ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ ΣΤΙΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

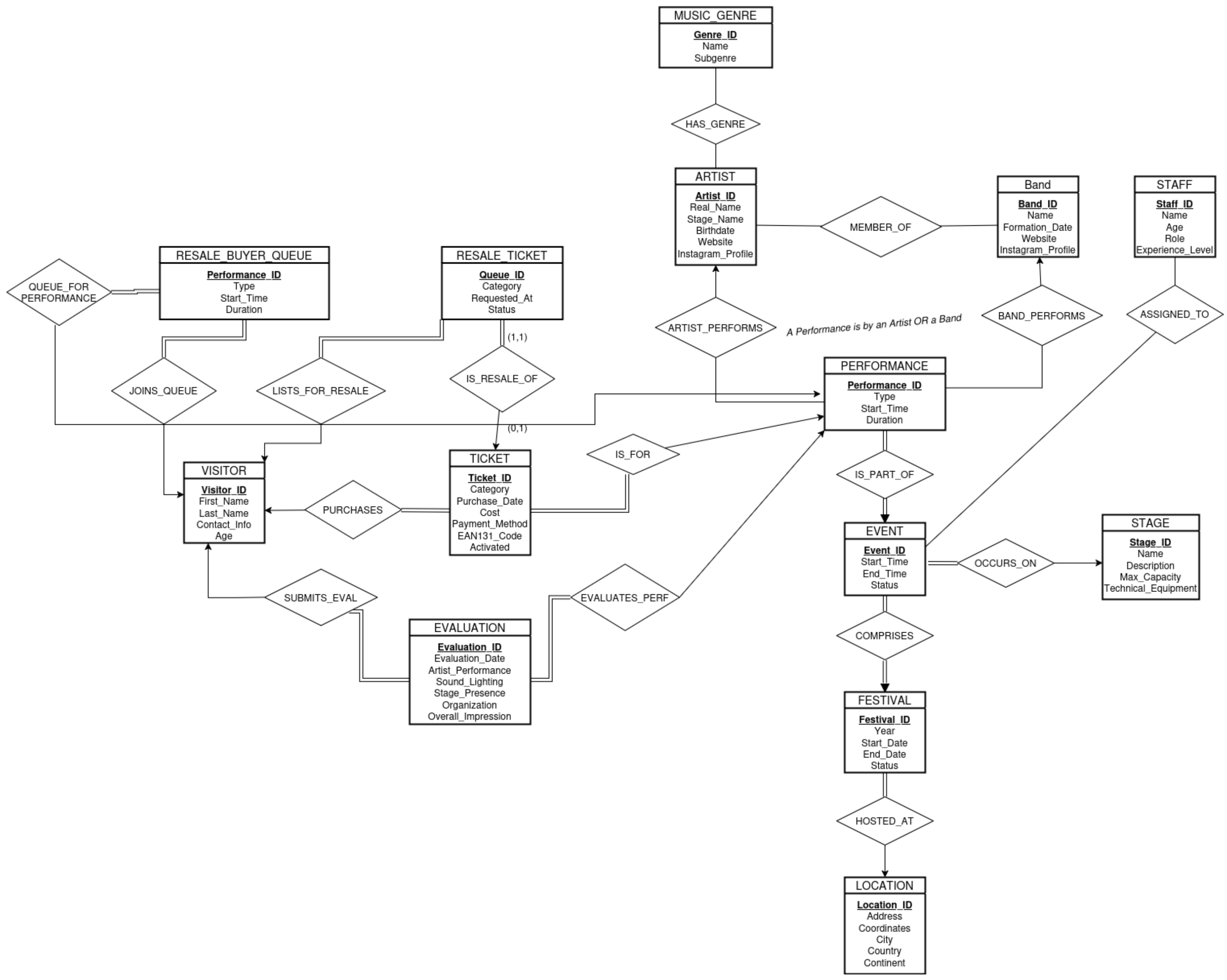
Ομάδα 133
Ανδριώτης Νικόλαος
03003244

Περιγραφή Εργασίας

Η παρούσα εργασία αφορά την δημιουργία μιας βάσης δεδομένων για ένα διεθνές φεστιβάλ μουσικής με όνομα **Pulse University**. Για την ανάπτυξη της βάσης χρησιμοποιήθηκε η **MariaDB/MySQL** σε συνδιασμό με ένα γραφικό περιβάλλον το **DbSchema** για την ανάπτυξη της βάσης και την εκτέλεση ερωτημάτων.

Σχεσιακό Διάγραμμα Βάσης Δεδομένων

Σαν πρώτο βήμα δημιουργούμε το ακόλουθο σχεσιακό διάγραμμα:



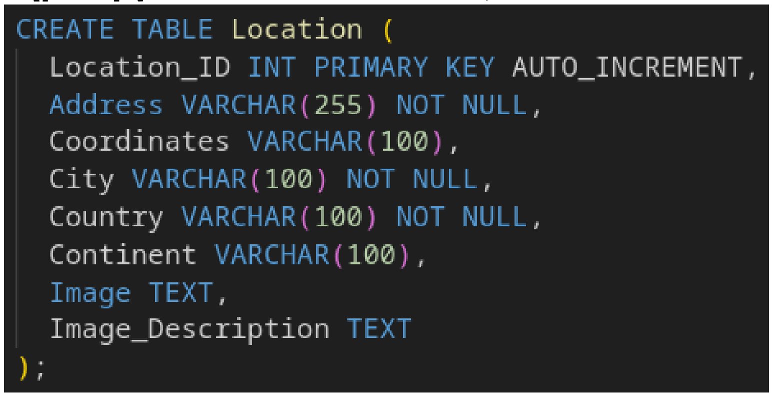
Το σχεσιακό μοντέλο είναι:

- **LOCATION** (Location_ID, Address, Coordinates, City, Country, Continent)
- **FESTIVAL** (Festival_ID, Year, Start_Date, End_Date, Status)
- **STAGE** (Stage_ID, Name, Description, Max_Capacity, Technical_Equipment)
- **EVENT** (Event_ID, Start_Time, End_Time, Status)
- **ARTIST** (Artist_ID, Real_Name, Stage_Name, Birthdate, Website, Instagram_Profile)
- **BAND** (Band_ID, Name, Formation_Date, Website, Instagram_Profile)
- **PERFORMANCE** (Performance_ID, Type, Start_Time, Duration)
- **MUSIC_GENRE** (Genre_ID, Name, Subgenre)
- **STAFF** (Staff_ID, Name, Age, Role, Experience_Level)
- **VISITOR** (Visitor_ID, First_Name, Last_Name, Contact_Info, Age)
- **TICKET** (Ticket_ID, Category, Purchase_Date, Cost, Payment_Method, EAN131_Code, Activated)
- **EVALUATION** (Evaluation_ID, Evaluation_Date, Artist_Performance, Sound_Lighting, Stage_Presence, Organization, Overall_Impression)
- **RESALE_TICKET** (ResaleTicket_ID, Listed_At, Status)
- **RESALE_BUYER_QUEUE** (Queue_ID, Category, Requested_At, Status)

Σχεδιασμός Βάσης Δεδομένων – SQL

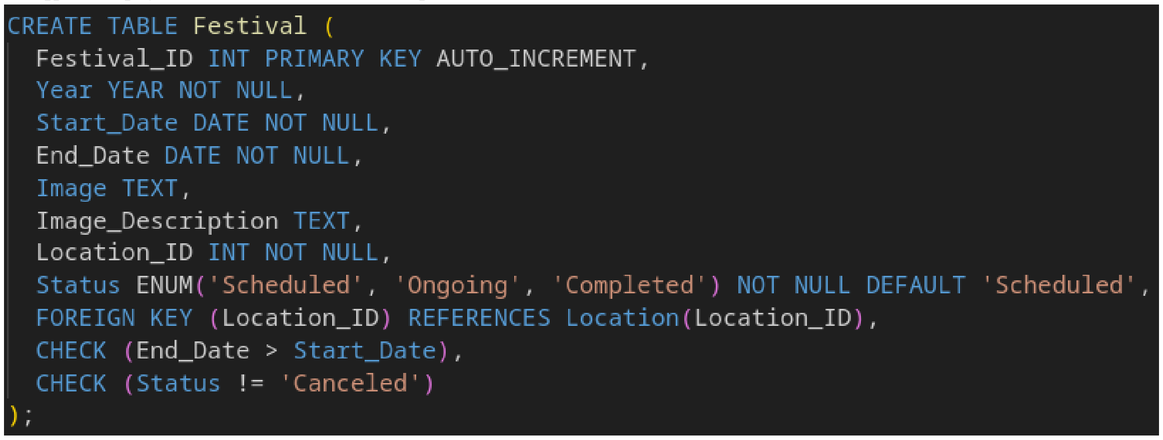
Παρακάτω οι πίνακες τις βάσεις:

Δημιουργία πίνακα Τοποθεσίας:



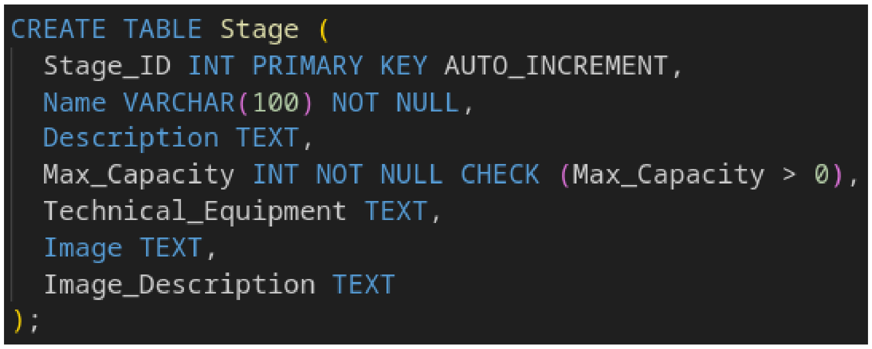
```sql
CREATE TABLE Location (
    Location_ID INT PRIMARY KEY AUTO_INCREMENT,
    Address VARCHAR(255) NOT NULL,
    Coordinates VARCHAR(100),
    City VARCHAR(100) NOT NULL,
    Country VARCHAR(100) NOT NULL,
    Continent VARCHAR(100),
    Image TEXT,
    Image_Description TEXT
);
```

<u>Δημιουργία πίνακα Φεστιβάλ:</u>



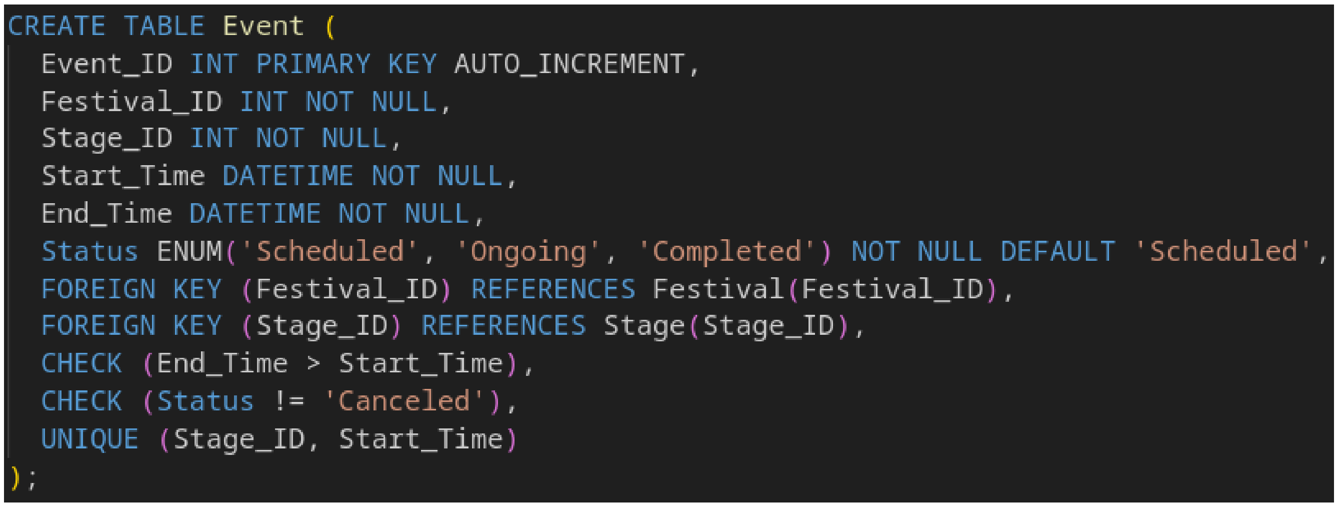
```sql
CREATE TABLE Festival (
  Festival_ID INT PRIMARY KEY AUTO_INCREMENT,
  Year YEAR NOT NULL,
  Start_Date DATE NOT NULL,
  End_Date DATE NOT NULL,
  Image TEXT,
  Image_Description TEXT,
  Location_ID INT NOT NULL,
  Status ENUM('Scheduled', 'Ongoing', 'Completed') NOT NULL DEFAULT 'Scheduled',
  FOREIGN KEY (Location_ID) REFERENCES Location(Location_ID),
  CHECK (End_Date > Start_Date),
  CHECK (Status != 'Canceled')
);
```

<u>Δημιουργία πίνακα Σκηνής:</u>



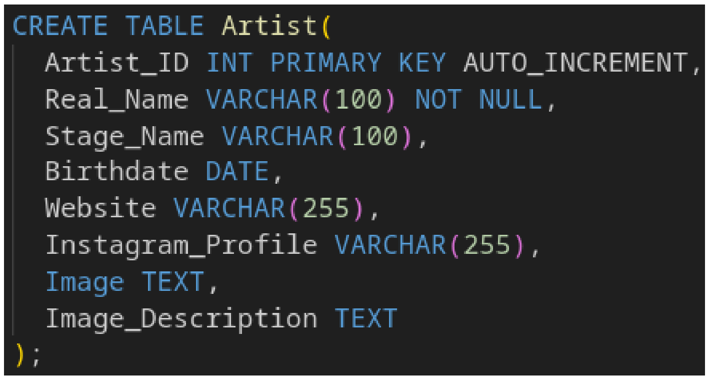
```sql
CREATE TABLE Stage (
  Stage_ID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  Description TEXT,
  Max_Capacity INT NOT NULL CHECK (Max_Capacity > 0),
  Technical_Equipment TEXT,
  Image TEXT,
  Image_Description TEXT
);
```

<u>Δημιουργία πίνακα Παράστασης:</u>



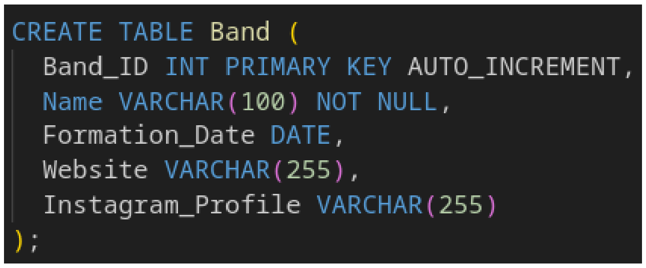
```sql
CREATE TABLE Event (
  Event_ID INT PRIMARY KEY AUTO_INCREMENT,
  Festival_ID INT NOT NULL,
  Stage_ID INT NOT NULL,
  Start_Time DATETIME NOT NULL,
  End_Time DATETIME NOT NULL,
  Status ENUM('Scheduled', 'Ongoing', 'Completed') NOT NULL DEFAULT 'Scheduled',
  FOREIGN KEY (Festival_ID) REFERENCES Festival(Festival_ID),
  FOREIGN KEY (Stage_ID) REFERENCES Stage(Stage_ID),
  CHECK (End_Time > Start_Time),
  CHECK (Status != 'Canceled'),
  UNIQUE (Stage_ID, Start_Time)
);
```

<u>Δημιουργία πίνακα Καλλιτέχνη:</u>



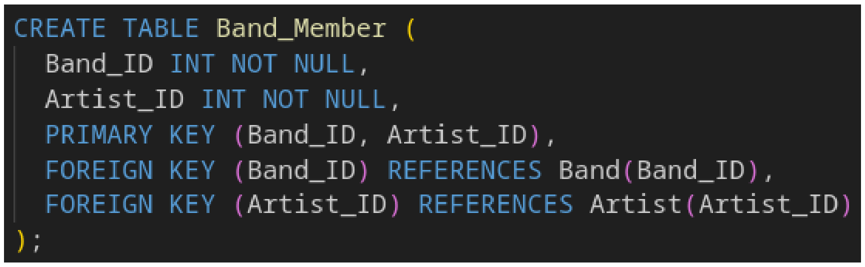
```sql
CREATE TABLE Artist(
  Artist_ID INT PRIMARY KEY AUTO_INCREMENT,
  Real_Name VARCHAR(100) NOT NULL,
  Stage_Name VARCHAR(100),
  Birthdate DATE,
  Website VARCHAR(255),
  Instagram_Profile VARCHAR(255),
  Image TEXT,
  Image_Description TEXT
);
```

<u>Δημιουργία πίνακα Μπάντας:</u>



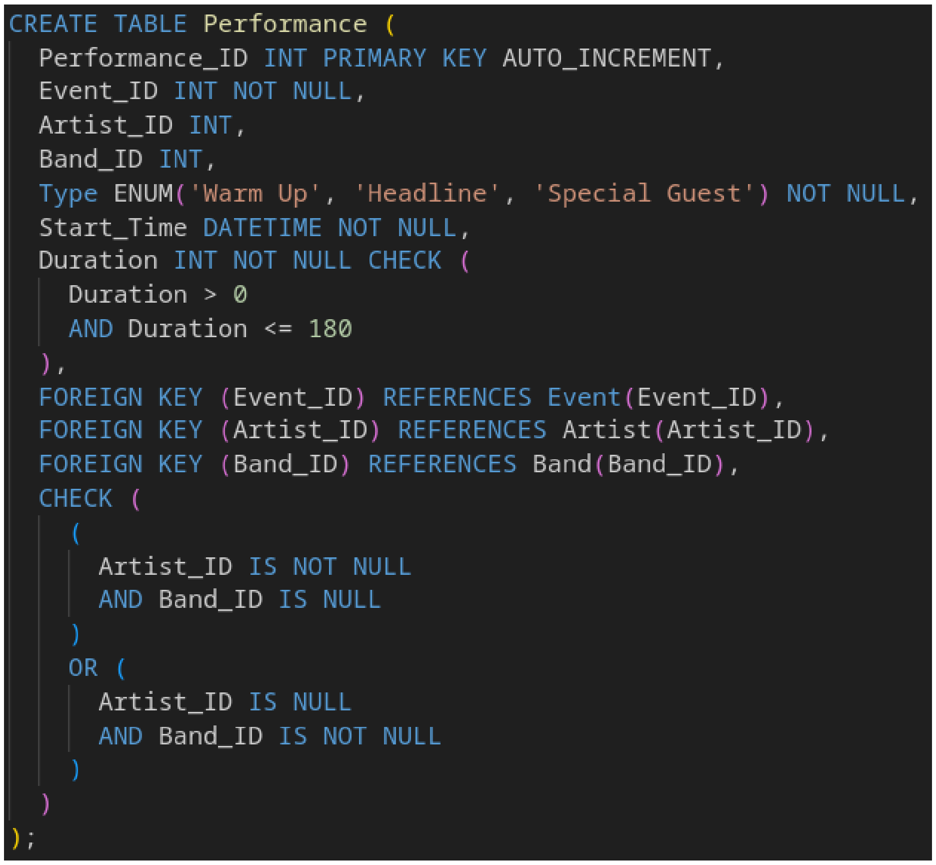
```sql
CREATE TABLE Band (
  Band_ID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  Formation_Date DATE,
  Website VARCHAR(255),
  Instagram_Profile VARCHAR(255)
);
```

<u>Δημιουργία πίνακα Μέλος Μπάντας:</u>



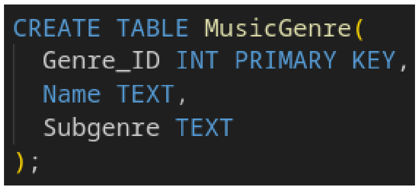
```sql
CREATE TABLE Band_Member (
  Band_ID INT NOT NULL,
  Artist_ID INT NOT NULL,
  PRIMARY KEY (Band_ID, Artist_ID),
  FOREIGN KEY (Band_ID) REFERENCES Band(Band_ID),
  FOREIGN KEY (Artist_ID) REFERENCES Artist(Artist_ID)
);
```

<u>Δημιουργία πίνακα Εμφάνισης:</u>



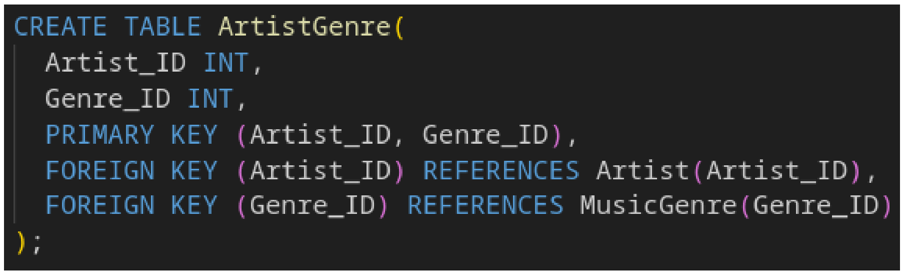
```sql
CREATE TABLE Performance (
  Performance_ID INT PRIMARY KEY AUTO_INCREMENT,
  Event_ID INT NOT NULL,
  Artist_ID INT,
  Band_ID INT,
  Type ENUM('Warm Up', 'Headline', 'Special Guest') NOT NULL,
  Start_Time DATETIME NOT NULL,
  Duration INT NOT NULL CHECK (
    Duration > 0
    AND Duration <= 180
  ),
  FOREIGN KEY (Event_ID) REFERENCES Event(Event_ID),
  FOREIGN KEY (Artist_ID) REFERENCES Artist(Artist_ID),
  FOREIGN KEY (Band_ID) REFERENCES Band(Band_ID),
  CHECK (
    (
      Artist_ID IS NOT NULL
      AND Band_ID IS NULL
    )
    OR (
      Artist_ID IS NULL
      AND Band_ID IS NOT NULL
    )
  )
);
```

<u>Δημιουργία πίνακα Μουσικού Είδους:</u>



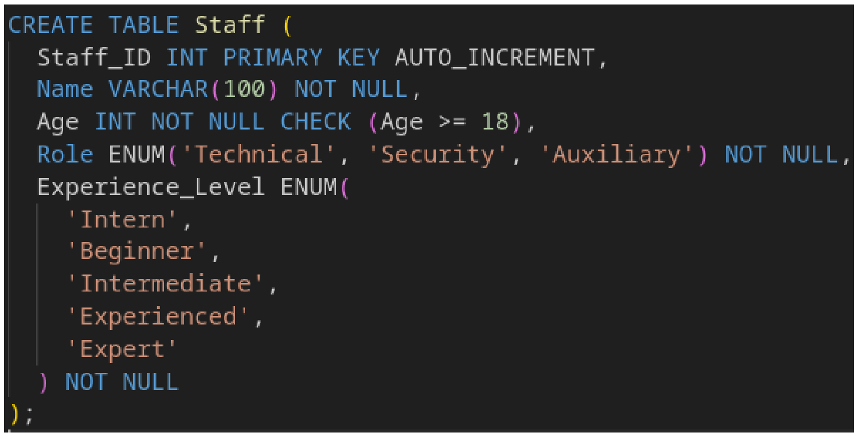
```sql
CREATE TABLE MusicGenre(
  Genre_ID INT PRIMARY KEY,
  Name TEXT,
  Subgenre TEXT
);
```

<u>Δημιουργία πίνακα Είδος Καλλιτέχνη:</u>



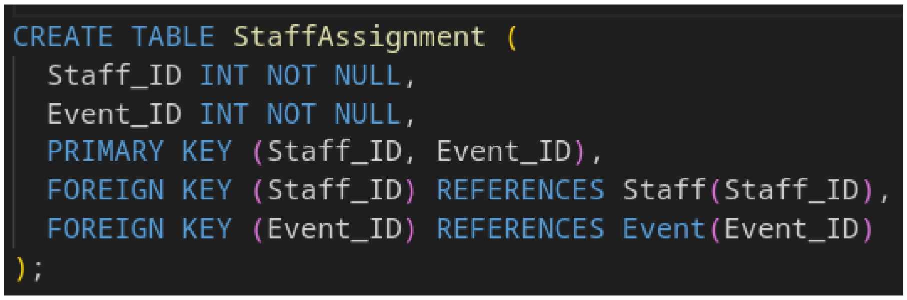
```sql
CREATE TABLE ArtistGenre(
  Artist_ID INT,
  Genre_ID INT,
  PRIMARY KEY (Artist_ID, Genre_ID),
  FOREIGN KEY (Artist_ID) REFERENCES Artist(Artist_ID),
  FOREIGN KEY (Genre_ID) REFERENCES MusicGenre(Genre_ID)
);
```

<u>Δημιουργία πίνακα Προσωπικού:</u>



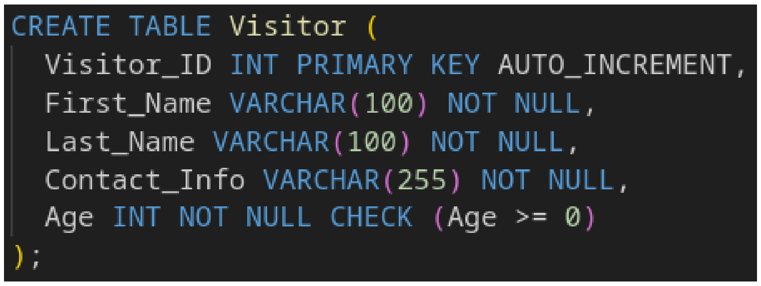
```sql
CREATE TABLE Staff (
  Staff_ID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  Age INT NOT NULL CHECK (Age >= 18),
  Role ENUM('Technical', 'Security', 'Auxiliary') NOT NULL,
  Experience_Level ENUM(
    'Intern',
    'Beginner',
    'Intermediate',
    'Experienced',
    'Expert'
  ) NOT NULL
);
```

<u>Δημιουργία πίνακα Ανάθεσης Προσωπικού:</u>



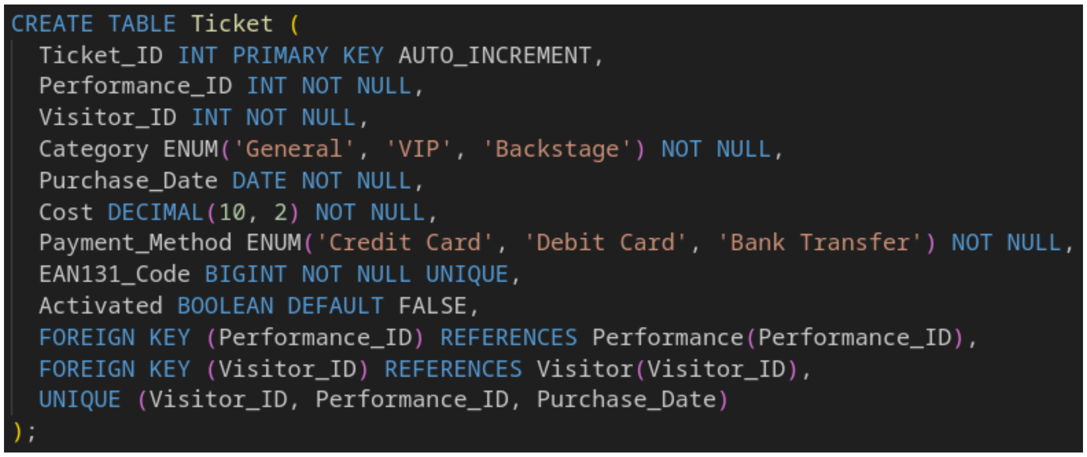
```sql
CREATE TABLE StaffAssignment (
  Staff_ID INT NOT NULL,
  Event_ID INT NOT NULL,
  PRIMARY KEY (Staff_ID, Event_ID),
  FOREIGN KEY (Staff_ID) REFERENCES Staff(Staff_ID),
  FOREIGN KEY (Event_ID) REFERENCES Event(Event_ID)
);
```

<u>Δημιουργία πίνακα Επισκέπτη:</u>



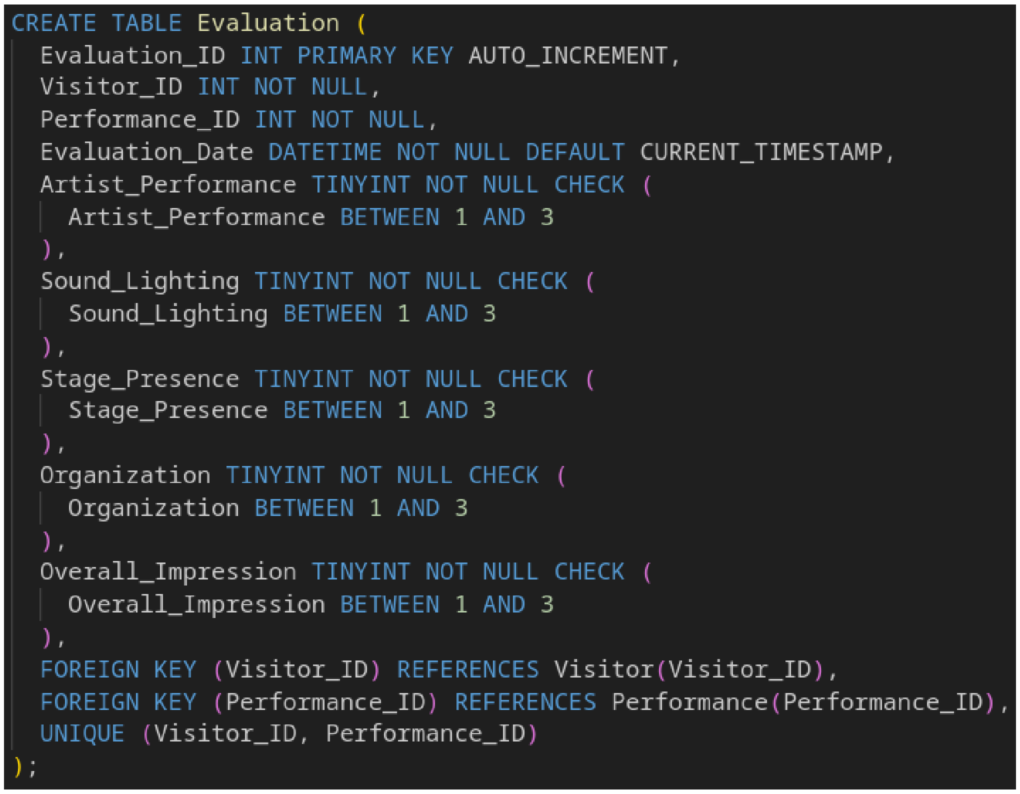
```sql
CREATE TABLE Visitor (
  Visitor_ID INT PRIMARY KEY AUTO_INCREMENT,
  First_Name VARCHAR(100) NOT NULL,
  Last_Name VARCHAR(100) NOT NULL,
  Contact_Info VARCHAR(255) NOT NULL,
  Age INT NOT NULL CHECK (Age >= 0)
);
```

<u>Δημιουργία πίνακα Εισιτηρίου:</u>



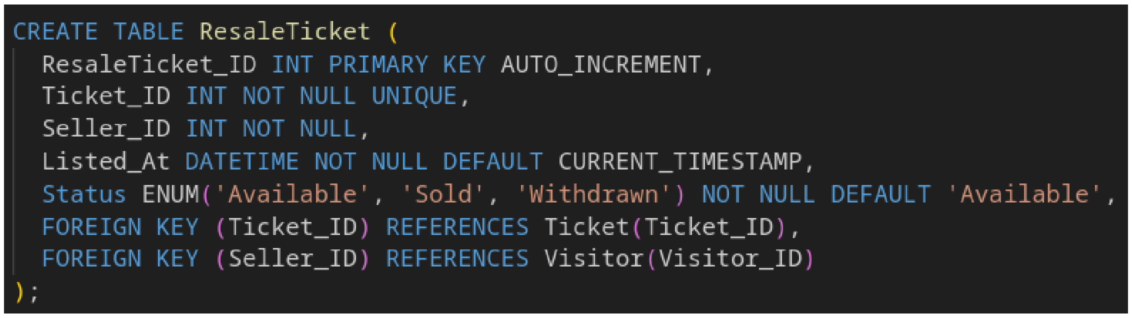
```sql
CREATE TABLE Ticket (
    Ticket_ID INT PRIMARY KEY AUTO_INCREMENT,
    Performance_ID INT NOT NULL,
    Visitor_ID INT NOT NULL,
    Category ENUM('General', 'VIP', 'Backstage') NOT NULL,
    Purchase_Date DATE NOT NULL,
    Cost DECIMAL(10, 2) NOT NULL,
    Payment_Method ENUM('Credit Card', 'Debit Card', 'Bank Transfer') NOT NULL,
    EAN131_Code BIGINT NOT NULL UNIQUE,
    Activated BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (Performance_ID) REFERENCES Performance(Performance_ID),
    FOREIGN KEY (Visitor_ID) REFERENCES Visitor(Visitor_ID),
    UNIQUE (Visitor_ID, Performance_ID, Purchase_Date)
);
```

<u>Δημιουργία πίνακα Αξιολόγησης:</u>



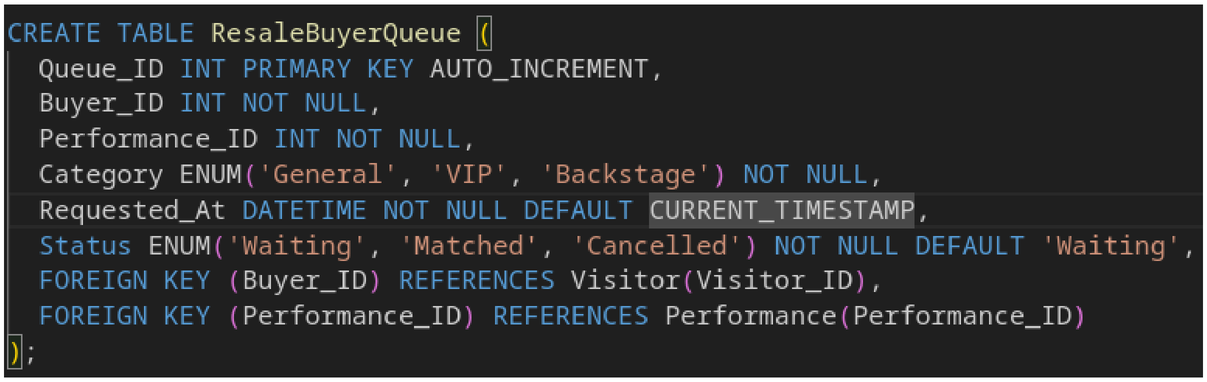
```sql
CREATE TABLE Evaluation (
    Evaluation_ID INT PRIMARY KEY AUTO_INCREMENT,
    Visitor_ID INT NOT NULL,
    Performance_ID INT NOT NULL,
    Evaluation_Date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Artist_Performance TINYINT NOT NULL CHECK (
        Artist_Performance BETWEEN 1 AND 3
    ),
    Sound_Lighting TINYINT NOT NULL CHECK (
        Sound_Lighting BETWEEN 1 AND 3
    ),
    Stage_Presence TINYINT NOT NULL CHECK (
        Stage_Presence BETWEEN 1 AND 3
    ),
    Organization TINYINT NOT NULL CHECK (
        Organization BETWEEN 1 AND 3
    ),
    Overall_Impression TINYINT NOT NULL CHECK (
        Overall_Impression BETWEEN 1 AND 3
    ),
    FOREIGN KEY (Visitor_ID) REFERENCES Visitor(Visitor_ID),
    FOREIGN KEY (Performance_ID) REFERENCES Performance(Performance_ID),
    UNIQUE (Visitor_ID, Performance_ID)
);
```

Δημιουργία πίνακα Μεταπώλησης Εισιτηρίων:



```sql
CREATE TABLE ResaleTicket (
  ResaleTicket_ID INT PRIMARY KEY AUTO_INCREMENT,
  Ticket_ID INT NOT NULL UNIQUE,
  Seller_ID INT NOT NULL,
  Listed_At DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  Status ENUM('Available', 'Sold', 'Withdrawn') NOT NULL DEFAULT 'Available',
  FOREIGN KEY (Ticket_ID) REFERENCES Ticket(Ticket_ID),
  FOREIGN KEY (Seller_ID) REFERENCES Visitor(Visitor_ID)
);
```

Δημιουργία πίνακα Ουράς Αγοραστών σε αναμονή:



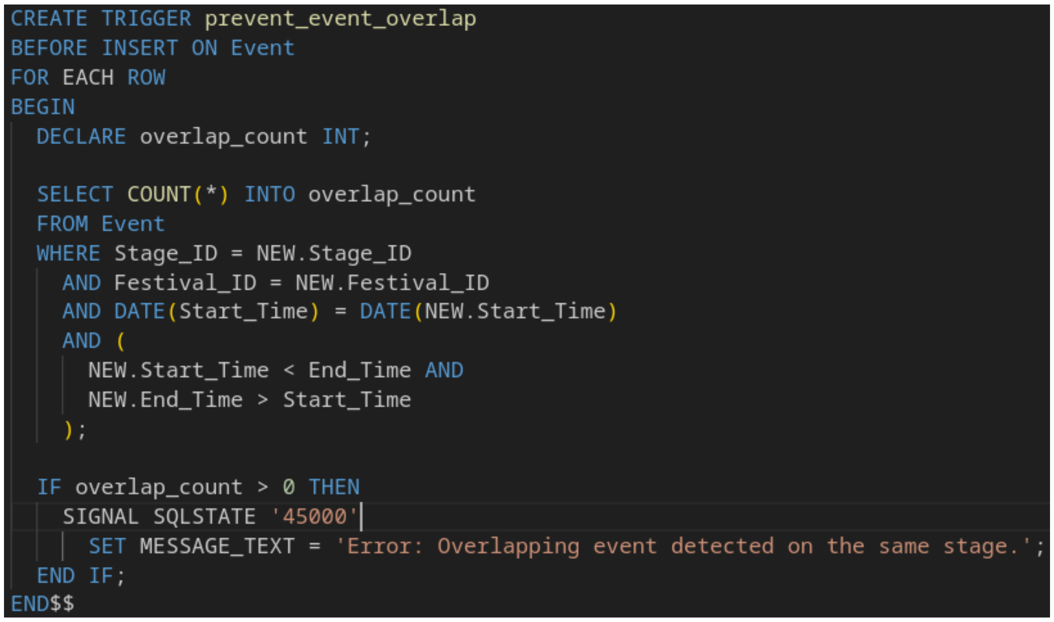
```sql
CREATE TABLE ResaleBuyerQueue (
  Queue_ID INT PRIMARY KEY AUTO_INCREMENT,
  Buyer_ID INT NOT NULL,
  Performance_ID INT NOT NULL,
  Category ENUM('General', 'VIP', 'Backstage') NOT NULL,
  Requested_At DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  Status ENUM('Waiting', 'Matched', 'Cancelled') NOT NULL DEFAULT 'Waiting',
  FOREIGN KEY (Buyer_ID) REFERENCES Visitor(Visitor_ID),
  FOREIGN KEY (Performance_ID) REFERENCES Performance(Performance_ID)
);
```

Triggers και Procedure Μεταπώλησης.

Στην βάση χρησιμοποιούμε περιορισμοί στηλών (check constraints), πεδίου τιμών (domain constraints), αναφορικής ακεραιότητας (foreign key constraints), ακεραιότητας οντότητας (primary key) και κάποιοι άλλοι (9 triggers). Τα triggers τα ελέγχουμε και στην εισαγωγή αλλά και στην ενημέρωση των records.

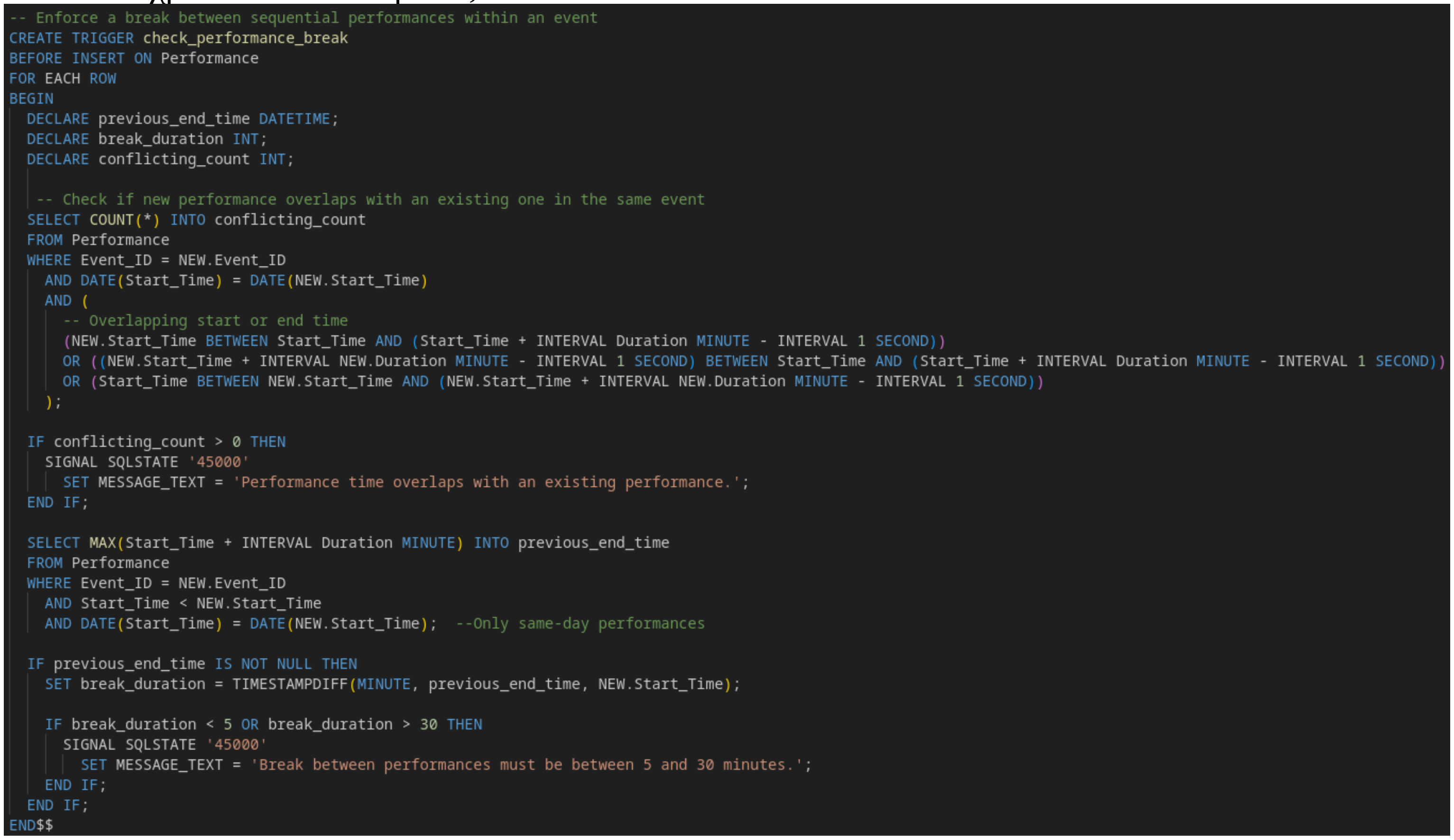
Τα triggers έχουν ως εξής:

1: Αποφυγή να υπάρξει event overlap στην ίδια σκηνή και στο ίδιο festival:



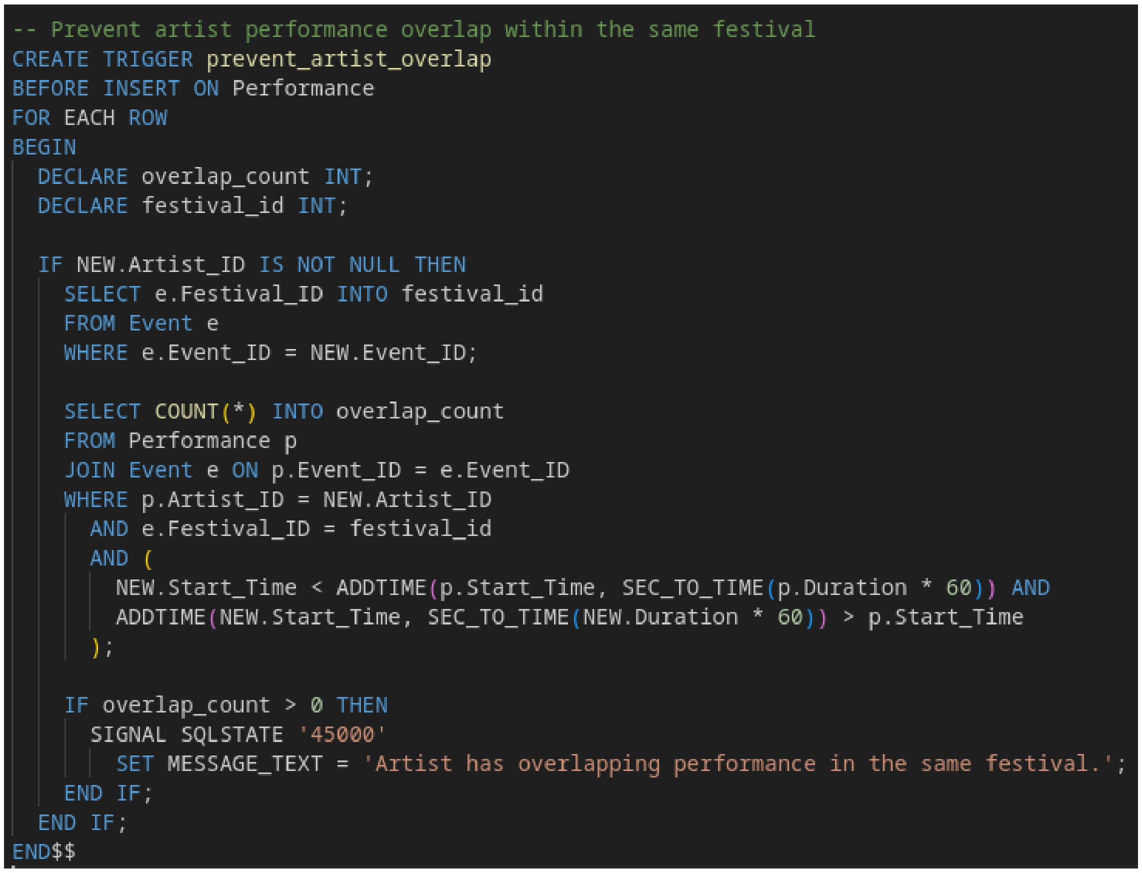
```sql
CREATE TRIGGER prevent_event_overlap
BEFORE INSERT ON Event
FOR EACH ROW
BEGIN
  DECLARE overlap_count INT;

  SELECT COUNT(*) INTO overlap_count
  FROM Event
  WHERE Stage_ID = NEW.Stage_ID
    AND Festival_ID = NEW.Festival_ID
    AND DATE(Start_Time) = DATE(NEW.Start_Time)
    AND (
      NEW.Start_Time < End_Time AND
      NEW.End_Time > Start_Time
    );

  IF overlap_count > 0 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Error: Overlapping event detected on the same stage.';
  END IF;
END$$
```

2) Αποφυγή να επικαλύπτονται εμφανίσεις στην ίδια παράσταση και ταυτόχρονα έλεγχο υποχρεωτικού break μεταξύ 5 και 30 λεπτών:



```sql
-- Enforce a break between sequential performances within an event
CREATE TRIGGER check_performance_break
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
  DECLARE previous_end_time DATETIME;
  DECLARE break_duration INT;
  DECLARE conflicting_count INT;

   -- Check if new performance overlaps with an existing one in the same event
  SELECT COUNT(*) INTO conflicting_count
  FROM Performance
  WHERE Event_ID = NEW.Event_ID
    AND DATE(Start_Time) = DATE(NEW.Start_Time)
    AND (
      -- Overlapping start or end time
      (NEW.Start_Time BETWEEN Start_Time AND (Start_Time + INTERVAL Duration MINUTE - INTERVAL 1 SECOND))
      OR ((NEW.Start_Time + INTERVAL NEW.Duration MINUTE - INTERVAL 1 SECOND) BETWEEN Start_Time AND (Start_Time + INTERVAL Duration MINUTE - INTERVAL 1 SECOND))
      OR (Start_Time BETWEEN NEW.Start_Time AND (NEW.Start_Time + INTERVAL NEW.Duration MINUTE - INTERVAL 1 SECOND))
    );

  IF conflicting_count > 0 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Performance time overlaps with an existing performance.';
  END IF;

  SELECT MAX(Start_Time + INTERVAL Duration MINUTE) INTO previous_end_time
  FROM Performance
  WHERE Event_ID = NEW.Event_ID
    AND Start_Time < NEW.Start_Time
    AND DATE(Start_Time) = DATE(NEW.Start_Time);  --Only same-day performances

  IF previous_end_time IS NOT NULL THEN
    SET break_duration = TIMESTAMPDIFF(MINUTE, previous_end_time, NEW.Start_Time);

    IF break_duration < 5 OR break_duration > 30 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Break between performances must be between 5 and 30 minutes.';
    END IF;
  END IF;
END$$
```

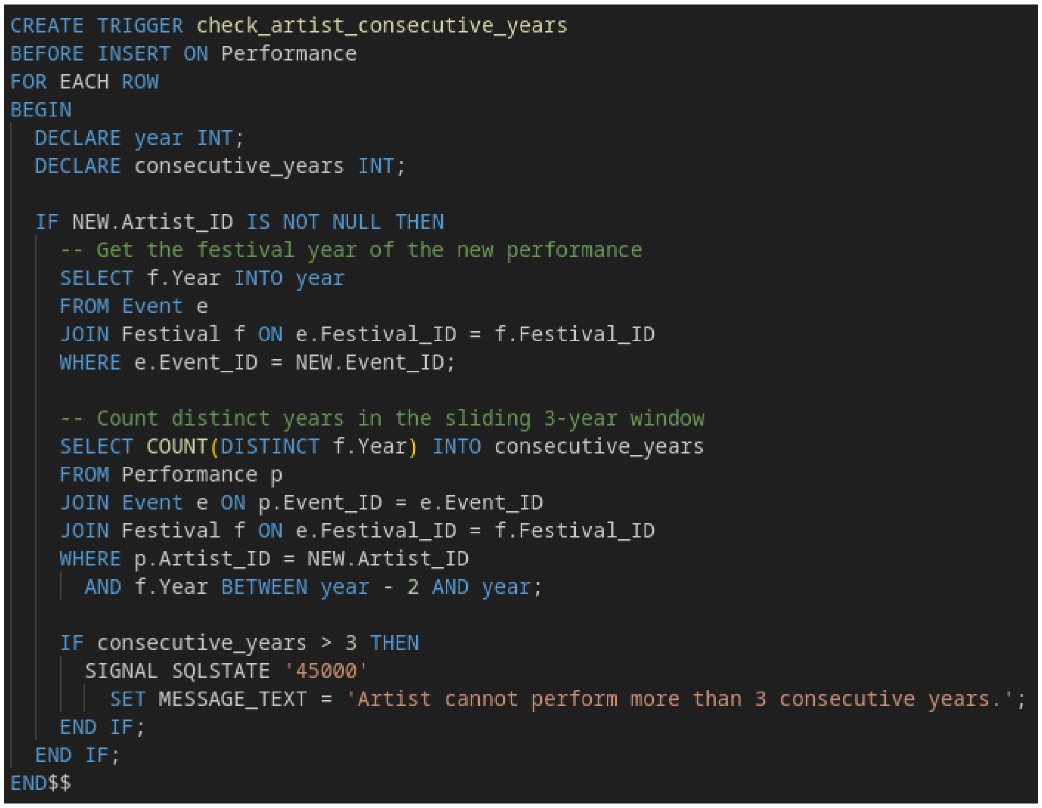
3) Αποφυγή καλλιτέχνη να έχει επικάλυψη σε εμφανίσεις στο ίδιο φεστιβάλ (could be possible with a virtual artist concept!!):



```sql
-- Prevent artist performance overlap within the same festival
CREATE TRIGGER prevent_artist_overlap
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
  DECLARE overlap_count INT;
  DECLARE festival_id INT;

  IF NEW.Artist_ID IS NOT NULL THEN
    SELECT e.Festival_ID INTO festival_id
    FROM Event e
    WHERE e.Event_ID = NEW.Event_ID;

    SELECT COUNT(*) INTO overlap_count
    FROM Performance p
    JOIN Event e ON p.Event_ID = e.Event_ID
    WHERE p.Artist_ID = NEW.Artist_ID
      AND e.Festival_ID = festival_id
      AND (
        NEW.Start_Time < ADDTIME(p.Start_Time, SEC_TO_TIME(p.Duration * 60)) AND
        ADDTIME(NEW.Start_Time, SEC_TO_TIME(NEW.Duration * 60)) > p.Start_Time
      );

    IF overlap_count > 0 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist has overlapping performance in the same festival.';
    END IF;
  END IF;
END$$
```

4) Το ίδιο αλλά για τις μπάντες (βλ. Line 403: sql/install.sql)

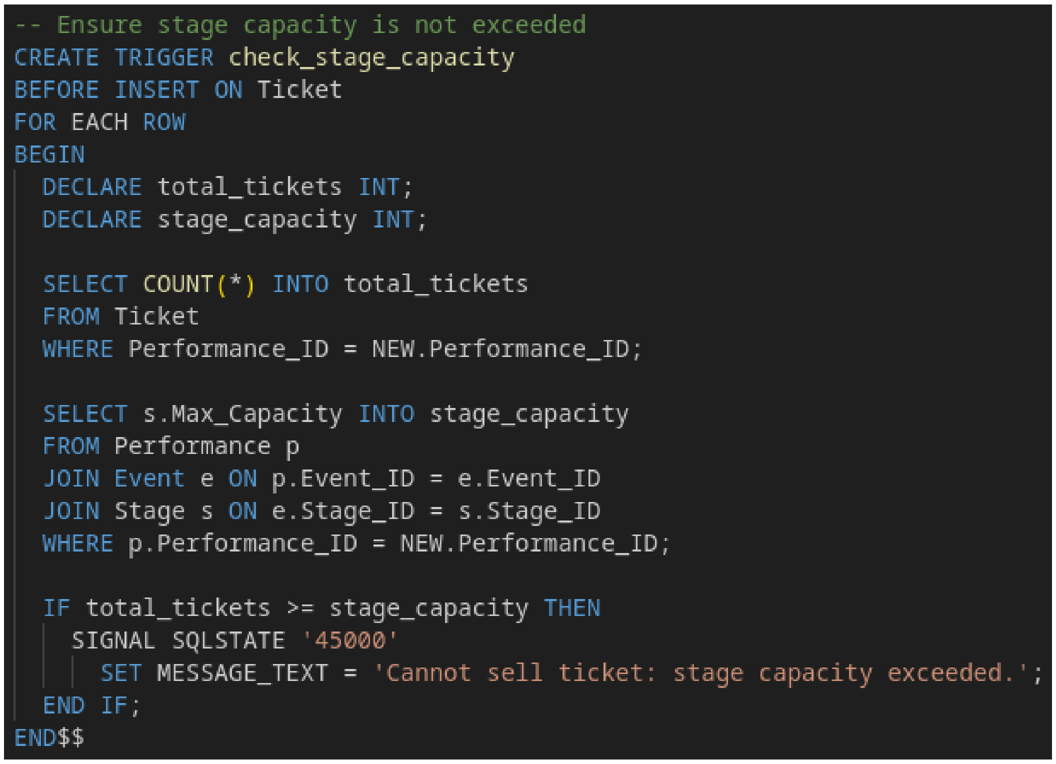
5) Περιορισμός καλλιτέχνη να μην μπορεί να συμμετέχει πάνω από 3 συνεχόμενες χρονιές:



```sql
CREATE TRIGGER check_artist_consecutive_years
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
  DECLARE year INT;
  DECLARE consecutive_years INT;

  IF NEW.Artist_ID IS NOT NULL THEN
    -- Get the festival year of the new performance
    SELECT f.Year INTO year
    FROM Event e
    JOIN Festival f ON e.Festival_ID = f.Festival_ID
    WHERE e.Event_ID = NEW.Event_ID;

    -- Count distinct years in the sliding 3-year window
    SELECT COUNT(DISTINCT f.Year) INTO consecutive_years
    FROM Performance p
    JOIN Event e ON p.Event_ID = e.Event_ID
    JOIN Festival f ON e.Festival_ID = f.Festival_ID
    WHERE p.Artist_ID = NEW.Artist_ID
      AND f.Year BETWEEN year - 2 AND year;

    IF consecutive_years > 3 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist cannot perform more than 3 consecutive years.';
    END IF;
  END IF;
END$$
```

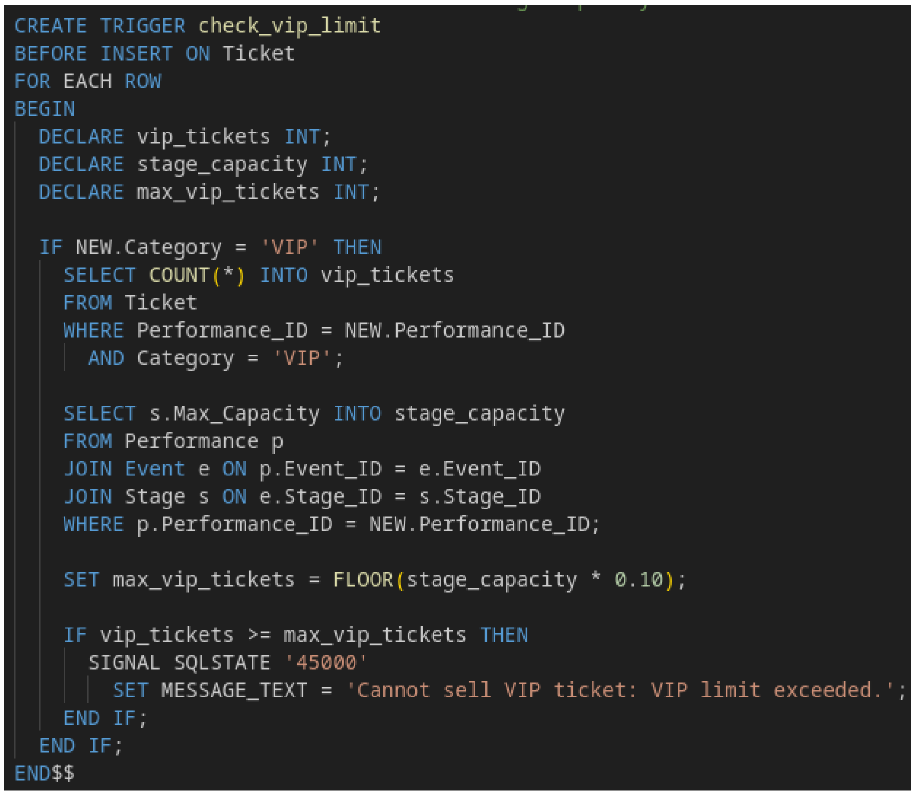
6) Το ίδιο για τις μπάντες.

7) Περιορισμός αγορασμένων εισιτηρίων με βάση τη χωριτικότητα σκηνής:



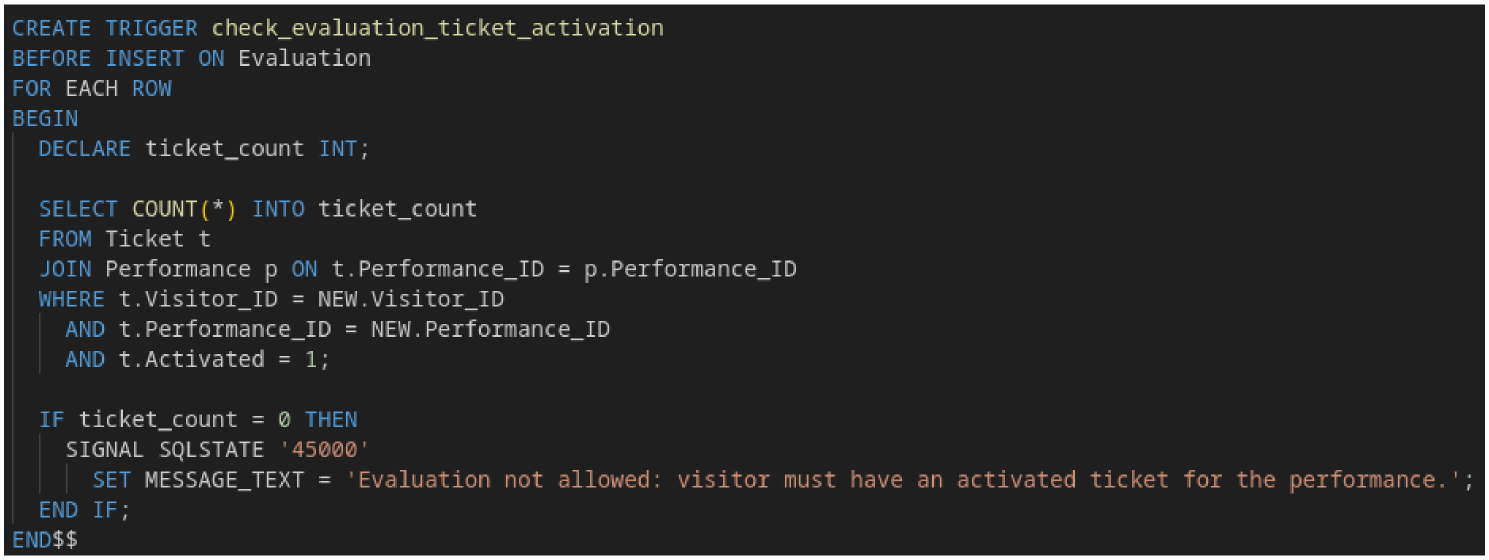
```sql
-- Ensure stage capacity is not exceeded
CREATE TRIGGER check_stage_capacity
BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
  DECLARE total_tickets INT;
  DECLARE stage_capacity INT;

  SELECT COUNT(*) INTO total_tickets
  FROM Ticket
  WHERE Performance_ID = NEW.Performance_ID;

  SELECT s.Max_Capacity INTO stage_capacity
  FROM Performance p
  JOIN Event e ON p.Event_ID = e.Event_ID
  JOIN Stage s ON e.Stage_ID = s.Stage_ID
  WHERE p.Performance_ID = NEW.Performance_ID;

  IF total_tickets >= stage_capacity THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Cannot sell ticket: stage capacity exceeded.';
  END IF;
END$$
```

8) Το όριο VIP εισιτηρίων να είναι το 10% από το όριο της σκηνής:



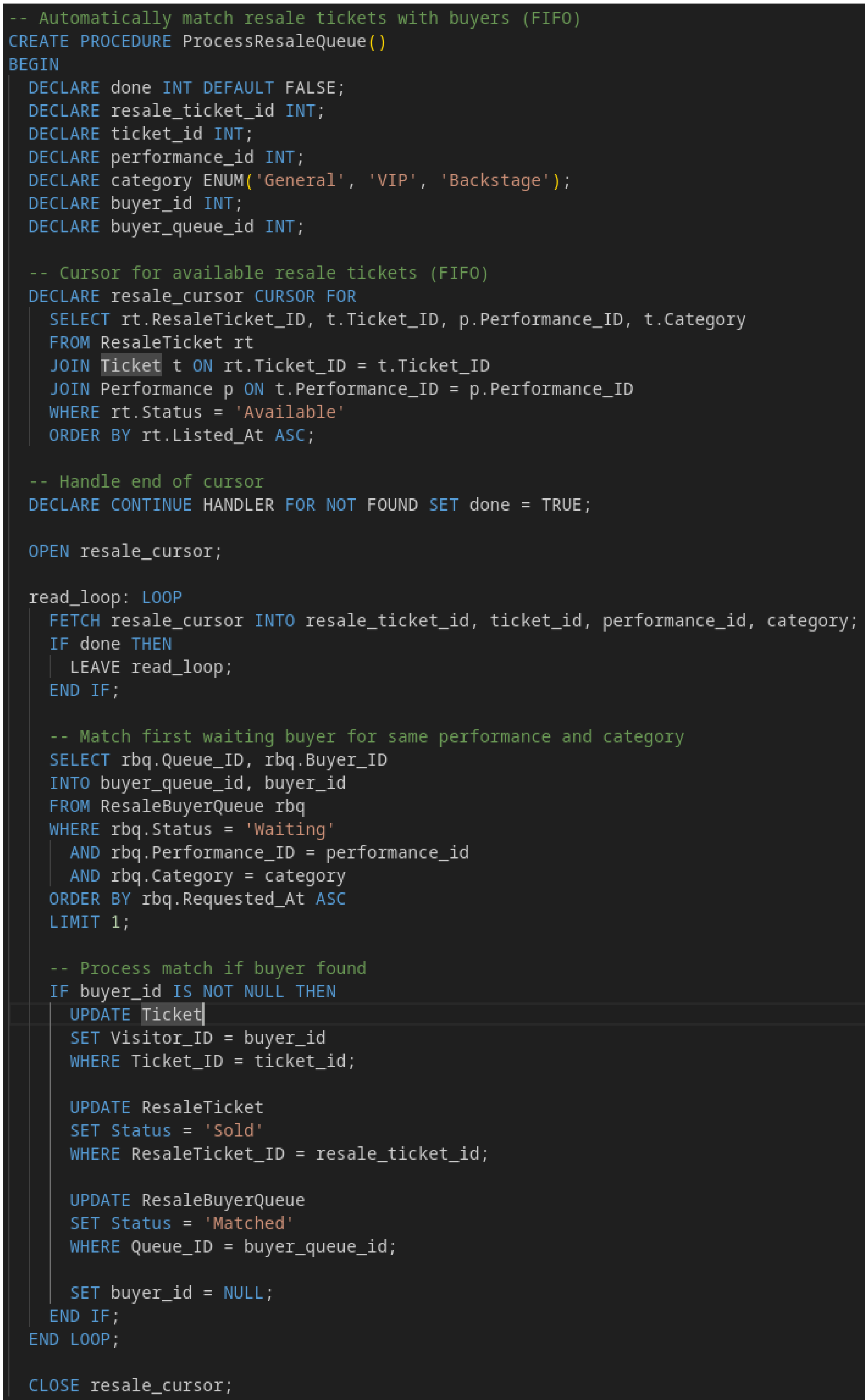
```sql
CREATE TRIGGER check_vip_limit
BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
  DECLARE vip_tickets INT;
  DECLARE stage_capacity INT;
  DECLARE max_vip_tickets INT;

  IF NEW.Category = 'VIP' THEN
    SELECT COUNT(*) INTO vip_tickets
    FROM Ticket
    WHERE Performance_ID = NEW.Performance_ID
      AND Category = 'VIP';

    SELECT s.Max_Capacity INTO stage_capacity
    FROM Performance p
    JOIN Event e ON p.Event_ID = e.Event_ID
    JOIN Stage s ON e.Stage_ID = s.Stage_ID
    WHERE p.Performance_ID = NEW.Performance_ID;

    SET max_vip_tickets = FLOOR(stage_capacity * 0.10);

    IF vip_tickets >= max_vip_tickets THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot sell VIP ticket: VIP limit exceeded.';
    END IF;
  END IF;
END$$
```

9) Εισαγωγή αξιολόγησης μόνο εάν ο επισκέπτης έχει ενεργοποιημένο εισιτήριο:



```sql
CREATE TRIGGER check_evaluation_ticket_activation
BEFORE INSERT ON Evaluation
FOR EACH ROW
BEGIN
  DECLARE ticket_count INT;

  SELECT COUNT(*) INTO ticket_count
  FROM Ticket t
  JOIN Performance p ON t.Performance_ID = p.Performance_ID
  WHERE t.Visitor_ID = NEW.Visitor_ID
    AND t.Performance_ID = NEW.Performance_ID
    AND t.Activated = 1;

  IF ticket_count = 0 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Evaluation not allowed: visitor must have an activated ticket for the performance.';
  END IF;
END$$
```

Τέλος έχουμε και ένα procedure που κάθε φορά που το τρέχουμε κάνει match τα εισιτήρια που βρίσκονται στη μεταπώληση, με το ticket_resale_buyer_queue με λογική FIFO



```
-- Automatically match resale tickets with buyers (FIFO)
CREATE PROCEDURE ProcessResaleQueue()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE resale_ticket_id INT;
  DECLARE ticket_id INT;
  DECLARE performance_id INT;
  DECLARE category ENUM('General', 'VIP', 'Backstage');
  DECLARE buyer_id INT;
  DECLARE buyer_queue_id INT;

  -- Cursor for available resale tickets (FIFO)
  DECLARE resale_cursor CURSOR FOR
    SELECT rt.ResaleTicket_ID, t.Ticket_ID, p.Performance_ID, t.Category
    FROM ResaleTicket rt
    JOIN Ticket t ON rt.Ticket_ID = t.Ticket_ID
    JOIN Performance p ON t.Performance_ID = p.Performance_ID
    WHERE rt.Status = 'Available'
    ORDER BY rt.Listed_At ASC;

  -- Handle end of cursor
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN resale_cursor;

  read_loop: LOOP
    FETCH resale_cursor INTO resale_ticket_id, ticket_id, performance_id, category;
    IF done THEN
      LEAVE read_loop;
    END IF;

    -- Match first waiting buyer for same performance and category
    SELECT rbq.Queue_ID, rbq.Buyer_ID
    INTO buyer_queue_id, buyer_id
    FROM ResaleBuyerQueue rbq
    WHERE rbq.Status = 'Waiting'
      AND rbq.Performance_ID = performance_id
      AND rbq.Category = category
    ORDER BY rbq.Requested_At ASC
    LIMIT 1;

    -- Process match if buyer found
    IF buyer_id IS NOT NULL THEN
      UPDATE Ticket
      SET Visitor_ID = buyer_id
      WHERE Ticket_ID = ticket_id;

      UPDATE ResaleTicket
      SET Status = 'Sold'
      WHERE ResaleTicket_ID = resale_ticket_id;

      UPDATE ResaleBuyerQueue
      SET Status = 'Matched'
      WHERE Queue_ID = buyer_queue_id;

      SET buyer_id = NULL;
    END IF;
  END LOOP;

  CLOSE resale_cursor;
```

Queries

1)

```
SELECT
    f.Year,
    t.Payment_Method,
    SUM(t.Cost) AS Total_Revenue
FROM Ticket t
JOIN Performance p ON t.Performance_ID = p.Performance_ID
JOIN Event e ON p.Event_ID = e.Event_ID
JOIN Festival f ON e.Festival_ID = f.Festival_ID
GROUP BY f.Year, t.Payment_Method
ORDER BY f.Year, t.Payment_Method;
```

2)

```
SELECT
    a.Artist_ID,
    a.Stage_Name,
    g.Name AS Genre_Name,
    f.Year AS Participated_Year
FROM Artist a
JOIN ArtistGenre ag ON a.Artist_ID = ag.Artist_ID
JOIN MusicGenre g ON ag.Genre_ID = g.Genre_ID
LEFT JOIN Performance p ON a.Artist_ID = p.Artist_ID
LEFT JOIN Event e ON p.Event_ID = e.Event_ID
LEFT JOIN Festival f ON e.Festival_ID = f.Festival_ID
WHERE g.Name = 'Rock'
GROUP BY a.Artist_ID, f.Year;
```

3)

```
SELECT
    a.Artist_ID,
    a.Stage_Name,
    f.Year,
    COUNT(*) AS Warmup_Count
FROM Performance p
JOIN Artist a ON p.Artist_ID = a.Artist_ID
JOIN Event e ON p.Event_ID = e.Event_ID
JOIN Festival f ON e.Festival_ID = f.Festival_ID
WHERE p.Type = 'warm up'
GROUP BY a.Artist_ID, f.Year
HAVING COUNT(*) > 2;
```

4)
```
SELECT
    a.Artist_ID,
    a.Stage_Name,
    ROUND(AVG(e.Artist_Performance), 2) AS Avg_Artist_Performance,
    ROUND(AVG(e.Overall_Impression), 2) AS Avg_Overall_Impression
FROM Artist a
JOIN Performance p ON a.Artist_ID = p.Artist_ID
JOIN Evaluation e ON p.Performance_ID = e.Performance_ID
WHERE a.Stage_Name = 'Lady Gaga'
GROUP BY a.Artist_ID, a.Stage_Name
ORDER BY Avg_Artist_Performance DESC;
```

5)
```
SELECT
    a.Artist_ID,
    a.Stage_Name,
    TIMESTAMPDIFF(YEAR, a.Birthdate, CURDATE()) AS Age,
    COUNT(DISTINCT f.Festival_ID) AS Festival_Count
FROM Artist a
JOIN Performance p ON a.Artist_ID = p.Artist_ID
JOIN Event e ON p.Event_ID = e.Event_ID
JOIN Festival f ON e.Festival_ID = f.Festival_ID
WHERE TIMESTAMPDIFF(YEAR, a.Birthdate, CURDATE()) < 30
GROUP BY a.Artist_ID
ORDER BY Festival_Count DESC
LIMIT 10;
```

6)
```
SELECT
    v.First_Name,
    v.Last_Name,
    p.Performance_ID,
    p.Type,
    p.Start_Time,
    a.Stage_Name AS Artist_Name,
    ROUND(AVG(
        (e.Artist_Performance + e.Sound_Lighting + e.Stage_Presence + e.Organization +
e.Overall_Impression) / 5
    ), 2) AS Avg_Rating
FROM Visitor v
JOIN Evaluation e ON v.Visitor_ID = e.Visitor_ID
JOIN Performance p ON e.Performance_ID = p.Performance_ID
JOIN Artist a ON p.Artist_ID = a.Artist_ID
WHERE v.First_Name = 'Mary' AND v.Last_Name = 'Choi'
GROUP BY v.First_Name, v.Last_Name, p.Performance_ID, a.Stage_Name, p.Type,
p.Start_Time
ORDER BY p.Start_Time;
```

7)
```
SELECT
    F.Festival_ID,
    F.Year,
    AVG(
        CASE S.Experience_Level
            WHEN 'Intern' THEN 1
            WHEN 'Beginner' THEN 2
            WHEN 'Intermediate' THEN 3
            WHEN 'Experienced' THEN 4
            WHEN 'Expert' THEN 5
        END
    ) AS Avg_Experience_Score
FROM Festival F
JOIN Event E ON F.Festival_ID = E.Festival_ID
JOIN StaffAssignment SA ON E.Event_ID = SA.Event_ID
JOIN Staff S ON SA.Staff_ID = S.Staff_ID
WHERE S.Role = 'Technical'
GROUP BY F.Festival_ID, F.Year
ORDER BY Avg_Experience_Score ASC
LIMIT 1;
```

8)

```
SELECT
   s.Staff_ID,
   s.Name,
   s.Role,
   s.Experience_Level
FROM
   Staff s
WHERE NOT EXISTS (
   SELECT 1
   FROM
      StaffAssignment sa
   JOIN
      Event e ON sa.Event_ID = e.Event_ID
   WHERE
      sa.Staff_ID = s.Staff_ID
      AND '2022-10-10' BETWEEN DATE(e.Start_Time) AND DATE(e.End_Time)
)
AND s.Role = 'Auxiliary';
```

9)

```
WITH VisitorYearlyEventAttendance AS (
   SELECT
      t.Visitor_ID,
      YEAR(e.Start_Time) AS AttendanceYear,
      COUNT(DISTINCT p.Event_ID) AS EventAttendanceCount
   FROM
      Ticket t
   JOIN
      Performance p ON t.Performance_ID = p.Performance_ID
   JOIN
      Event e ON p.Event_ID = e.Event_ID
   GROUP BY
      t.Visitor_ID,
      YEAR(e.Start_Time)
   HAVING
      COUNT(DISTINCT p.Event_ID) > 3
),
RankedEventAttendance AS (
   SELECT
      Visitor_ID,
      AttendanceYear,
      EventAttendanceCount,
      COUNT(*) OVER (PARTITION BY AttendanceYear, EventAttendanceCount) AS
GroupSize
   FROM
      VisitorYearlyEventAttendance
)

SELECT
   CONCAT(v.First_Name, ' ', v.Last_Name) AS VisitorName,
   rea.AttendanceYear,
   rea.EventAttendanceCount
FROM
   RankedEventAttendance rea
JOIN
   Visitor v ON rea.Visitor_ID = v.Visitor_ID
WHERE
   rea.GroupSize > 1
ORDER BY
   rea.AttendanceYear,
   rea.EventAttendanceCount,
   VisitorName;
```

10)

```
SELECT
   mg1.Name AS Genre1,
   mg2.Name AS Genre2,
   COUNT(DISTINCT ag1.Artist_ID) AS PairCount
FROM
   ArtistGenre ag1
INNER JOIN
   ArtistGenre ag2 ON ag1.Artist_ID = ag2.Artist_ID AND ag1.Genre_ID <
ag2.Genre_ID
INNER JOIN
   MusicGenre mg1 ON ag1.Genre_ID = mg1.Genre_ID
INNER JOIN
   MusicGenre mg2 ON ag2.Genre_ID = mg2.Genre_ID
WHERE
    Artist_ID)
    EXISTS (SELECT 1 FROM Performance p WHERE p.Artist_ID = ag1.Artist_ID)
GROUP BY
    mg1.Genre_ID, mg2.Genre_ID, Genre1, Genre2
ORDER BY
    PairCount DESC
LIMIT 3;
```

11)

```
WITH ArtistPerformanceCounts AS (
    SELECT
        p.Artist_ID,
        COUNT(p.Performance_ID) AS AppearanceCount
    FROM
        Performance p
    WHERE
        p.Artist_ID IS NOT NULL
    GROUP BY
        p.Artist_ID
),
MaxAppearances AS (
    SELECT MAX(AppearanceCount) AS MaxCount
    FROM ArtistPerformanceCounts
)
SELECT
    a.Artist_ID,
    a.Stage_Name,
    apc.AppearanceCount
FROM
    ArtistPerformanceCounts apc
JOIN
    Artist a ON apc.Artist_ID = a.Artist_ID
CROSS JOIN
    MaxAppearances ma
WHERE
    apc.AppearanceCount <= (ma.MaxCount - 5)
ORDER BY
    apc.AppearanceCount ASC;
```

12)
```
SELECT
    f.Year AS FestivalYear,
    DATE(e.Start_Time) AS AssignmentDate,
    s.Role AS StaffCategory,
    COUNT(DISTINCT sa.Staff_ID) AS RequiredStaffCount
FROM
    StaffAssignment sa
JOIN
    Staff s ON sa.Staff_ID = s.Staff_ID
JOIN
    Event e ON sa.Event_ID = e.Event_ID
JOIN
    Festival f ON e.Festival_ID = f.Festival_ID
GROUP BY
    f.Year,
    DATE(e.Start_Time),
    s.Role
ORDER BY
    FestivalYear ASC,
    AssignmentDate ASC,
    StaffCategory ASC;
```

13)
```
SELECT
    a.Artist_ID,
    a.Stage_Name,
    COUNT(DISTINCT loc.Continent) AS DistinctContinents
FROM
    Artist a
JOIN
    Performance p ON a.Artist_ID = p.Artist_ID
JOIN
    Event e ON p.Event_ID = e.Event_ID        t
JOIN
    Festival f ON e.Festival_ID = f.Festival_ID
JOIN
    Location loc ON f.Location_ID = loc.Location_ID
WHERE
    p.Artist_ID IS NOT NULL
    AND loc.Continent IS NOT NULL AND loc.Continent != ''
GROUP BY
```

```
    a.Artist_ID, a.Stage_Name
HAVING
    COUNT(DISTINCT loc.Continent) >= 3
ORDER BY
    DistinctContinents DESC,
    a.Stage_Name ASC;
```

14)

```
WITH GenreYearlyPerformanceCounts AS (
    SELECT
        ag.Genre_ID,
        mg.Name AS GenreName,
        YEAR(p.Start_Time) AS PerformanceYear,
        COUNT(p.Performance_ID) AS PerformanceCount
    FROM
        Performance p
    JOIN
        ArtistGenre ag ON p.Artist_ID = ag.Artist_ID
    JOIN
        MusicGenre mg ON ag.Genre_ID = mg.Genre_ID
    WHERE
        p.Artist_ID IS NOT NULL
    GROUP BY
        ag.Genre_ID, mg.Name, YEAR(p.Start_Time)
    HAVING
        COUNT(p.Performance_ID) >= 3
)
SELECT
    gypc1.GenreName,
    gypc1.PerformanceYear AS Year1,
    gypc2.PerformanceYear AS Year2,
    gypc1.PerformanceCount
FROM
    GenreYearlyPerformanceCounts gypc1
JOIN
    GenreYearlyPerformanceCounts gypc2
    ON gypc1.Genre_ID = gypc2.Genre_ID
    AND gypc1.PerformanceYear = gypc2.PerformanceYear - 1
    AND gypc1.PerformanceCount = gypc2.PerformanceCount
ORDER BY
    gypc1.GenreName,
    Year1;
```

15)

```
SELECT
    CONCAT(v.First_Name, ' ', v.Last_Name) AS VisitorName,
    a.Stage_Name AS ArtistName,
    SUM(e.Overall_Impression) AS TotalOverallRatingToArtist
FROM
    Evaluation e
JOIN
    Visitor v ON e.Visitor_ID = v.Visitor_ID
JOIN
    Performance p ON e.Performance_ID = p.Performance_ID
JOIN
    Artist a ON p.Artist_ID = a.Artist_ID
WHERE
    p.Artist_ID IS NOT NULL
GROUP BY
    v.Visitor_ID, a.Artist_ID, VisitorName, ArtistName
ORDER BY
    TotalOverallRatingToArtist DESC
LIMIT 5;
```