END //

DELIMITER ;
```

- **GenerateTickets**:

Η διαδικασία GenerateTickets δημιουργεί 200 εισιτήρια με τυχαία χαρακτηριστικά όπως τιμή, τρόπος πληρωμής και κατηγορία, και εισάγει μερικά στη λίστα μεταπώλησης.

Αποθηκεύει στις μεταβλητές που έγιναν declare τον αριθμό των παραστάσεων και των επισκεπτών και σε βρόχο 200 επαναλήψεων, επιλέγει παράσταση και επισκέπτη με modulo ώστε να επαναχρησιμοποιεί τις διαθέσιμες εγγραφές κυκλικά.

Αναφορικά με τα εισιτήρια, ανάλογα με τον αριθμό i της επανάληψης, καθορίζει την κατηγορία του εισιτηρίου (περισσότερα General, λιγότερα VIP και ακόμα λιγότερα Backstage) και την τιμή του εισιτηρίου, ανάλογα την κατηγορία, καθώς και τον τρόπο πληρωμής. Επιπροσθέτως, παράγει έναν έγκυρο 13-ψήφιο EAN κωδικό και χειρίζεται εάν το εισιτήριο έχει χρησιμοποιηθεί ή αν πρόκειται για μεταπώληση, εισάγοντας τελικά με τις παραπάνω πληροφορίες το εισιτήριο στη βάση δεδομένων μας.

Αυτή η διαδικασία επίσης καταχωρεί 20 επισκέπτες που αιτούνται εισιτήριο για μεταπώληση και βρίσκοντας όλα τα εισιτήρια που είναι για μεταπώληση, τα προσθέτει στην ουρά ResaleSellerQueue.

```
-- Insert Tickets (at least 200 tickets)
-- First, create a procedure to generate the tickets with proper EAN codes
DELIMITER //
CREATE PROCEDURE GenerateTickets()
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE j INT DEFAULT 1;
    DECLARE event_count INT;
    DECLARE visitor_count INT;
    DECLARE category_id INT;
    DECLARE price DECIMAL(10,2);
    DECLARE payment_method VARCHAR(20);
    DECLARE ean VARCHAR(13);
    DECLARE curr_event_id INT;
    DECLARE curr_visitor_id INT;
    DECLARE used BOOLEAN;
    DECLARE resale BOOLEAN;

    -- Get counts
    SELECT COUNT(*) INTO event_count FROM Event;
    SELECT COUNT(*) INTO visitor_count FROM Visitor;
```

```
    -- Generate 200 tickets
    WHILE i <= 200 DO
        -- Select event
        SET curr_event_id = (i % event_count) + 1;

        -- Select visitor
        SET curr_visitor_id = (i % visitor_count) + 1;

        -- Select category (mostly general, some VIP, few backstage)
        IF i % 10 = 0 THEN
            SET category_id = 2; -- VIP
        ELSEIF i % 50 = 0 THEN
            SET category_id = 3; -- Backstage
        ELSE
            SET category_id = 1; -- General
        END IF;

        -- Set price based on category
        IF category_id = 1 THEN
            SET price = 50.00 + (RAND() * 50);
        ELSEIF category_id = 2 THEN
            SET price = 150.00 + (RAND() * 100);
        ELSE
            SET price = 300.00 + (RAND() * 200);
        END IF;

        -- Set payment method
        IF i % 3 = 0 THEN
            SET payment_method = 'credit_card';
        ELSEIF i % 3 = 1 THEN
            SET payment_method = 'debit_card';
        ELSE
            SET payment_method = 'bank_transfer';
        END IF;

        -- Generate EAN code (simple for demonstration)
        SET ean = LPAD(i, 13, '0');

        -- Set used flag (tickets for past events are used)
        SELECT IF(event_date < CURDATE(), TRUE, FALSE) INTO used
        FROM Event WHERE event_id = curr_event_id;

        -- Set resale flag (small chance)
        SET resale = IF(i % 40 = 0, TRUE, FALSE);

        -- Create ticket
        INSERT INTO Ticket (event_id, visitor_id, category_id, purchase_date, price, payment_method, ean_code, is_used, is_for_resale)
        VALUES (
            curr_event_id,
            curr_visitor_id,
            category_id,
            DATE_SUB(CURDATE(), INTERVAL (365 - i) DAY),
            ROUND(price, 2),
            payment_method,
            ean,
            used,
            resale
        );

        SET i = i + 1;
    END WHILE;

    -- Add some buyers to resale queue
    WHILE j <= 20 DO
        INSERT INTO ResaleBuyerQueue (visitor_id, event_id, category_id, request_date, is_fulfilled)
        VALUES (
            (j % visitor_count) + 1,
            (j % event_count) + 1,
            IF(j % 5 = 0, 2, 1),  -- Mix of general and VIP
            DATE_SUB(CURDATE(), INTERVAL j DAY),
            FALSE
        );

        SET j = j + 1;
    END WHILE;

    -- Add some tickets to seller queue for those marked for resale
    INSERT INTO ResaleSellerQueue (ticket_id, request_date, is_sold)
    SELECT ticket_id, DATE_SUB(CURDATE(), INTERVAL RAND()*30 DAY), FALSE
    FROM Ticket
    WHERE is_for_resale = TRUE;
END //
DELIMITER ;
```

- **GenerateRatings**:

Η διαδικασία GenerateRatings προσθέτει αξιολογήσεις για χρησιμοποιημένα εισιτήρια. Περίπου το 70% αυτών των εισιτηρίων βαθμολογεί μια τυχαία παράσταση της εκδήλωσης με πέντε τυχαίες τιμές από 1 έως 5 σύμφωνα με ορισμένα κριτήρια.

Ορίζει έναν κέρσορα που περιέχει όλα τα χρησιμοποιημένα εισιτήρια, από τα οποία θα λάβουμε δεδομένα και ομοίως με το procedure AssignStaffToEvents, χρησιμοποιεί τη σημαία done για να τερματίζεται η επανάληψη όταν τελειώσουν τα αποτελέσματα του κέρσορα.

Για κάθε επανάληψη του βρόχου επιλέγεται μια τυχαία εμφάνιση από την παράσταση για να συσχετιστεί με τη βαθμολογία και παράγονται πέντε τυχαίοι ακέραιοι αριθμοί μεταξύ 1 και 5 για κάθε κατηγορία αξιολόγησης, καταχωρίζοντας τελικά τις εγγραφές εφόσον υπάρχει έγκυρο performance_id.

```sql
-- Insert Ratings
-- Let's create a procedure to generate ratings for used tickets
DELIMITER //
CREATE PROCEDURE GenerateRatings()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE t_id INT;
    DECLARE e_id INT;
    DECLARE p_id INT;
    DECLARE rating1 INT;
    DECLARE rating2 INT;
    DECLARE rating3 INT;
    DECLARE rating4 INT;
    DECLARE rating5 INT;

    -- Cursor for used tickets
    DECLARE ticket_cursor CURSOR FOR
        SELECT t.ticket_id, t.event_id
        FROM Ticket t
        WHERE t.is_used = TRUE;

    -- Continue handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN ticket_cursor;

    read_loop: LOOP
        FETCH ticket_cursor INTO t_id, e_id;

        IF done THEN
            LEAVE read_loop;
        END IF;

        -- About 70% of used tickets have ratings
        IF RAND() < 0.7 THEN
            -- Get a random performance for this event
            SELECT performance_id INTO p_id
            FROM Performance
            WHERE event_id = e_id
            ORDER BY RAND()
            LIMIT 1;

            IF p_id IS NOT NULL THEN
                -- Generate random ratings between 1 and 5
                SET rating1 = FLOOR(1 + RAND() * 5);
                SET rating2 = FLOOR(1 + RAND() * 5);
                SET rating3 = FLOOR(1 + RAND() * 5);
                SET rating4 = FLOOR(1 + RAND() * 5);
                SET rating5 = FLOOR(1 + RAND() * 5);

                -- Insert the rating
                INSERT INTO Rating (ticket_id, performance_id, artist_performance, sound_lighting, stage_presence, organization, overall_impression)
                VALUES (t_id, p_id, rating1, rating2, rating3, rating4, rating5);
            END IF;
        END IF;
    END LOOP;

    CLOSE ticket_cursor;
END //
DELIMITER ;
```

Β. Σχεδιασμός και Υλοποίηση ερωτημάτων (Queries)

Σημείωση: Η υλοποίηση και τα αποτελέσματα των ερωτημάτων βρίσκονται στο directory sql/queries. Ακολουθεί η εξήγησή τους:

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

Αυτό το ερώτημα παράγει μια αναφορά των εσόδων του φεστιβάλ ανά έτος και τις μεθόδους πληρωμής. Συνδέει τους πίνακες Ticket → Event → Stage → Festival μέσω JOIN πινάκων ξεκινώντας από τον πίνακα Ticket, για να συνδέσει τα έσοδα με κάθε έτος φεστιβάλ. Ομαδοποιεί τα αποτελέσματα ανά έτος, δηλαδή YEAR(f.start_date) (εξάγεται από την ημερομηνία έναρξης του φεστιβάλ). Υπολογίζει το άθροισμα των τιμών των εισιτηρίων ως τα επιμέρους έσοδα από τις μεθόδους πληρωμής(credit_card_revenue, debit_card_revenue, bank_transfer_revenue), αλλά και τα συνολικά έσοδα από όλα τα εισιτήρια του έτους (total_revenue).Τέλος, ταξινομεί τα αποτελέσματα πρώτα κατά φθίνουσα σειρά χρόνου, εντός κάθε έτους.. Οι δείκτες idx_ticket_payment, idx_event_festival και idx_stage_festival βοηθούν στη βελτιστοποίηση αυτού του ερωτήματος επιταχύνοντας τις συνδέσεις μεταξύ των πινάκων.

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

Αυτό το ερώτημα παρέχει μια λίστα όλων των καλλιτεχνών του είδους Pop, δείχνοντας ποιοι εμφανίστηκαν στο φεστιβάλ του 2026. Πρώτα εντοπίζει όλους τους καλλιτέχνες Pop συνδέοντας τους πίνακες Artist → ArtistGenre → Genre μέσω της εντολής JOIN. Για κάθε καλλιτέχνη, χρησιμοποιεί μια έκφραση CASE με υποερώτημα EXISTS για να προσδιορίσει αν συμμετείχαν την ημερομηνία που του έχουμε ζητήσει. Το υποερώτημα στο CASE συνδέει τους πίνακες Performance → Event → Stage → Festival μέσω JOIN για να επαληθεύσει τη συμμετοχή σε μία συγκεκριμένη χρονιά για τον κάθε artist που ανήκει στο αντίστοιχο μουσικό είδος που ζητήσουμε. Αυτό το υποερώτημα επιστρέφει "Yes" ή "No" για κάθε καλλιτέχνη στη στήλη performed_in_festival_20XX. Τέλος, τα αποτελέσματα ταξινομούνται αλφαβητικά κατά όνομα καλλιτέχνη.

Ο δείκτης idx_performance_artist βοηθά στη βελτιστοποίηση της απόδοσης του υποερωτήματος.

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

Αυτό το ερώτημα εντοπίζει καλλιτέχνες που εμφανίστηκαν ως "warm up" περισσότερες από δύο φορές στο ίδιο φεστιβάλ. Ουσιαστικά, συνδέει τους πίνακες Performance → Artist και Performance → Event → Stage → Festival μέσω JOIN πινάκων, φιλτράρει μόνο για τύπους εμφάνισης "warm up" μέσω WHERE, και στο τέλος ομαδοποιεί τα αποτελέσματα ανά καλλιτέχνη και φεστιβάλ. Ταυτόχρονα, χρησιμοποιεί τη ρήτρα HAVING για να κρατήσει μόνο ομάδες με περισσότερες από 2 εμφανίσεις και επιστρέφει πληροφορίες καλλιτέχνη, στοιχεία φεστιβάλ και τον συνολικό αριθμό εμφανίσεων warm-up.

Ο δείκτης idx_performance_type βοηθά στη βελτιστοποίηση του φιλτραρίσματος της ρήτρας WHERE.

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

<u>Αρχική Έκδοση:</u> Αυτό το ερώτημα υπολογίζει τις μέσες βαθμολογίες κοινού για έναν συγκεκριμένο καλλιτέχνη (ID=4). Αρχικά, συνδέει τους πίνακες Artist → Performance → Rating μέσω απλών JOIN. Από εκεί, υπολογίζει μέσες βαθμολογίες τόσο για την ερμηνεία του καλλιτέχνη όσο και για τη συνολική εντύπωση, και επιλέγει-περιορίζει συγκεκριμένο id καλλιτέχνη. Τέλος, τα αποτελέσματα ομαδοποιούνται ανά πληροφορίες καλλιτέχνη.

Παρέχοντας ανάλυση με χρήση της “EXPLAIN SELECT …” και της “SET profiling =1; … SHOW PROFILES; “, παρατηρούμε ότι:

```
    ->     a.artist_id, a.artist_name;
+----+-------------+-------+-------+-----------------------------+-----------------------------+---------+-----------------------------------+------+-------------+
| id | select_type | table | type  | possible_keys               | key                         | key_len | ref                               | rows | Extra       |
+----+-------------+-------+-------+-----------------------------+-----------------------------+---------+-----------------------------------+------+-------------+
|  1 | SIMPLE      | a     | const | PRIMARY                     | PRIMARY                     | 4       | const                             |    1 |             |
|  1 | SIMPLE      | p     | ref   | PRIMARY,idx_performance_artist | idx_performance_artist | 5       | const                             |    4 | Using index |
|  1 | SIMPLE      | r     | ref   | idx_rating_performance      | idx_rating_performance      | 4       | pulse_university.p.performance_id |   50 |             |
+----+-------------+-------+-------+-----------------------------+-----------------------------+---------+-----------------------------------+------+-------------+
3 rows in set (0.001 sec)
```

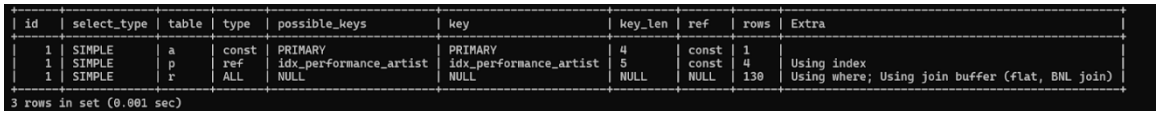
```
|       16 | 0.00092060 | EXPLAIN
```

Παρατηρούμε ότι το σύστημα επιλέγει μόνο του τα indexes, αλλά παρόλα αυτά, διατρέχει όλα τα rows του performance, για αυτό καταλήγει και στο χρόνο που παρατηρούμε.

- **<u>Query 4 Force Index:</u>**

Χρησιμοποιούμε το ίδιο ακριβώς query, με τη διαφορά ότι προσθέτουμε την εντολή `FORCE INDEX` στα JOIN των πινάκων Performance και Rating, η οποία ουσιαστικά δηλώνει το index με το οποίο θέλουμε να υπολογίσουμε το ερώτημα αυτό, χωρίς να αφήνουμε από μόνη της τη βάση να βρει αυτό που θεωρεί εκείνη τον βέλτιστο τρόπο. Με αυτό το τρόπο, το σύστημα επιλέγει τα indexes idx_performance_artist, idx_rating_performance χωρίς να ψάχνει το βέλτιστο τρόπο μόνο του. Έτσι , όμως, παρατηρούμε:

- Αν έχουμε λάθος force index, ή index το οποίο δεν βοηθάει, αυξάνεται απότομα ο χρόνος επίλυσης και ο αριθμός των σειρών που διατρέχει η βάση μας.

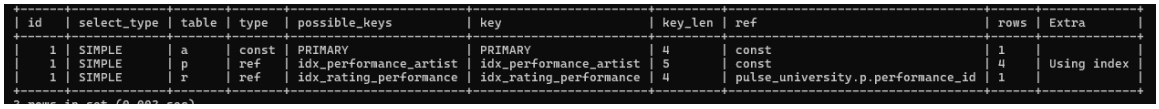


id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	p	ref	idx_performance_artist	idx_performance_artist	5	const	4	Using index
1	SIMPLE	r	ALL	NULL	NULL	NULL	NULL	130	Using where; Using join buffer (flat, BNL join)

3 rows in set (0.001 sec)

| | 166 | 0.00119370 | EXPLAIN

- Με σωστό indexing, παρατηρούμε ότι εκτός από το γεγονός ότι ο αριθμός των σειρών που διατρέχονται είναι πολύ λιγότερος, ο χρόνος κόβεται σχεδόν στα δύο, και στα 3 για λάθος force index:



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	p	ref	idx_performance_artist	idx_performance_artist	5	const	4	Using index
1	SIMPLE	r	ref	idx_rating_performance	idx_rating_performance	4	pulse_university.p.performance_id	1	

| | 18 | 0.00047580 | EXPLAIN

- **<u>ΣΤΡΑΤΗΓΙΚΕΣ JOIN:</u>**

Αναζητώντας διαφορετικές στρατηγικές για βελτιστοποίηση του ερωτήματος, καταρχάς αλλάξαμε τα JOIN των πινάκων Performance και Rating σε INNER JOIN, καθώς έτσι το σύστημα λαμβάνει αρχικά από τον πίνακα Performance τις εγγραφές που αφορούν μόνο τον καλλιτέχνη με το ID που ζητάμε, ενώ στη συνέχεια για το Rating λαμβάνουμε τα rating για μόνο τα performances με artist_ID = 4. Επιπλέον, μέσω της αναζήτησής μας στην βιβλιογραφία του συστήματος που χρησιμοποιούμε (MariaDB 10.4.32) παρατηρήσαμε πως το σύστημα έχει εν γένει χαρακτηριστικά τα οποία μας επιτρέπουν να χρησιμοποιήσουμε διαφορετικές στρατηγικές βελτιστοποίησης του συστήματος.

Η MariaDB βασίζεται σε Block Based Join algorithms, με επίπεδα 1 έως 8 ανάλογα τη διαμόρφωση που θέλουμε. Αυτά αποτελούν:

- 1 – Flat BNL
- 2 – Incremental BNL
- 3 – Flat BNLH
- 4 – Incremental BNLH
- 5 – Flat BKA

- 6 – Incremental BKA
- 7 – Flat BKAH
- 8 – Incremental BKAH

Το σύστημα αρχικά βασίζεται σε τρία optimizer switches:

- join_cache_incremental,
- join_cache_hashed,
- join_cache_bka

και τα θέτουμε σε λειτουργία μέσω των εντολών, π.χ. “SET SESSION optimizer_switch = 'join_cache_hashed=on,join_cache_bka=off';”. Οι **incremental μορφές** των block-based join αλγορίθμων αναφέρονται σε ειδικούς **incremental buffers** που αποθηκεύουν μόνο τα ενδιαφέροντα πεδία του δεύτερου πίνακα και έναν δείκτη (reference) στον πρώτο πίνακα, αντί να αντιγράφουν ολόκληρες εγγραφές από κάθε πλευρά. Οι incremental παραλλαγές μπορούν να μειώσουν τον χρόνο επαναλαμβανόμενης ανάγνωσης, και θέτουμε το switch τους σε “on” όταν θέλουμε να τους αξιοποιήσουμε σε κάθε επίπεδο. Το join_cache_hashed είναι μεταβλητή στο optimizer_switch που όταν είναι on επιτρέπει στον optimizer να επιλέξει τις hashed παραλλαγές BNLH και BKAH. Τέλος, η σημαία join_cache_bka στο optimizer_switch της MariaDB ελέγχει αν θα επιτρέπονται οι Batch Key Access (BKA) και Batch Key Access Hash (BKAH) παραλλαγές των block-based join αλγορίθμων.

- **1η Στρατηγική: Nested Loop Join**

Αρχικά, η στρατηγική αυτή αποτελεί την default στρατηγική της SQL, καθώς έχουμε default join_cache_level = 2. Επομένως, με hint στην αρχή του SELECT, σύστημά μας αρχικά διαλέγει έναν πίνακα ως “εξωτερικό”, και στη συνέχεια ,για κάθε γραμμή του εξωτερικού κάνει lookup στον εσωτερικό (inner) και ξανά στον εσωτερικό, είτε με full scan είτε με index, όπως δείξαμε στις τεχνικές με force index. Ιδανικό όταν ο ένας πίνακας είναι πολύ μικρός ή ο εσωτερικός διαθέτει κατάλληλο index. Η στρατηγική αυτή θα περιμένουμε να μην έχει διαφορά με την αρχική μας επιλογή, αλλά να κερδίσει με την βελτιστοποίηση των JOIN του πίνακα το οποίο και επιβεβαιώνουν τα traces:

```
MariaDB [pulse_university]> SET join_cache_level = 2;
Query OK, 0 rows affected (0.000 sec)

MariaDB [pulse_university]> SET SESSION optimizer_switch =
    -> 'join_cache_hashed=off,join_cache_bka=off';
Query OK, 0 rows affected (0.001 sec)
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	p	ref	PRIMARY,idx_performance_artist	idx_performance_artist	5	const	4	Using index
1	SIMPLE	r	ref	idx_rating_performance	idx_rating_performance	4	pulse_university.p.performance_id	1	

- **2η Στρατηγική: HashJoin**

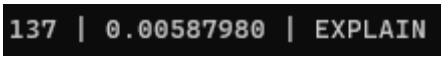
Όπως αναφέραμε και πριν, η `join_cache_level` (0–8) καθορίζει ποιες flat και incremental παραλλαγές των block-based αλγορίθμων είναι διαθέσιμες. Ακόμη κι αν η `join_cache_level` είναι υψηλή (π.χ. 8), χωρίς `join_cache_hashed=on` οι hashed παραλλαγές δεν θα χρησιμοποιηθούν. Έχουμε δύο βασικούς HASH JOIN αλγόριθμους: Block Nested Loop Hash (BNLH) και Batch Key Access Hash (BKAH). Ο πρώτος χωρίζεται σε δύο φάσεις, τις **build phase**: Δημιουργείται in-memory hash table με τα join κλειδιά του μικρότερου πίνακα και **probe phase**: Ο μεγαλύτερος πίνακας σαρώνεται και για κάθε εγγραφή γίνεται lookup στον hash πίνακα για γρήγορες αντιστοιχίες. Ο δεύτερος συνδυάζει το batch-collection pattern με hashed lookup. Αρχικά, συλλέγει κλειδιά σε παρτίδες (batches) από τον έναν πίνακα και στη συνέχεια δομεί hash table για όλα τα κλειδιά μαζί, επιταχύνοντας joins σε μεγάλες συγκεντρώσεις δεδομένων. Και οι δύο τρόποι είναι πολύ γρήγοροι σε μεγάλες ισοδύναμες συνενώσεις (equi-joins), και δεν απαιτούν ταξινόμηση.

Έτσι, ερευνήσαμε και τους δύο τύπους:

- flat BNLH JOIN με join_cache_level = 4 και SET SESSION optimizer_switch = 'join_cache_hashed=on,join_cache_bka=off', παρατηρούμε:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	p	ref	PRIMARY,idx_performance_artist	idx_performance_artist	5	const	4	Using where; Using index
1	SIMPLE	r	hash_ALL	idx_rating_performance	#hash#idx_rating_performance	4	pulse_university.p.performance_id	130	Using join buffer (flat, BNLH join)

Ενώ, άμα δεν χρησιμοποιήσουμε τα inner join στο πίνακα ο χρόνος αυξάνεται:



- flat BKAH JOIN με join_cache_level = 6 και SET SESSION optimizer_switch = 'join_cache_hashed=on,join_cache_bka=on';

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	p	ref	PRIMARY,idx_performance_artist	idx_performance_artist	5	const	4	Using index
1	SIMPLE	r	hash_ALL	idx_rating_performance	#hash#idx_rating_performance	4	pulse_university.p.performance_id	130	Using join buffer (flat, BNLH join)

```
|       22 | 0.00045300 | EXPLAIN
SELECT
```

3η Στρατηγική: Merge Join

Προτάθηκε από την εκφώνηση να ψάξουμε τρόπο για merge join βελτιστοποίηση, δηλαδή μία διαδικασία ταξινόμησης πινάκων και στη συνέχεια το "merge" τους. Παρόλα αυτά, μετά από σχετική αναζήτηση στο documentation της MariaDB, παρατηρήσαμε πως δεν υπάρχει σχετικό optimizer για merge join. Καταλάβαμε πως οι δημιουργοί αποφάσισαν πως δεν χρειάστηκε να φτιάξουν αλγόριθμο με sorting πριν την ένωση tables, καθώς οι block based join algorithms κάνουν παρόμοια δουλειά με πολλές φορές γρηγορότερα χαρακτηριστικά.

Τα traces του 4ου ερωτήματος παρατίθενται στο αρχείο Q04_traces.txt, όπως και στη συνέχεια τα traces του ερωτήματος 6.

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

Αυτό το ερώτημα εντοπίζει νέους καλλιτέχνες (κάτω των 30 ετών) με τις περισσότερες εμφανίσεις σε φεστιβάλ. Μέσω τη χρήση της JOIN, συνδέει τους πίνακες Artist → Performance → Event → Stage → Festival και φιλτράρει για καλλιτέχνες νεότερους από 30 ετών μέσω του WHERE. Υπολογίζει την ηλικία κάθε καλλιτέχνη χρησιμοποιώντας TIMESTAMPDIFF για την διαφορά των χρόνων και CURDATE() για την ημερομηνία όταν γίνεται η ερώτηση. Επιπλέον, μετρά τα διαφορετικά φεστιβάλ για κάθε καλλιτέχνη με COUNT() και DISTINCT() ⇒ distinct festival count. Τέλος, ταξινομεί τα αποτελέσματα πρώτα κατά τις περισσότερες εμφανίσεις σε φεστιβάλ, και μετά αλφαβητικά κατά όνομα

Ο νέος δείκτης idx_artist_birth_date που προσθέσαμε θα βελτιστοποιήσει αυτό το ερώτημα επιταχύνοντας τη συνθήκη φιλτραρίσματος ηλικίας.

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

Αρχικά επιλέγουμε να φαίνονται στο output τα στοιχεία του επισκέπτη και της παράστασης που παρακολούθησε, καθώς και ο μέσος όρος αξιολόγησής του ανά παράσταση.

Με διαδοχικά JOIN (Ticket → Event → Performance) και το LEFT JOIN Performance → Rating, προκειμένου να συμπεριλάβει και περιπτώσεις που δεν υπάρχει βαθμολογία για

κάποια εμφάνιση, και έπειτα δηλώνοντας το visitor_id του επισκέπτη που επιθυμούμε, πράγματι λαμβάνουμε το ζητούμενο output ανά παράσταση.

Αξίζει να σημειωθεί η χρήση των indexes idx_ticket_visitor, idx_performance_event, idx_rating_ticket που, ακόμη και αν δεν δηλώνονται ρητά στο query, επιταχύνουν την εκτέλεσή του.

Χρησιμοποιώντας EXPLAIN, SET profiling=1 και SHOW PROFILES, εξάγουμε συμπεράσματα για τον χρόνο εκτέλεσης κάθε στρατηγικής, συγκεκριμένα για την αρχική έκδοση του query έχουμε:

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 f
1	SIMPLE	t	ref	idx_ticket_event_category,idx_ticket_visitor	idx_ticket_visitor	4	const	4	
1	SIMPLE	e	eq_ref	PRIMARY	PRIMARY	4	pulse_university.t.event_id	1	
1	SIMPLE	p	ref	idx_performance_event	idx_performance_event	4	pulse_university.t.event_id	1	Using index
1	SIMPLE	r	eq_ref	ticket_id,performance_id,idx_rating_ticket	ticket_id	8	pulse_university.t.ticket_id,pulse_university.p.performance_id	1	

```
1 | 0.00126780 | EXPLAIN
```

- <u>Force Index</u>

Με το FORCE INDEX, εξαναγκάζουμε τον optimizer να χρησιμοποιήσει συγκεκριμένο ευρετήριο για να βρει τα εισιτήρια πιο γρήγορα, κάτι το οποίο είναι χρήσιμο όταν ο optimizer δεν επιλέγει το καλύτερο ευρετήριο από μόνος του, όμως αυξάνει τον χρόνο εκτέλεσης σε περίπτωση που το ευρετήριο δεν είναι αποδοτικό. Στο συγκεκριμένο query ορίζουμε στον optimizer να χρησιμοποιήσει τα indexes idx_ticket_visitor, idx_performance_event, idx_rating_ticket και παρατηρούμε πως έχουμε καλύτερα αποτελέσματα σε σχέση με το αρχικό query.

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	t	ref	idx_ticket_visitor	idx_ticket_visitor	4	const	4	
1	SIMPLE	e	eq_ref	PRIMARY	PRIMARY	4	pulse_university.t.event_id	1	
1	SIMPLE	p	ref	idx_performance_event	idx_performance_event	4	pulse_university.t.event_id	1	Using index
1	SIMPLE	r	ref	idx_rating_ticket	idx_rating_ticket	4	pulse_university.t.ticket_id	58	Using where

```
2 | 0.00060560 | EXPLAIN
```

Οι διαφορετικές στρατηγικές JOIN που ακολουθήσαμε έχουν αναλυθεί στο query 4, ωστόσο τις εξηγούμε συνοπτικά και σε αυτό το ερώτημα για λόγους πληρότητας.

- Nested Loop Join

Σε αυτή τη στρατηγική κάθε γραμμή της πρώτης σχέσης ελέγχεται με όλες τις γραμμές της δεύτερης, το οποίο είναι αποτελεσματικό όταν πρόκειται για μικρούς πίνακες ή όταν ο δεύτερος πίνακας έχει index στην στήλη join, ωστόσο ο χρόνος αυξάνεται σημαντικά όταν οι πίνακες είναι μεγάλοι. Συνεπώς αναμένουμε τα αποτελέσματα να μην ενθαρρύνουν τη χρήση αυτής της στρατηγικής.

```
-- Query 6 Alternative with Nested Loop Join:
SELECT /*+ JOIN_ORDER(v,t,e,p,r) JOIN_PREFIX(v,t,e USING(nested_loop)*/
```

- Hash Join

Δημιουργεί έναν πίνακα κατακερματισμού για τον έναν πίνακα, συνήθως τον μικρότερο, και ελέγχει αν τα στοιχεία του άλλου πίνακα υπάρχουν εκεί. Αυτή η στρατηγική είναι χρήσιμη για μεγάλους πίνακες και όταν δεν υπάρχουν ευρετήρια, άρα θεωρητικά πρέπει να είναι ευνοϊκή για τη συγκεκριμένη περίπτωση.

```
-- Query 6 Alternative with Hash Join:
SELECT /*+ HASH_JOIN(v t e p r) */
```

- Merge Join

Στη MariaDB δεν υπάρχει optimizer για Merge Join.

Η MariaDB δεν μας δίνει μόνο τη δυνατότητα απλής χρήσης Nested Loop Join και Hash Join, αλλά προσφέρει ειδικές ρυθμίσεις για την εκτέλεση αυτών, ώστε να είναι οι καταλληλότερες για το κάθε query. Στο αρχείο Q06.sql φαίνονται οι κώδικες του query της αρχικής έκδοσης, με force index και Nested Loop Join, Hash Join και εμείς μεταβάλλοντας τις ρυθμίσεις του optimizer και τρέχοντας αυτούς τους κώδικες λάβαμε τα παρακάτω αποτελέσματα.

Συγκεκριμένα, ανάλογα με την τιμή join_cache_level, που καθορίζει πόσο επιθετικά γίνεται η αποθήκευση των ενδιάμεσων αποτελεσμάτων κατά την εκτέλεση ενός join, και τις ρυθμίσεις του optimizer_switch της MariaDB, δηλαδή join_cache_incremental, join_cache_hashed, join_cache_bka έχουμε τις εξής στρατηγικές μαζί με τον χρόνο εκτέλεσής τους στο συγκεκριμένο query:

1. **Flat BNL (Block Nested Loop)**: Είναι η βασική μορφή του αλγορίθμου Block Nested Loop Join, όπου κάθε μπλοκ του εξωτερικού πίνακα συγκρίνεται με ολόκληρο τον εσωτερικό πίνακα.

```
5 | 0.00047560 | EXPLAIN
```

2. **Incremental BNL**: Βελτιωμένη έκδοση του Flat BNL, όπου γίνεται σταδιακή αποθήκευση ενδιάμεσων αποτελεσμάτων για καλύτερη απόδοση.

```
8 | 0.00074450 | EXPLAIN
```

3. **Flat BNLH (BNL with Hashing)**: Χρησιμοποιεί κατακερματισμό (hashing) στον εσωτερικό πίνακα για ταχύτερη εύρεση των αντίστοιχων εγγραφών κατά τη σύγκριση.

id	select_type	table	type	possible_keys	key	key_len	ref
1	SIMPLE	v	const	PRIMARY	PRIMARY	4	const
1	SIMPLE	t	ref	idx_ticket_event_category,idx_ticket_visitor	idx_ticket_visitor	4	const
1	SIMPLE	e	hash_ALL	PRIMARY	#hash#PRIMARY	4	pulse_university.t.event_id
1	SIMPLE	p	hash_index	idx_performance_event	#hash#idx_performance_event:idx_performance_event	4:4	pulse_university.t.event_id
1	SIMPLE	r	hash_ALL	ticket_id,performance_id,idx_rating_ticket	#hash#ticket_id	8	pulse_university.t.ticket_id,pulse_university.p.performance_id

```
11 | 0.00070420 | EXPLAIN
```

4. **Incremental BNLH**: Συνδυάζει το hashing με σταδιακή αποθήκευση, επιταχύνοντας την επεξεργασία μεγάλων joins.

```
15 | 0.00057320 | EXPLAIN
```

5. **Flat BKA (Block Key Access)**: Αντί να σαρώνει, χρησιμοποιεί ευρετήριο για την εύρεση των αντίστοιχων εγγραφών του εσωτερικού πίνακα.

```
18 | 0.00047220 | EXPLAIN
```

6. **Incremental BKA**: Εκτελεί το ίδιο με το Flat BKA, αλλά προσθέτει ενδιάμεση αποθήκευση και βελτιστοποιήσεις για επαναχρησιμοποίηση των ευρετηρίων.

```
21 | 0.00119090 | EXPLAIN
```

7. **Flat BKAH (BKA with Hashing)**: Συνδυάζει το BKA με hashing, ώστε να μειώνει το κόστος των αναζητήσεων.

8. **Incremental BKAH**: Η πιο εξελιγμένη εκδοχή, που ενσωματώνει όλα τα παραπάνω χαρακτηριστικά (ευρετήρια, κατακερματισμό, σταδιακή εκτέλεση) για μέγιστη απόδοση.

Από τα παραπάνω αποτελέσματα συμπεραίνουμε πως, λόγω της εκμετάλλευσης των indexes που έχουμε συμπεριλάβει στη βάση δεδομένων μας, πιο γρήγορες είναι οι Flat BKA και Flat BNL. Οι hashed τεχνικές γενικά έχουν επίσης καλή απόδοση, ωστόσο η αποδοτικότητά τους φαίνεται κυρίως σε περιπτώσεις χωρίς indexes. Τέλος, παρατηρούμε πως οι incremental τεχνικές είναι βελτιστοποιημένες για χρήση μνήμης και όχι για ταχύτητα, καθώς έχουν σχετικά υψηλούς χρόνους εκτέλεσης.

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

Αυτό το ερώτημα έχει ως στόχο να εντοπίσει το φεστιβάλ στο οποίο συμμετείχε τεχνικό προσωπικό με τη χαμηλότερη συνολική εμπειρία. Αρχικά επιλέγουμε να φαίνονται στην απάντηση του RDBMS τα στοιχεία festival_id, festival_name και avg_experience_level, τα οποία εκφράζουν τον μοναδικό αριθμό που αντιπροσωπεύει το φεστιβάλ, το όνομα του φεστιβάλ και τον μέσο όρο εμπειρίας τεχνικού προσωπικού αντίστοιχα. Αξιοσημείωτη είναι η μετατροπή των λεκτικών χαρακτηρισμών της εμπειρίας σε αριθμούς ώστε να είναι δυνατή η εύρεση του μέσου όρου, ως εξής:

- trainee: 1
- beginner: 2
- intermediate: 3
- experienced: 4
- expert: 5

Έπειτα, χρησιμοποιώντας διαδοχικά JOIN φτάνουμε από το Festival στο Staff, συγκεκριμένα ακολουθώντας την πορεία Festival → Stage → Event → StaffAssignment→ Staff.

Επιλέγουμε με το WHERE clause ο τύπος του προσωπικού να είναι technical, σύμφωνα με την εκφώνηση, χρησιμοποιούμε το GROUP BY για να λάβουμε συγκεκριμένο

φεστιβάλ και με τον συνδυασμό ORDER BY και LIMIT 1 παίρνουμε πράγματι το φεστιβάλ με τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού.

Σημαντική είναι η χρήση των indexes idx_stage_festival, idx_event_festival, idx_staff_type που, βελτιστοποιούν την εκτέλεσή του query.

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

Το συγκεκριμένο ερώτημα εντοπίζει τα μέλη του υποστηρικτικού προσωπικού που δεν έχουν ανατεθεί να εργαστούν σε κάποιο γεγονός για μια συγκεκριμένη ημερομηνία, αξιοποιώντας τα indexes idx_staff_type, idx_event_date. Επιλέγουμε να φαίνονται στο output τα στοιχεία staff_id, first_name, last_name, role του προσωπικού και θέτουμε ως περιορισμό να είναι προσωπικό υποστήριξης (staff_type = 'support').

Στη συνέχεια από το StaffAssignment, με JOIN στην οντότητα Event, ορίζουμε την ημερομηνία που επιθυμούμε χρησιμοποιώντας το attribute event_date και λαμβάνουμε το ζητούμενο.

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

Το ερώτημα βρίσκει επισκέπτες που έχουν παρευρεθεί στον ίδιο αριθμό παραστάσεων σε ένα συγκεκριμένο έτος με άλλους επισκέπτες, εφόσον αυτός ο αριθμός είναι μεγαλύτερος από 3. Αρχικά επιλέγουμε να φαίνονται στο RDBMS τα στοιχεία visitor_id, first_name, last_name, event_year και events_attended, τα οποία αποτελούν τα πιο βασικά προσωπικά στοιχεία των επισκεπτών και τις ζητούμενες πληροφορίες της εκφώνησης.

Για κάθε επισκέπτη (έστω v1) έπειτα παίρνουμε τα στοιχεία του και υπολογίζουμε πόσες διαφορετικές παραστάσεις παρακολούθησε ανά έτος, χρησιμοποιώντας JOIN για να φτάσουμε από τον επισκέπτη στην παράσταση μέσω του εισιτηρίου, κρατώντας μόνο τα ενεργοποιημένα εισιτήρια.

Ομοίως λαμβάνουμε για κάποιον άλλο επισκέπτη (έστω v2) παρόμοιες πληροφορίες, χωρίς να μας ενδιαφέρουν τα προσωπικά του στοιχεία, αφού τον χρησιμοποιούμε απλά για τη συχνότητα παρακολούθησης ανά επισκέπτη και ανά έτος.

Τελικά για τους 2 παραπάνω επισκέπτες v1 και v2, λαμβάνουμε αυτούς για τους οποίους προφανώς ο v1 είναι διαφορετικός του v2, το έτος των παραστάσεων που παρακολούθησαν είναι κοινό και ο αριθμός των παραστάσεων αυτών είναι ίδιος, με τον περιορισμό να είναι μεγαλύτερος από 3. Τα αποτελέσματα ταξινομούνται κατά αύξουσα σειρά ως προς το έτος των παραστάσεων και για το ίδιο έτος κατά φθίνουσα σειρά ως προς τον αριθμό των

παραστάσεων που παρακολούθησε ο επισκέπτης, ενώ ο optimizer χρησιμοποιεί τα indexes idx_ticket_visitor, idx_event_date.

Query 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.

Αυτό το ερώτημα στοχεύει στον εντοπισμό των 3 πιο συχνών συνδυασμών μουσικών ειδών μεταξύ των καλλιτεχνών που συμμετείχαν σε φεστιβάλ. Κάνουμε SELECT ώστε να φαίνονται στην έξοδο τα ονόματα κάθε ζευγαριού μουσικών ειδών και ο αριθμός των παραστάσεων στις οποίες εμφανίστηκαν αυτά τα είδη.

Όσον αφορά τις εμφανίσεις των μεμονωμένων καλλιτεχνών, επιλέγουμε αυτές που αντιστοιχούν σε μη κενό artist_id με τον περιορισμό ότι ag1.genre_id < ag2.genre_id ώστε να αποφεύγουμε τις διπλοεγγραφές των μουσικών ειδών.

Ομοίως για τα συγκροτήματα ακολουθούμε την ίδια διαδικασία και συνδυάζουμε τα αποτελέσματα χρησιμοποιώντας UNION.

Στα παραπάνω ο optimizer αξιοποιεί τα indexes idx_performance_artist, idx_performance_band για μια αποδοτική εκτέλεση.

Τέλος ομαδοποιούμε τα αποτελέσματα σύμφωνα με τα ζευγάρια των μουσικών ειδών και τα ταξινομούμε με φθίνουσα σειρά ως προς τον αριθμό των εμφανίσεων, απεικονίζοντας τα 3 πρώτα, με το LIMIT 3.

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

Πρώτα δημιουργούμε ένα πίνακα MaxPerformances, ο οποίος εμπεριέχει τον αριθμό των εμφανίσεων των καλλιτεχνών (=performance_count), λαμβάνοντας υπόψιν όλες τις εμφανίσεις που έχουν γίνει. Αφού τον κατατάξουμε σε φθίνουσα σειρά, περιορίζουμε αυτόν τον πίνακα στο πρώτο στοιχείο του, οπότε τελικά περιέχει μόνο τον αριθμό εμφανίσεων του καλλιτέχνη με τις πιο πολλές εμφανίσεις. Έπειτα, διαλέγουμε όλους τους καλλιτέχνες οι οποίοι έχουν λιγότερες εμφανίσεις από τη διαφορά του MaxPerformances με το 5 (=difference), και τους επιστρέφουμε με τη μορφή artist_id, artist_name, performance_count και difference, με φθίνουσα σειρά αριθμού εμφανίσεων. Γίνεται χρήση του δείκτη idx_performance_artist για βελτιστοποίηση του ερωτήματος.

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

Στόχος του ερωτήματος είναι να επιστραφούν οι ημερομηνίες event_date ενός συγκεκριμένου festival (το οποίο θα δίνεται ως όρισμα στο ερώτημα) μαζί με τα διάφορα staff_count, που θα αντιπροσωπεύουν το πλήθος των διαφορετικών ειδών προσωπικού που θα αναλογούν σε κάθε παράσταση. Μέσα από κατάλληλη ένωση πινάκων, με χρήση των primary και των foreign keys, λαμβάνουμε τα αντίστοιχα πεδία που χρειαζόμαστε, ομαδοποιημένα και καταταγμένα με βάση το event_date. Για να έχουμε πιο κατανοητό αποτέλεσμα, έχουμε προσθέσει και μια στήλη total_count, η οποία παρουσιάζει το άθροισμα όλου του προσωπικού για κάθε παράσταση του festival. Η διαδικασία της ένωσης του πίνακα festival με τους υπόλοιπους επιταχύνεται μέσω του δείκτη idx_stage_festival.

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

Για το ερώτημα αυτό απαιτούνται αρκετές ενώσεις πινάκων, ώστε τελικά να καταλήξουμε σε έναν πίνακα που να περιέχει πληροφορίες για τους καλλιτέχνες αλλά και για τις τοποθεσίες των festival. Αυτό γίνεται πολύ πιο γρήγορο μέσω της χρήσης των δεικτών idx_festival_location, idx_stage_festival, idx_event_festival και idx_performance_event. Προϋπόθεση για την επιλογή των γραμμών αποτελεί το πλήθος των DISTINCT continent (=continent_count) να είναι μεγαλύτερο ή ίσο του 3. Με φθίνουσα σειρά, ως προς τον αριθμό των ηπείρων, επιστρέφουμε τα artist_id, artist_name και continent_count.

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

Για να διευκολύνουμε την προσέγγιση αυτού του ερωτήματος, δημιουργούμε πρώτα δύο βοηθητικούς πίνακες. Συλλέγουμε μέσα από τις απαραίτητες ενώσεις πινάκων τα genre_id, genre_name και το έτος από το event_date (=performance_year), όσον αφορά τα είδη μουσικής με τα οποία ασχολούνται οι καλλιτέχνες. Εκτελούμε την αντίστοιχη δουλεία για τις μπάντες, και στην συνέχεια τα ενώνουμε όλα μέσω της πράξης UNION ALL, κάτι το οποίο επιστρέφει όλες τις γραμμές και των δύο πινάκων, χωρίς να αφαιρεί τα διπλότυπα. Αυτός ο πίνακας αποθηκεύεται ως PerfGenre. Ύστερα φτιάχνουμε έναν πίνακα ονομαζόμενο GenreYearCount, ο οποίος περιέχει genre_id, genre_name, performance_year και το πλήθος των γραμμών του PerfGenre (=performance_count), για όσες γραμμές ισχύει ότι perfomance_count >= 3. Τέλος, ενώνουμε 2 διαφορετικούς πίνακες GenreYearCount, έστω g1 και g2, και επιλέγουμε το genre_name, τα δύο έτη και

το performance_count για όσες χρονιές επαληθεύουν το κριτήριο να έχουνε ίδιο performance_count και ταυτόχρονα το g2.performance_year = g1.performance_year + 1, το οποίο αναπαριστά την υλοποίηση των δύο συνεχόμενων ετών. Εδώ χρησιμοποιήθηκαν οι εξής δείκτες: idx_performance_artist, idx_performance_artist.

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

Μέσα από τις ενώσεις που φαίνονται, χρησιμοποιώντας για βελτιστοποίηση τον δείκτη idx_performance_artist, δημιουργούμε έναν πίνακα, από τον οποίο συλλέγουμε τα visitor_id, τα first_name και last_name του, το artist_id και το artist_name του καθώς και το σύνολο τον στηλών που αφορούν βαθμολογία που ορίζουν οι επισκέπτες (=total_rating). Κατατάσσοντας αυτόν τον πίνακα σε φθίνουσα σειρά με βάση το total_rating και παίρνοντας μόνο τις 5 πρώτες γραμμές, λαμβάνουμε τους top-5 καλλιτέχνες που πληρούν τα κριτήρια του ερωτήματος.

Γ. Παράρτημα

1. Λογισμικό και Εφαρμογές που χρησιμοποιήθηκαν

- MariaDB (v10.4.32)
- XAMPP (v8.0.30)
- MySQL Workbench(v8.0.42)
- Python 3.13.3 (Faker, Random, Datetime Libraries)

2. Αρχεία Υποβολής

- sql/install.sql : Το DDL script, που περιέχει όλους τους πίνακες, τα ευρετήρια και τα triggers για την δημιουργία της βάσης δεδομένων.
- sql/load.sql : Το DML script που περιέχει τις εντολές για την εισαγωγή τυχαίων δεδομένων στην βάση καθώς και τις procedures που απαιτούνται για τη λειτουργία της βάσης.
- sql/queries/Qx.sql και sql/queries/Qx_out.txt : Τα ερωτήματα της βάσης δεδομένων (SQL queries) και τα αντίστοιχα αποτελέσματα τους.
- diagrams/er.pdf και diagrams/relational.pdf : Το Διάγραμμα ER και το Relational Schema της βάσης δεδομένων μας.
- code/fake_data.py : Python script που χρησιμοποιείται για την δημιουργία του SQL script που εισάγει τυχαία δεδομένα στη βάση.
- docs/report.pdf : Αναφορά της εργασίας μας και των βημάτων που ακολουθήσαμε για να υλοποιήσουμε την εφαρμογή μας.

3.Οδηγίες Χρήσης της Βάσης Δεδομένων

Βήμα 1ο: Clone το git repository μέσω του terminal με την εντολή
git clone https://github.com/GregoryStam04/CONCERTS-DATA-BASE

Εναλλακτικά αποσυμπίεση του zip αρχείου σε ένα εύκολα προσβάσιμο directory

Βήμα 2ο: Εγκατάσταση xampp και ενεργοποίηση Apache και MySQL (MariaDB)

Βήμα 3ο: Στο directory που είναι εγκατεστημένη η MySQL (default στο xampp: C:\xampp\mysql\bin) εκτέλεση της εντολής mysql -u root -p

Βήμα 4ο: Για τη δημιουργία της δομής της βάσης δεδομένων τοπικά και τη φόρτωση των δεδομένων εκτελούμε τις εντολές:

```
source \path\to\install.sql
source \path\to\load.sql
```

Βήμα 5ο: Ομοίως με τη χρήση της εντολής source τρέχουμε τα queries και βλέπουμε στο terminal το output