```sql
-- Disable foreign key checks before dropping tables
SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS Continent;
DROP TABLE IF EXISTS Location;
DROP TABLE IF EXISTS Festival;
DROP TABLE IF EXISTS Stage;
DROP TABLE IF EXISTS Event;
DROP TABLE IF EXISTS Staff_role;
DROP TABLE IF EXISTS Experience_level;
DROP TABLE IF EXISTS Staff;
DROP TABLE IF EXISTS Artist;
DROP TABLE IF EXISTS Band;
DROP TABLE IF EXISTS Performs;
DROP TABLE IF EXISTS Performance;
DROP TABLE IF EXISTS Payment_method;
DROP TABLE IF EXISTS Visitor;
DROP TABLE IF EXISTS Ticket_type;
DROP TABLE IF EXISTS Ticket;
DROP TABLE IF EXISTS Rating;
DROP TABLE IF EXISTS Resale_Buyer;
DROP TABLE IF EXISTS Resale_Queue;
DROP TABLE IF EXISTS Works_on;
DROP TABLE IF EXISTS Demand_Queue;
DROP TABLE IF EXISTS Buys_specific_ticket;
DROP TABLE IF EXISTS Belongs_to;
DROP TABLE IF EXISTS Resale_Log;
DROP TABLE IF EXISTS Music_genres;
DROP TABLE IF EXISTS Sub_music_genres;
DROP TABLE IF EXISTS Artist_music_genres;
DROP TABLE IF EXISTS Artist_sub_music_genres;
DROP TABLE IF EXISTS Band_music_genres;
DROP TABLE IF EXISTS Band_sub_music_genres;

-- Enables again foreign keys
SET FOREIGN_KEY_CHECKS = 1;

-- Continent Lookup Table
CREATE TABLE IF NOT EXISTS Continent (
    continent_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);
-- Location Table
CREATE TABLE IF NOT EXISTS Location (
    location_id INT AUTO_INCREMENT PRIMARY KEY ,
    address VARCHAR(255) NOT NULL,
    city VARCHAR(255) NOT NULL,
    longitude DECIMAL(9,6) NOT NULL,
    latitude DECIMAL(9,6) NOT NULL,
    country VARCHAR(255) NOT NULL,
    continent_id INT NOT NULL,
    photo_url TEXT NOT NULL,
    photo_description TEXT NOT NULL,

    FOREIGN KEY (continent_id) REFERENCES Continent(continent_id)
);

-- Festival Table
CREATE TABLE IF NOT EXISTS Festival (
    festival_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    location_id INT NOT NULL,
    year INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    duration_days INT,
    photo_url TEXT NOT NULL,
    photo_description TEXT NOT NULL,

    FOREIGN KEY (location_id) REFERENCES Location(location_id),
    UNIQUE (year, name) /* Constraint: only one specific festival per year*/
);
/*To Do: Trigger to check the location of the next year*/

-- Stage Table
CREATE TABLE IF NOT EXISTS Stage (
    stage_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
```

```
    description TEXT,
    max_capacity INT,
    infos_technical_equipment TEXT,
    photo_url TEXT NOT NULL,
    photo_description TEXT NOT NULL
);

-- Event Table
CREATE TABLE IF NOT EXISTS Event (
    event_id INT NOT NULL PRIMARY KEY,
    festival_id INT NOT NULL,
    stage_id INT NOT NULL,
    event_name VARCHAR(255) NOT NULL,
    event_date DATE NOT NULL,
    duration INT NOT NULL,
    photo_url TEXT NOT NULL,
    photo_description TEXT NOT NULL,
    FOREIGN KEY (festival_id) REFERENCES Festival(festival_id),
    FOREIGN KEY (stage_id) REFERENCES Stage(stage_id)
);

-- Staff Role Table
CREATE TABLE IF NOT EXISTS Staff_role (
    staff_role_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);

-- Experience level Lookup Table
CREATE TABLE IF NOT EXISTS Experience_level (
    experience_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);

-- Staff Table
CREATE TABLE IF NOT EXISTS Staff (
    staff_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    age INT NOT NULL,
    staff_role_id INT NOT NULL,
    experience_id INT NOT NULL,
    photo_url TEXT NOT NULL,
    photo_description TEXT NOT NULL,

    FOREIGN KEY (staff_role_id) REFERENCES Staff_role(staff_role_id),
    FOREIGN KEY (experience_id) REFERENCES Experience_level(experience_id)
);

-- Music genre Lookup Table --
CREATE TABLE IF NOT EXISTS Music_genres(
    music_genre_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

-- Sub music genre Lookup Table --
CREATE TABLE IF NOT EXISTS Sub_music_genres(
    sub_music_genre_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    music_genre_id INT,

    FOREIGN KEY (music_genre_id) REFERENCES Music_genres(music_genre_id)
);
-- Artist Table
CREATE TABLE IF NOT EXISTS Artist (
    artist_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    nickname VARCHAR(255),
    birth_date DATE NOT NULL,
    is_solo BOOLEAN NOT NULL,
    website VARCHAR(255),
    instagram VARCHAR(255),
    photo_url TEXT NOT NULL,
    photo_description TEXT NOT NULL

);

-- Artist Music genres many-to-many relationship Table --
```

```sql
CREATE TABLE IF NOT EXISTS Artist_music_genres (
    artist_id INT,
    music_genre_id INT,
    PRIMARY KEY (artist_id, music_genre_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (music_genre_id) REFERENCES Music_genres(music_genre_id)
);

-- Artist Sub Music genres many-to-many relationship Table --
CREATE TABLE IF NOT EXISTS Artist_sub_music_genres (
    artist_id INT,
    sub_music_genre_id INT,
    PRIMARY KEY (artist_id, sub_music_genre_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (sub_music_genre_id) REFERENCES Sub_music_genres(sub_music_genre_id)
);

-- Band Table
CREATE TABLE IF NOT EXISTS Band (
    band_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    formation_date DATE,
    member_list TEXT,
    instagram TEXT,
    website TEXT,
    photo_url TEXT,
    photo_description TEXT

);

CREATE TABLE IF NOT EXISTS Band_music_genres (
    band_id INT,
    music_genre_id INT,
    PRIMARY KEY (band_id, music_genre_id),
    FOREIGN KEY (band_id) REFERENCES Band(band_id),
    FOREIGN KEY (music_genre_id) REFERENCES Music_genres(music_genre_id)
);

CREATE TABLE IF NOT EXISTS Band_sub_music_genres (
    band_id INT,
    sub_music_genre_id INT,
    PRIMARY KEY (band_id, sub_music_genre_id),
    FOREIGN KEY (band_id) REFERENCES Band(band_id),
    FOREIGN KEY (sub_music_genre_id) REFERENCES Sub_music_genres(sub_music_genre_id)
);

-- Performs Table
CREATE TABLE IF NOT EXISTS Performs (
    performs_id INT NOT NULL PRIMARY KEY,
    artist_id INT,
    band_id INT,
   /* performance_id INT,*/

    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (band_id) REFERENCES Band(band_id),
    /*FOREIGN KEY (performance_id) REFERENCES Performance(performance_id),*/

    CONSTRAINT chk_only_one_entity_of_performer
    CHECK(
        (artist_id IS NOT NULL AND band_id IS NULL) OR (artist_id IS NULL AND band_id IS NOT NULL)
    )
);

-- Performance Table
CREATE TABLE IF NOT EXISTS Performance (
    performance_id INT NOT NULL PRIMARY KEY,
    event_id INT NOT NULL,
    performs_id INT NOT NULL,
    type_of_performance VARCHAR(255) NOT NULL,
    duration FLOAT NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    photo_url TEXT,
    photo_description TEXT,

    CONSTRAINT duration_of_performance
```

```
        CHECK (duration <= 3.0),

    FOREIGN KEY(event_id) REFERENCES Event(event_id),
    FOREIGN KEY(performs_id) REFERENCES Performs(performs_id)
);

-- Visitor Table
CREATE TABLE IF NOT EXISTS Visitor (
    visitor_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    age INT NOT NULL,
    email VARCHAR(255) UNIQUE,
    phone_number VARCHAR(20) NOT NULL,
    photo_url TEXT,
    photo_description TEXT,

    CONSTRAINT check_age CHECK (age >= 12 AND age <= 99),
    CONSTRAINT check_email
        CHECK (
            email IS NULL OR
            email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        )
);

-- Ticket Type Table
CREATE TABLE IF NOT EXISTS Ticket_type (
    ticket_type_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);

-- Payment_method Lookup Table
CREATE TABLE IF NOT EXISTS Payment_method (
    payment_method_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,

    CONSTRAINT chk_payment_method CHECK (name IN ('Credit Card', 'Debit Card', 'Bank Transfer'))
);

-- Ticket Table
CREATE TABLE IF NOT EXISTS Ticket (
    ticket_id INT NOT NULL PRIMARY KEY,
    event_id INT NOT NULL,
    visitor_id INT NOT NULL,
    ticket_type_id INT NOT NULL,
    purchase_date DATE NOT NULL,
    price DECIMAL(6,2) NOT NULL,
    payment_method_id INT NOT NULL,
    ean_code VARCHAR(20) NOT NULL,
    used BOOLEAN NOT NULL DEFAULT FALSE,
    photo_url TEXT,
    photo_description TEXT,
    FOREIGN KEY (event_id) REFERENCES Event(event_id),
    FOREIGN KEY (visitor_id) REFERENCES Visitor(visitor_id),
    FOREIGN KEY (ticket_type_id) REFERENCES Ticket_type(ticket_type_id),
    FOREIGN KEY (payment_method_id) REFERENCES Payment_method(payment_method_id),

    CONSTRAINT check_price_be_over_zero CHECK (price > 0),
    CONSTRAINT check_used_value CHECK (used IS TRUE OR used IS FALSE)

);

-- Rating Table (we use likert rating)
CREATE TABLE IF NOT EXISTS Rating (
    rating_id INT NOT NULL PRIMARY KEY,
    ticket_id INT NOT NULL,
    performance_id INT NOT NULL,
    interpretation_rating INT NOT NULL CHECK (interpretation_rating BETWEEN 1 AND 5),
    sound_and_lighting_rating INT NOT NULL CHECK (sound_and_lighting_rating BETWEEN 1 AND 5),
    stage_presence_rating INT NOT NULL CHECK (stage_presence_rating BETWEEN 1 AND 5),
    organization_rating INT NOT NULL CHECK (organization_rating BETWEEN 1 AND 5),
    overall_impression INT NOT NULL CHECK (overall_impression BETWEEN 1 AND 5),
    photo_url TEXT,
    photo_description TEXT,
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id),
    FOREIGN KEY (performance_id) REFERENCES Performance(performance_id)
```

```
);

-- Resale Buyer Table
CREATE TABLE IF NOT EXISTS Resale_Buyer (
    buyer_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    age INT NOT NULL,
    email VARCHAR(255) UNIQUE,
    phone_number VARCHAR(20),
    photo_url TEXT,
    photo_description TEXT,

    CONSTRAINT chk_age CHECK (age >= 12 AND age <= 99),
    CONSTRAINT chk_email
        CHECK (
            email IS NULL OR
            email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        )
);

-- Resale Ticket Table
CREATE TABLE IF NOT EXISTS Resale_Queue (
    resale_id INT NOT NULL PRIMARY KEY,
    ticket_id INT NOT NULL,
    seller_id INT NOT NULL,  /* first owner of the ticket */
    listing_date DATE,
    price  FLOAT(6,2) NOT NULL,
    status BOOLEAN NOT NULL,

    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id) ,
    FOREIGN KEY (seller_id) REFERENCES Visitor(visitor_id)
);

-- Works on Table
CREATE TABLE IF NOT EXISTS Works_on (
    staff_id INT NOT NULL,
    stage_id INT NOT NULL,
    event_id INT NOT NULL,

    PRIMARY KEY (stage_id, staff_id, event_id),
    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id),
    FOREIGN KEY (stage_id) REFERENCES Stage(stage_id),
    FOREIGN KEY (event_id) REFERENCES Event(event_id)
);

CREATE TABLE IF NOT EXISTS Demand_Queue(
    demand_id INT NOT NULL PRIMARY KEY,
    buyer_id INT NOT NULL,
    preferred_ticket_type INT,
    preferred_event_id INT,
    request_date DATE NOT NULL,
    status BOOLEAN NOT NULL DEFAULT FALSE,

    FOREIGN KEY (buyer_id) REFERENCES Resale_Buyer(buyer_id) ON DELETE CASCADE
);

-- Table to store logs from direct buys
CREATE TABLE IF NOT EXISTS Buys_specific_ticket(
    buyer_id INT NOT NULL,
    resale_id INT NOT NULL,
    interest_date DATE,
    status VARCHAR(20),


    PRIMARY KEY (buyer_id, resale_id)
);

-- Belongs to Table
CREATE TABLE IF NOT EXISTS Belongs_to (
    artist_id INT NOT NULL,
    band_id INT NOT NULL,

    PRIMARY KEY (artist_id, band_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (band_id) REFERENCES Band(band_id)
```

```sql
);

CREATE TABLE Resale_Log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    ticket_id INT,
    old_owner_id INT,
    new_owner_id INT,
    sale_price DECIMAL(10,2),
    sale_date DATETIME DEFAULT NOW()
);

-- Creating Indexes after all tables are created
-- indexes  --
CREATE INDEX idx_ticket_event ON Ticket(event_id);
CREATE INDEX idx_performs_artist_id ON Performs(artist_id);
CREATE INDEX idx_performs_band_id ON Performs(band_id);
CREATE INDEX idx_performance_event_performs ON Performance(event_id, performs_id);
CREATE INDEX idx_ticket_visitor_used ON Ticket(visitor_id, used, ticket_id);
CREATE INDEX idx_rating_ticket_perf ON Rating(ticket_id, performance_id);
CREATE INDEX idx_ticket_visitor ON Ticket(ticket_id, visitor_id);
CREATE INDEX idx_rating_performance_id ON Rating(performance_id); --Q4
CREATE INDEX idx_rating_ticket_id ON Rating(ticket_id, performance_id); --Q6
CREATE INDEX idx_perf_event_start ON Performance(performance_id, event_id, start_time); --Q6


-- View to check the population of assigned staff to every stage --
CREATE OR REPLACE VIEW Stage_Staff_Coverage AS
SELECT
    w.event_id,
    w.stage_id,
    stg.name AS stage_name,
    stf.staff_role_id,
    r.name AS role_name,
    COUNT(*) AS assigned_staff,
    stg.max_capacity,
    CASE
        WHEN stf.staff_role_id = 2 THEN ROUND(stg.max_capacity * 0.05, 0)
        WHEN stf.staff_role_id = 3 THEN ROUND(stg.max_capacity * 0.02, 0)
        ELSE 0
    END AS required_staff,
    CASE
        WHEN stf.staff_role_id IN (2, 3) THEN
            CASE
                WHEN COUNT(*) >=
                    CASE
                        WHEN stf.staff_role_id = 2 THEN ROUND(stg.max_capacity * 0.05, 0)
                        WHEN stf.staff_role_id = 3 THEN ROUND(stg.max_capacity * 0.02, 0)
                    END
                THEN '✔ OK'
                ELSE '✘ Understaffed'
            END
        ELSE 'N/A'
    END AS status
FROM Works_on w
JOIN Stage stg ON w.stage_id = stg.stage_id
JOIN Staff stf ON w.staff_id = stf.staff_id
JOIN Staff_role r ON stf.staff_role_id = r.staff_role_id
WHERE stf.staff_role_id IN (1, 2, 3)
GROUP BY w.event_id, w.stage_id, stf.staff_role_id;


-- ==================================
-- TRIGGERS INSERTION
-- ==================================

DELIMITER //

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

```
    IF NEW.ticket_type_id = 1 THEN

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

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

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

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

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

    END IF;
END;
//


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

    DECLARE current_count INT;
    DECLARE max_capacity INT;
    DECLARE event_id_for_ticket INT;

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

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

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

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


END;

//


CREATE TRIGGER trg_after_resale_insert
AFTER INSERT ON Resale_Queue
FOR EACH ROW
BEGIN
```

```
    DECLARE v_demand_id INT;
    DECLARE v_buyer_id INT;
    DECLARE ticket_count INT;
    DECLARE new_visitor_id INT;

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

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

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

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


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

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

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

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

END;

//

CREATE TRIGGER trg_after_demand_queue_insert
AFTER INSERT ON Demand_Queue
FOR EACH ROW
```

```
BEGIN
    DECLARE v_resale_id INT;
    DECLARE v_seller_id INT;
    DECLARE ticket_count INT;
    DECLARE new_visitor_id INT;
    DECLARE v_ticket_id INT;
    DECLARE v_event_id INT;
    DECLARE v_ticket_type INT;
    DECLARE resale_price FLOAT;

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

    IF v_resale_id IS NOT NULL THEN

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

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

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

            UPDATE Ticket
            SET visitor_id = new_visitor_id
            WHERE ticket_id = v_ticket_id;

        ELSE  /*Update infos about visitor*/
            UPDATE Visitor
            JOIN Resale_Buyer rb ON rb.buyer_id = NEW.buyer_id
            SET
                Visitor.name = rb.name,
                Visitor.surname = rb.surname,
                Visitor.age = rb.age,
                Visitor.email = rb.email,
                Visitor.phone_number = rb.phone_number,
                Visitor.photo_url = rb.photo_url,
                Visitor.photo_description = rb.photo_description
            WHERE Visitor.visitor_id = v_seller_id;
        END IF;

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

        );


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

    END IF;


END;
```

```
//

CREATE TRIGGER chk_after_insert_to_performance_for_brake
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN

    DECLARE last_end_time TIME;

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

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

//

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

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

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

//

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

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

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

    IF ticket_used = FALSE THEN
    set @message_text = CONCAT_WS('Cannot insert rating: ticket with ID', ticket_id_str, ' was not used.');
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = @message_text;
    END IF;
END;


//

-- Trigger to check if the artist whos going to be inserted on a band is solo or not
```

```sql
CREATE TRIGGER check_if_artist_is_solo_blngs_to
BEFORE INSERT ON Belongs_to
FOR EACH ROW
BEGIN

    DECLARE v_is_solo BOOLEAN;

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

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

END;

//

DELIMITER ;

/* All seem to work. Check the status in resale and demand queue tables */




-- ==================================
-- PROCEDURES INSERTION
-- ==================================

-- ============================================================================
-- Procedure: buy_specific_ticket
-- Description:
--     Handles the direct purchase of a specific resale ticket by a buyer.
--     The procedure performs the following steps:
--       1. Retrieves the ticket associated with the given resale ID.
--       2. Updates the current ticket owner (Visitor) with the buyer's information.
--       3. Records the purchase in the Buys_specific_ticket log table.
--       4. Deletes the resale entry from the Resale_Queue.
--     This procedure assumes that resale_id exists before calling and that
--     no foreign key is enforced on resale_id in the log table to allow deletion.
-- Parameters:
--     IN in_buyer_id     INT     → The buyer's ID (from Resale_Buyer table).
--     IN in_resale_id    INT     → The resale entry ID (from Resale_Queue table).

--      CALL buy_specific_ticket(<buyer_id>, <resale_id>);
-- ============================================================================

DELIMITER $$

CREATE PROCEDURE buy_specific_ticket(
    IN in_buyer_id INT,
    IN in_resale_id INT
)
BEGIN

    DECLARE ticket_id_var INT;
    DECLARE current_visitor_id INT;
    DECLARE ticket_count INT;
    DECLARE new_visitor_id INT;
/*Find ticket id from resale queue*/
SELECT ticket_id
INTO ticket_id_var
FROM Resale_Queue
WHERE resale_id = in_resale_id;

/*Find visitor from the ticket*/
SELECT visitor_id
INTO current_visitor_id
FROM Ticket
WHERE ticket_id = ticket_id_var;

/*Count how many tickets the seller owns*/
```

```
SELECT COUNT(*) INTO ticket_count
FROM Ticket
WHERE visitor_id = current_visitor_id;

IF ticket_count > 1 THEN
    /*Create new visitor*/

    INSERT INTO Visitor (name, surname, age, email, phone_number, photo_url, photo_description)
    SELECT name, surname, age, email, phone_number, photo_url, photo_description
    FROM Resale_Buyer
    WHERE buyer_id = in_buyer_id;

    SET new_visitor_id = LAST_INSERT_ID();

    UPDATE Ticket
    SET visitor_id = new_visitor_id
    WHERE ticket_id = ticket_id_var;

ELSE
    /*Set the buyer's values to visitor's values (change buyer with visitor)*/
    UPDATE Visitor
    SET
    name = (SELECT name FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
    surname = (SELECT surname FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
    age = (SELECT age FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
    email = (SELECT email FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
    phone_number = (SELECT phone_number FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
    photo_url = (SELECT photo_url FROM Resale_Buyer WHERE buyer_id = in_buyer_id),
    photo_description = (SELECT photo_description FROM Resale_Buyer WHERE buyer_id = in_buyer_id)
    WHERE visitor_id = current_visitor_id;
END IF;

/*Logging the transaction*/
INSERT INTO Buys_specific_ticket(buyer_id, resale_id, interest_date, status)
VALUES
(in_buyer_id, in_resale_id, NOW(), 'completed');

/*Update status of the item from the resale queue*/
UPDATE Resale_Queue
SET Status = TRUE
WHERE resale_id = in_resale_id;

DELETE FROM Resale_Buyer
WHERE buyer_id = in_buyer_id;


END$$

/*=============================================
  Procedure to insert data into Works_On Table
  =============================================
*/

DELIMITER $$

CREATE PROCEDURE Insert_Staff_Assignment (
    IN in_staff_id INT,
    IN in_stage_id INT,
    IN in_event_id INT
)
BEGIN
    -- Check if assignment already exists
    IF EXISTS (
        SELECT 1 FROM Works_on
        WHERE staff_id = in_staff_id AND stage_id = in_stage_id AND event_id = in_event_id
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Η ανάθεση υπάρχει ήδη.';
    ELSE
        INSERT INTO Works_on (staff_id, stage_id, event_id)
        VALUES (in_staff_id, in_stage_id, in_event_id);
    END IF;
END $$


DELIMITER $$
```

```
/*=====================================================================
  Check staff coverage per stage and event based on role requirements
  =====================================================================
*/
CREATE OR REPLACE PROCEDURE Check_Stage_Staff_Coverage()
BEGIN
    SELECT
        w.event_id,
        w.stage_id,
        stg.name AS stage_name,
        stf.staff_role_id,
        r.name AS role_name,
        COUNT(*) AS assigned_staff,
        stg.max_capacity,
        CASE
            WHEN stf.staff_role_id = 2 THEN ROUND(stg.max_capacity * 0.05)
            WHEN stf.staff_role_id = 3 THEN ROUND(stg.max_capacity * 0.02)
            ELSE 0
        END AS required_staff,
        CASE
            WHEN stf.staff_role_id IN (2, 3) THEN
                CASE
                    WHEN COUNT(*) >=
                        CASE
                            WHEN stf.staff_role_id = 2 THEN ROUND(stg.max_capacity * 0.05)
                            WHEN stf.staff_role_id = 3 THEN ROUND(stg.max_capacity * 0.02)
                        END
                    THEN '✔ OK'
                    ELSE '✘ Understaffed'
                END
            ELSE 'N/A'
        END AS status
    FROM Works_on w
    JOIN Stage stg ON w.stage_id = stg.stage_id
    JOIN Staff stf ON w.staff_id = stf.staff_id
    JOIN Staff_role r ON stf.staff_role_id = r.staff_role_id
    WHERE stf.staff_role_id IN (1, 2, 3)
    GROUP BY w.event_id, w.stage_id, stf.staff_role_id;
END $$

DELIMITER ;
```