Αναφορά Εξαμηνιαίας Εργασίας

Βάσεις Δεδομένων

Βασιλείου Παναγιώτα	03122806
Μπάλλος Γεώργιος	03122076
Παναγιωτάκου Μαρία-Ελένη	03122047

14 Μαΐου 2025

Περιεχόμενα

Κεφάλαιο 1

Βάση Δεδομένων

1.1 Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship Diagram)

Αρχικά, σχεδιάσαμε το **διάγραμμα οντοτήτων-συσχετίσεων (E-R Diagram)** που προκύπτει από την εκφώνηση όπως φαίνεται στο Σχήμα 1.1.

Ιδιαίτερο ενδιαφέρον παρουσιάζει ο τρόπος υλοποίησης της **ουράς μεταπώλησης.** Αποτελείται απο δύο βασικούς πίνακες δεδομένων, έναν για τους **αγοραστές** και έναν για τα **εισητήρια προς μεταπώληση**. Και οι δύο πίνακες περιέχουν τα βασικά στοιχεία του εισιτηρίου (που έχει ζητηθεί ή διατεθεί) καθώς και ένα timestamp ζήτησης/διάθεσης για την υλοποίηση της λειτουργίας FIFO. Έτσι έχουμε όλα τα δεδομένα που χρειάζονται για να υλοποήσουμε αργότερα (σε επίπεδο βάσης) τις διαδικασίες που ενεργοποιούν την ουρά και ταιριάζουν αγοραστές και εισητήρια.

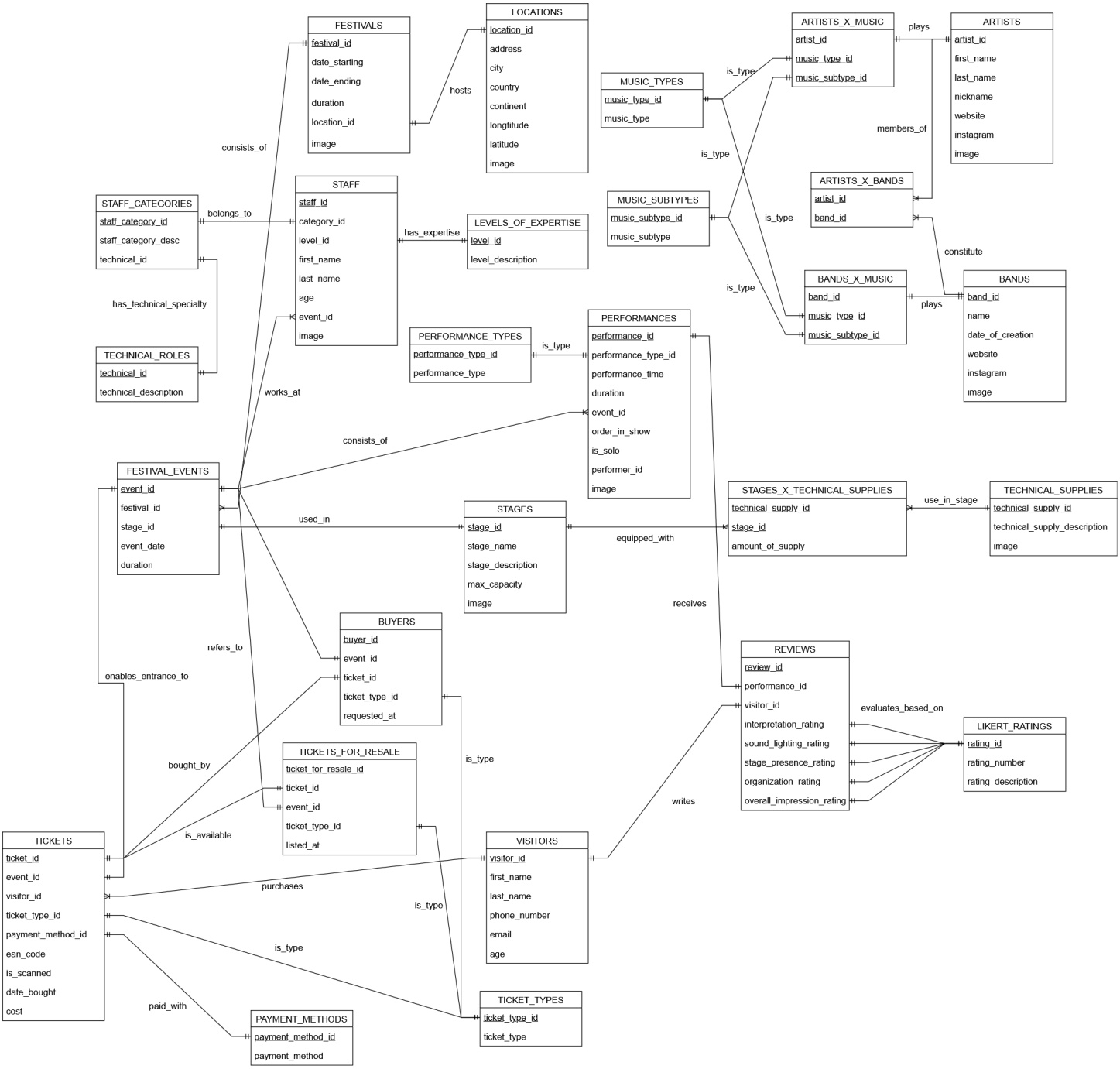
1.2 Σχεσιακό Σχήμα (Relational Schema)

Αφού υλοποιήσαμε το κύριο σχήμα της βάσης, οδηγηθήκαμε στην εξαγωγή του **Σχεσιακού Σχήματος (Relational Schema)** το οποίο είναι το εξής:

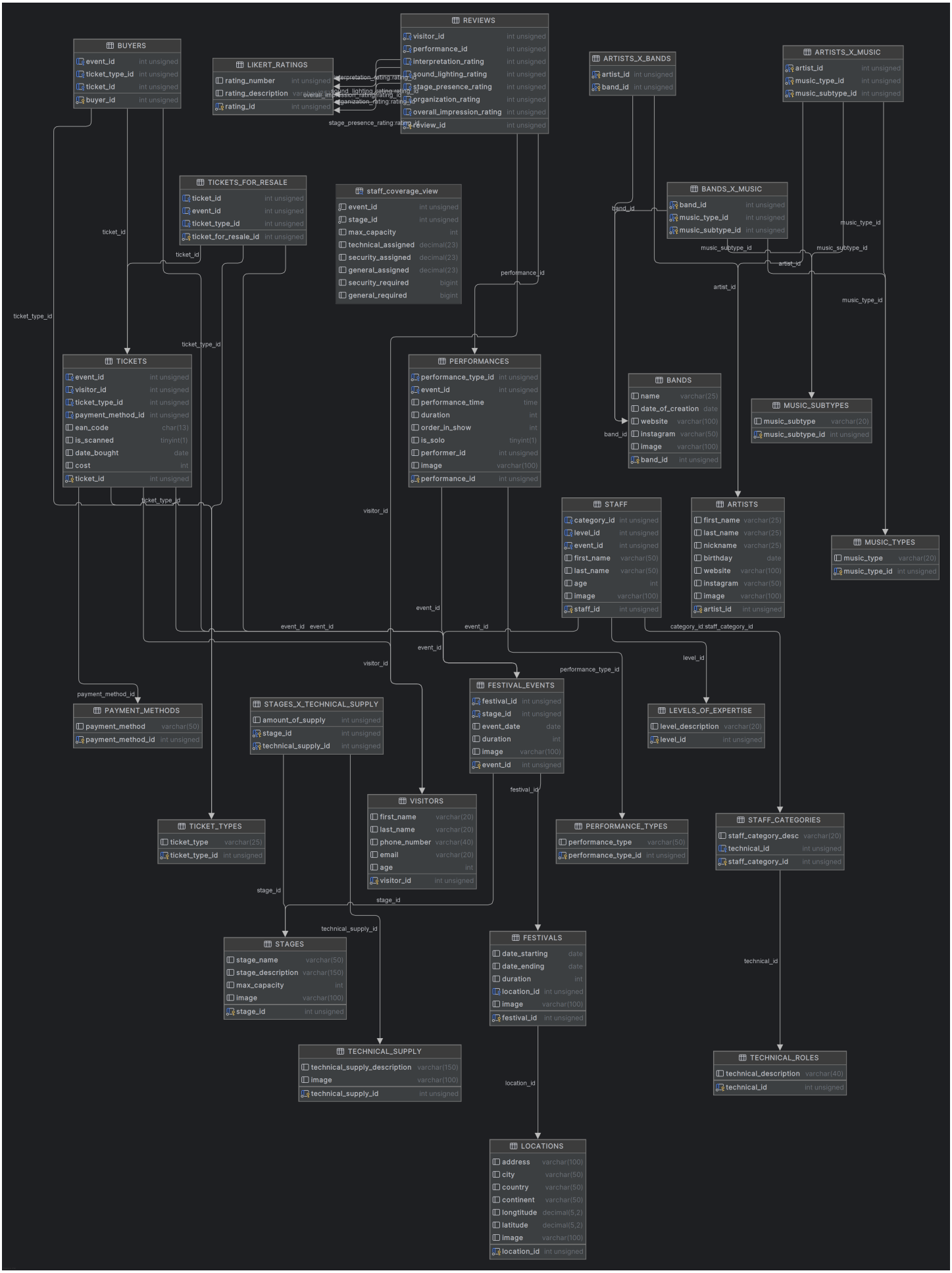
- LOCATIONS (location_id, address, city, country, continent, longtitude, latitude, image)
- FESTIVALS (festival_id, date_starting, date_ending, duration, location_id, image, festival_year)
- STAGES (stage_id, stage_name, stage_description, max_capacity, image)
- TECHNICAL_SUPPLY (technical_supply_id, technical_supply_description, image)
- STAGES_X_TECHNICAL_SUPPLY (stage_id, technical_supply_id, amount_of_supply)
- FESTIVAL_EVENTS (event_id, festival_id, stage_id, event_date, duration, image)
- MUSIC_TYPES (music_type_id, music_type)
- MUSIC_SUBTYPES (music_subtype_id, music_subtype)

- ARTISTS (artist_id, first_name, last_name, nickname, birthday, website, instagram, image)
- BANDS (band_id, name, date_of_creation, website, instagram, image)
- ARTISTS_X_BANDS (artist_id, band_id)
- ARTISTS_X_MUSIC (artist_id, music_type_id, music_subtype_id)
- BANDS_X_MUSIC (band_id, music_type_id, music_subtype_id)
- PERFORMANCE_TYPES (performance_type_id, performance_type)
- PERFORMANCES (performance_id, performance_type_id, event_id, performance_time, duration, order_in_show, is_solo, performer_id, image)
- TECHNICAL_ROLES (technical_id, technical_description)
- STAFF_CATEGORIES (staff_category_id, staff_category_desc, technical_id)
- LEVELS_OF_EXPERTISE (level_id, level_description)
- STAFF (staff_id, category_id, level_id, event_id, first_name, last_name, age, image)
- VISITORS (visitor_id, first_name, last_name, phone_number, email, age)
- TICKET_TYPES (ticket_type_id, ticket_type)
- PAYMENT_METHODS (payment_method_id, payment_method)
- TICKETS (ticket_id, event_id, visitor_id, ticket_type_id, payment_method_id, ean_code, is_scanned, date_bought, cost)
- LIKERT_RATINGS (rating_id, rating_number, rating_description)
- REVIEWS (review_id, visitor_id, performance_id, interpretation_rating, sound_lighting_rating, stage_presence_rating, organization_rating, overall_impression_rating)
- BUYERS (buyer_id, event_id, ticket_type_id, ticket_id, requested_at)
- TICKETS_FOR_RESALE (ticket_for_resale_id, ticket_id, event_id, ticket_type_id, listed_at)

Το διάγραμμα Σχήματος προέκυψε βάσει του Σχεσιακού Σχήματος όπως φαίνεται στο Σχήμα 1.2.



Σχήμα 1.1: Διάγραμμα Οντοτήτων-Συσχετίσεων (E-R)



Σχήμα 1.2: Σχεσιακό Σχήμα

1.3 Περιορισμοί (Constraints) και Πυροδότες (Triggers)

Οι **περιορισμοί (constraints)** εξασφαλίζουν την ορθότητα και την αξιοπιστία της βάσης δεδομένων. Στην παρούσα εργασία εφαρμόζονται οι εξής κατηγορίες περιορισμών ακεραιότητας:

Πρωτεύοντα Κλειδιά (Primary Keys)

Κάθε πίνακας περιλαμβάνει ένα πρωτεύον κλειδί για την μοναδική ταυτοποίηση των εγγραφών σε αυτό. Συγκεκριμένα:

- Για τη σχέση `LOCATIONS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(location_id)`
- Για τη σχέση `FESTIVALS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(festival_id)`
- Για τη σχέση `STAGES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(stage_id)`
- Για τη σχέση `TECHNICAL_SUPPLY` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(technical_supply_id)`
- Για τη σχέση `STAGES_X_TECHNICAL_SUPPLY` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(stage_id, technical_supply_id)`
- Για τη σχέση `FESTIVAL_EVENTS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(event_id)`
- Για τη σχέση `MUSIC_TYPES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(music_type_id)`
- Για τη σχέση `MUSIC_SUBTYPES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(music_subtype_id)`
- Για τη σχέση `ARTISTS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(artist_id)`
- Για τη σχέση `BANDS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(band_id)`
- Για τη σχέση `ARTISTS_X_BANDS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(artist_id, band_id)`
- Για τη σχέση `ARTISTS_X_MUSIC` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(artist_id, music_type_id, music_subtype_id)`
- Για τη σχέση `BANDS_X_MUSIC` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(band_id, music_type_id, music_subtype_id)`
- Για τη σχέση `PERFORMANCE_TYPES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(performance_type_id)`
- Για τη σχέση `PERFORMANCES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(performance_id)`
- Για τη σχέση `TECHNICAL_ROLES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(technical_id)`
- Για τη σχέση `STAFF_CATEGORIES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(staff_category_id)`
- Για τη σχέση `LEVELS_OF_EXPERTISE` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(level_id)`
- Για τη σχέση `STAFF` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(staff_id)`
- Για τη σχέση `VISITORS` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(visitor_id)`
- Για τη σχέση `TICKET_TYPES` ορίζεται πρωτεύον κλειδί `PRIMARY KEY(ticket_type_id)`

- Για τη σχέση PAYMENT_METHODS ορίζεται πρωτεύον κλειδί PRIMARY KEY(payment_method_id)
- Για τη σχέση TICKETS ορίζεται πρωτεύον κλειδί PRIMARY KEY(ticket_id)
- Για τη σχέση LIKERT_RATINGS ορίζεται πρωτεύον κλειδί PRIMARY KEY(rating_id)
- Για τη σχέση REVIEWS ορίζεται πρωτεύον κλειδί PRIMARY KEY(review_id)
- Για τη σχέση BUYERS ορίζεται πρωτεύον κλειδί PRIMARY KEY(buyer_id)
- Για τη σχέση TICKETS_FOR_RESALE ορίζεται πρωτεύον κλειδί PRIMARY KEY(ticket_for_resale_id)

Ξένα Κλειδιά (Foreign Keys)

Σε αρκετούς πίνακες εμφανίζονται attributes των οποίων οι τιμές σχετίζονται με εγγραφές άλλων πινάκων. Για τη διασφάλιση της συνέπειας αυτών των σχέσεων, ορίζονται περιορισμοί αναφορικής ακεραιότητας μέσω ξένων κλειδιών (foreign keys), οι οποίοι εξασφαλίζουν ότι οι τιμές αυτές αντιστοιχούν σε υπαρκτές εγγραφές των σχετιζόμενων πινάκων. Ενδεικτικά αναφέρουμε τα εξής:

- Για τη σχέση FESTIVALS ορίζεται ξένο κλειδί FOREIGN KEY(location_id) REFERENCES LOCATIONS(location_id)
- Για τη σχέση STAGES_X_TECHNICAL_SUPPLY ορίζονται ξένα κλειδιά FOREIGN KEY(stage_id) REFERENCES STAGES(stage_id) και FOREIGN KEY(technical_supply_id) REFERENCES TECHNICAL_SUPPLY(technical_supply_id)

Περιορισμοί NOT NULL

Οι περιορισμοί NOT NULL χρησιμοποιούνται για να διασφαλίσουν ότι τα αντίστοιχα πεδία δεν θα παραμείνουν κενά (null) κατά την εισαγωγή ή τροποποίηση μιας εγγραφής. Στη βάση μας τους χρησιμοποιούμε σχεδόν εξ ολοκλήρου για τα πρωτεύοντα κλειδιά.

Περιορισμοί CHECK

Ο όρος CHECK(P) καθορίζει ένα κατηγόρημα P που πρέπει να ικανοποιείται από κάθε πλειάδα μίας σχέσης της βάσης. Ενδεικτικά αναφέρουμε τα εξής:

- Για τις σχέσεις που χρησιμοποιείται και εικόνα, ο όρος CHECK(image LIKE 'https://%') ελέγχει ότι το πεδίο image περιέχει έγκυρο σύνδεσμο εικόνας που ξεκινά με https://
- Στη σχέση LIKERT_RATINGS, ο όρος CHECK(rating_number BETWEEN 1 AND 5) επιτρέπει μόνο έγκυρες τιμές αξιολόγησης σύμφωνα με την κλίμακα Likert

Περιορισμοί UNIQUE

Οι περιορισμοί UNIQUE χρησιμοποιούνται για να διασφαλίσουν ότι το αντίστοιχο attribute μιας σχέσης περιέχει μοναδικές τιμές μεταξύ των πλειάδων της σχέσης. Ενδεικτικά αναφέρουμε τα εξής:

- Στη σχέση FESTIVALS, οι όροι UNIQUE(festival_year) και UNIQUE(location_id) ελέγχουν ότι το φεστιβάλ διεξάγεται μία φορά ανά έτος και σε διαφορετική τοποθεσία ανά έτος αντίστοιχα
- Στη σχέση TICKETS, ο όρος UNIQUE(event_id, visitor_id) ελέγχει ότι κάθε επισκέπτης μπορεί να αγοράσει ένα εισιτήριο ανά ημέρα και παράσταση

Σημειώνεται ότι, σε αντίθεση με το `PRIMARY KEY`*, ο περιορισμός* `UNIQUE` *επιτρέπει την ύπαρξη* `NULL` *τιμών, εκτός αν οριστεί διαφορετικά.*

Κατά τον σχεδιασμό του σχήματος της βάσης δεδομένων χρησιμοποιήθηκαν, επίσης, κατάλληλοι τύποι δεδομένων όπως `VARCHAR(x)`, `DATE`, `BOOLEAN` κ.α. ώστε να ανταποκρίνονται στις ανάγκες της εφαρμογής και να διασφαλίζεται η αποδοτικότητα και η ακρίβεια στην αποθήκευση των πληροφοριών.

Οι **πυροδότες (triggers)** είναι εντολές που εκτελεί το σύστημα αυτόματα ως αποτέλεσμα μίας τροποποίησης στη βάση δεδομένων και μπορούν να υλοποιήσουν σύνθετους περιορισμούς.

Το trigger

```
CREATE TRIGGER delete_performance_after_artist
AFTER DELETE ON ARTISTS
FOR EACH ROW
BEGIN
    DELETE FROM PERFORMANCES
    WHERE performer_id = OLD.artist_id AND is_solo = 1;
END;
```

εξασφαλίζει τη συνοχή των εγγραφών, καθώς όταν διαγράφεται ένας καλλιτέχνης, πρέπει να διαγραφούν όλες οι solo παραστάσεις που τον αφορούν. Έτσι, ελέγχονται όλες οι εγγραφές στον πίνακα `PERFORMANCES` και διαγράφονται τα ξένα κλειδιά τα οποία δημιουργούν αντιστοίχιση σε αυτόν.

Το trigger

```
CREATE TRIGGER delete_performance_after_band
AFTER DELETE ON BANDS
FOR EACH ROW
BEGIN
    DELETE FROM PERFORMANCES
    WHERE performer_id = OLD.band_id AND is_solo = 0;
END;
```

λειτουργεί αντίστοιχα με το παραπάνω. Όταν διαγράφεται ένα συγκρότημα, πρέπει πρώτα να διαγραφούν όλες οι παραστάσεις στις οποίες συμμετείχε, ώστε να μην υπάρχουν «ορφανά» ξένα κλειδιά.

Το trigger

```
CREATE TRIGGER check_review_ticket_scanned
BEFORE INSERT ON REVIEWS
FOR EACH ROW
BEGIN
    DECLARE event_of_perf INT;
    DECLARE ticket_scanned BOOL;

    -- Get the event for the performance being reviewed
    SELECT event_id INTO event_of_perf
    FROM PERFORMANCES
    WHERE performance_id = NEW.performance_id;
```

```
    -- Check if visitor has a scanned ticket for that event
    SELECT COUNT(*) > 0 INTO ticket_scanned
    FROM TICKETS
    WHERE visitor_id = NEW.visitor_id
      AND event_id = event_of_perf
      AND is_scanned = TRUE;

    -- If not, block the review
    IF NOT ticket_scanned THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You can only review performances you attended
        ↪  (ticket must be scanned).';
    END IF;
END
```

ενεργοποιείται με την εισαγωγή μιας κριτικής, με σκοπό τον έλεγχο του εισητηρίου του επισκέπτη που την πραγματοποίησε. Για να είναι έγκυρη μία κριτική, απαιτείται το εισητήριο να είναι σκαναρισμένο. Επομένως, γίνεται ο έλεγχος με βάση το id του επισκέπτη `visitor_id = NEW.visitor_id`, της παράστασης `event_id = event_of_perf` και της αντίστοιχης εμφάνισης που αυτό αφορά `performance_id = NEW.performance_id` και αν έχει σκαναριστεί, επιτρέπεται η εισαγωγή της κριτικής. Στην αντίθετη περίπτωση, ο επισκέπτης δεν μπορεί να παρακολουθήσει και εμφανίζεται μήνυμα σφάλματος.

Το trigger

```
CREATE TRIGGER check_performer
BEFORE INSERT ON PERFORMANCES
FOR EACH ROW
BEGIN
   DECLARE artist_count INT;
   DECLARE band_count INT;

   IF NEW.is_solo = TRUE THEN
       SELECT COUNT(*) INTO artist_count FROM ARTISTS WHERE artist_id =
       ↪  NEW.performer_id;
       IF artist_count = 0 THEN
           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No matching artist for
           ↪  solo performance.';
       END IF;
   ELSE
       SELECT COUNT(*) INTO band_count FROM BANDS WHERE band_id =
       ↪  NEW.performer_id;
       IF band_count = 0 THEN
           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No matching band for
           ↪  ensemble performance.';
       END IF;
   END IF;
END
```

ενεργοποιείται πριν την εισαγωγή μίας παράστασης. Αν αυτή είναι solo, δηλαδή αφορά μεμονωμένο καλλιτέχνη, ελέγχει αν αυτός υπάρχει. Πιο συγκεκριμένα, διατρέχει τους καλλιτέχνες και αν στη μεταβλητή band_count εισαχθεί ο αριθμός 0, τότε δεν υπάρχει ο ζητούμενος καλλιτέχνης. Αν η νέα παράσταση αφορά συγκρότημα, τότε εκτελεί την αντίστοιχη διαδικασία αναζήτησης στα συγκροτήματα. Αν επιστραφεί μηδενικό αποτέλεσμα και στις δύο περιπτώσεις, αποτρέπεται η εισαγωγή της παράστασης και εμφανίζεται το αντίστοιχο μήνυμα σφάλματος.

Το trigger

```
CREATE TRIGGER check_4th_year
BEFORE INSERT ON PERFORMANCES
FOR EACH ROW
BEGIN
    DECLARE performer_year INT;
    DECLARE prev_years INT;

    -- Get the year of the current performance
    SELECT YEAR(event_date) INTO performer_year
    FROM FESTIVAL_EVENTS
    WHERE event_id = NEW.event_id;

    -- Count how many times this performer performed in the 3 years before
    ↪  this one
    IF NEW.is_solo = TRUE THEN
        SELECT COUNT(DISTINCT YEAR(fe.event_date)) INTO prev_years
        FROM PERFORMANCES p
        JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
        WHERE p.performer_id = NEW.performer_id
          AND p.is_solo = 1
          AND YEAR(fe.event_date) BETWEEN performer_year - 3 AND
          ↪  performer_year - 1;
    ELSE
        SELECT COUNT(DISTINCT YEAR(fe.event_date)) INTO prev_years
        FROM PERFORMANCES p
        JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
        WHERE p.performer_id = NEW.performer_id
          AND p.is_solo = 0
          AND YEAR(fe.event_date) BETWEEN performer_year - 3 AND
          ↪  performer_year - 1;
    END IF;

    IF prev_years = 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Performer cannot take part in the festival for
        ↪  more than 3 consecutive years.';
    END IF;
END
```

καλείται πριν την εισαγωγή μιας νέας εμφάνισης, για να εξασφαλίσει πως δε θα συμμετέχει ένας καλλιτέχνης/ένα συγκρότημα για πάνω από τρία συνεχόμενα έτη. Πιο αναλυτικά, στη μεταβλητή `performer_year` εισάγεται το έτος της τρέχουσας νέας παράστασης. Ο ίδιος έλεγχος πραγματοποιείται για solo εμφανίσεις και για συγκροτήματα: γίνεται μέτρηση των ετών `prev_years` στα οποία ο συγκεκριμένος/ο καλλιτέχνης/συγκρότημα έχει συμμετάσχει και ταυτόχρονα ελέγχει αν είναι εντός του ορίου τριών ετών. Αν αυτά είναι 3 συνεχόμενα, η εισαγωγή απορρίπτεται και εμφανίζεται το μήνυμα σφάλματος.

Το trigger

```
CREATE TRIGGER check_stage_capacity
BEFORE INSERT ON TICKETS
FOR EACH ROW
BEGIN
    DECLARE cap INT;
    DECLARE ticket_count INT;
    DECLARE vip_count INT;
    DECLARE ticket_type_name VARCHAR(20);

    SELECT ticket_type INTO ticket_type_name
    FROM TICKET_TYPES
    WHERE ticket_type_id = NEW.ticket_type_id;

    SELECT s.max_capacity INTO cap
    FROM FESTIVAL_EVENTS fe
    JOIN STAGES s ON s.stage_id = fe.event_id
    WHERE NEW.event_id = fe.event_id;

    SELECT COUNT(*) INTO ticket_count
    FROM TICKETS
    WHERE event_id = NEW.event_id;

    SELECT COUNT(*) INTO vip_count
    FROM TICKETS
    WHERE event_id = NEW.event_id AND ticket_type_name = 'VIP';

    IF ticket_count >= cap THEN
        INSERT INTO BUYERS (event_id, ticket_type_id, ticket_id,
        ↪  requested_at)
        VALUES (NEW.event_id, NEW.ticket_type_id, NULL, NOW());
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Exceeded stage capacity. No more tickets
        ↪  available. Added buyer to queue';
    END IF;

    IF vip_count >= 0.1*cap THEN
```

```
            INSERT INTO BUYERS (event_id, ticket_type_id, ticket_id,
            ↪  requested_at)
            VALUES (NEW.event_id, NEW.ticket_type_id, NULL, NOW());
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'No more VIP tickets available. Added buyer to
            ↪  queue';
        END IF;
    END
```

ενεργοποιείται πριν την εισαγωγή ενός εισητηρίου για μια συγκεκριμένη παράσταση. Αρχικά, βρίσκει τον τύπο του νέου εισητηρίου `ticket_type_name` και έπειτα τη μέγιστη χωρητικότητα της σκηνής στην οποία θα πραγματοποιηθεί η παράσταση `cap`. Στη συνέχεια, καταμετρόνται τα εισητήρια `ticket_count`, όπως επίσης και τα VIP εισητήρια `vip_count`, τα οποία έχουν ήδη πωληθεί για την παράσταση. Αν τα εισητήρια (γενικά) φτάσουν ή ξεπεράσουν τη διαθέσιμη χωρητικότητα της σκηνής (`ticket_count` $\geq$ `cap`), τότε το άτομο που επιθυμεί να αγοράσει το εισητήριο εισέρχεται στους αγοραστές, οι οποίοι αναμένουν στην ουρά μεταπώλησης. Επιπλέον, υπάρχει ο περιορισμός πως τα VIP εισητήρια δεν πρέπει να ξεπερνούν το 10% της χωρητικότητας κάθε σκηνής. Άρα, αν αυτό συμβεί (`vip_count` $\geq$ `0.1 * cap`), τότε και πάλι πρέπει να γίνει το ίδιο για τους αντίστοιχους αγοραστές.

Η παρακάτω διαδικασία υλοποιεί το σύστημα διαχείρισης της ουρά επαναπώλησης εισητηρίων, με βάση τους αργοραστές `BUYERS` οι οποίοι επιθυμούν συγκεκριμένα εισητήρια και τα διαθέσιμα εισητήρια για επαναπώληση `TICKETS_FOR_RESALE`.

```
CREATE PROCEDURE match_resale_queue()
BEGIN
    DECLARE b_id INT;
    DECLARE b_event INT;
    DECLARE b_ticket INT;
    DECLARE b_ticket_type INT;
    DECLARE r_id INT;
    DECLARE r_ticket INT;

    match_loop: LOOP
        SELECT buyer_id, event_id, ticket_type_id, ticket_id
        INTO b_id, b_event, b_ticket_type, b_ticket
        FROM BUYERS
        ORDER BY requested_at
        LIMIT 1;

        -- No buyers left
        IF b_id IS NULL THEN
            LEAVE match_loop;
        END IF;

        -- Check if requested by id:
        IF b_ticket IS NOT NULL THEN
            SELECT ticket_for_resale_id
            INTO r_id
```

```
            FROM TICKETS_FOR_RESALE
            WHERE ticket_id = b_ticket
            ORDER BY listed_at
            LIMIT 1;

            IF r_id IS NOT NULL THEN
                DELETE FROM TICKETS_FOR_RESALE WHERE ticket_for_resale_id =
                ↪  r_id;
                DELETE FROM BUYERS WHERE buyer_id = b_id;
                ITERATE match_loop;
            END IF;
        END IF;

        -- Otherwise match by event + type
        SELECT ticket_for_resale_id, ticket_id
        INTO r_id, r_ticket
        FROM TICKETS_FOR_RESALE
        WHERE event_id = b_event AND ticket_type_id = b_ticket_type
        ORDER BY listed_at
        LIMIT 1;

        IF r_id IS NOT NULL THEN
          DELETE FROM TICKETS_FOR_RESALE WHERE ticket_for_resale_id = r_id;
          DELETE FROM BUYERS WHERE buyer_id = b_id;
          ITERATE match_loop;
        END IF;

        -- If here no match found
        LEAVE match_loop;
    END LOOP match_loop;
END
```

Χρησιμοποιείται δομή ουράς (FIFO, First In First Out), επομένως πρώτα θα εξεταστεί η παλαιότερη «αίτηση» για εισητήριο (`ORDER BY requested_at`, default είναι η αύξουσα σειρά - `LIMIT 1`, θα κρατήσει το πρώτο αποτέλεσμα). Η μία περίπτωση είναι ο αγοραστής να επιθυμεί συγκεκριμένο εισητήριο, επομένως πρέπει να έχουμε ακριβές ταίριασμα `ticket_id` (`WHERE ticket_id = b_ticket`). Αν το ζητούμενο εισητήριο βρεθεί, τότε διαγράφεται από τα εισητήρια που διατίθενται προς επαναπώληση, επίσης θα διαγραφεί και ο αγοραστής από την ουρά και η διαδικασία θα επαναληφθεί από την αρχή. Για το γενικότερο ταίριασμα, ο αγοραστής ζητάει εισητήριο συγκεκριμένης παράστασης και συγκεκριμένου τύπου. Αν υπάρχει διαθέσιμο εισητήριο που πληρεί τις προϋποθέσεις, τότε πωλείται στον πελάτη, διαγράφεται από τα διαθέσιμα εισητήρια, όπως επίσης αφαιρείται και ο αγοραστής. Η διαδικασία θα επανεκτελεστεί και η συνθήκη για να τερματίσει είναι να μην υπάρχουν άλλοι διαθέσιμοι πιθανοί αγοραστές.

Το trigger

```
CREATE TRIGGER after_ticket_resale_insert
AFTER INSERT ON TICKETS_FOR_RESALE
```

```
FOR EACH ROW
BEGIN
    DECLARE not_valid BOOL;

    SELECT t.is_scanned INTO not_valid
    FROM TICKETS t
    WHERE (NEW.ticket_id = t.ticket_id);

    IF not_valid = 0 THEN
        CALL match_resale_queue();
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You cannot list scanned ticket.';
    END IF;
END;
```

ενεργοποιείται με την εισαγωγή ενός εισητηρίου στα διαθέσιμα προς επαναπώληση. Πρώτα, το νέο εισητήριο πρέπει να ελεγχθεί και να επιβεβαιωθεί πως δεν είναι σκαναρισμένο. Αν είναι πράγματι διαθέσιμο (`not_valid = 0`), τότε καλείται η διαδικασία η οποία ταιριάζει αγοραστές με τα διαθέσιμα εισητήρια για μεταπώληση. Τα εισητήρια που έχουν ήδη σκαναριστεί απορρίπτονται και εμφανίζεται το μήνυμα σφάλματος.

Το trigger

```
CREATE TRIGGER after_buyer_insert
AFTER INSERT ON BUYERS
FOR EACH ROW
BEGIN
    CALL match_resale_queue();
END;
```

καλείται με την εισαγωγή ενός αγοραστή και καλείται η αυτοματοποιημένη διαδικασία η οποία θα προσπαθήσει να τον ταιριάξει απευθείας με τα διαθέσιμα προς επαναπώληση εισητήρια.

Το trigger

```
CREATE TRIGGER event_dates_in_festival
BEFORE INSERT ON FESTIVAL_EVENTS
FOR EACH ROW
BEGIN
    DECLARE fest_start_date DATE;
    DECLARE fest_end_date DATE;

    SELECT date_starting, date_ending
        INTO fest_start_date, fest_end_date
        FROM FESTIVALS
    WHERE festival_id = NEW.festival_id;
```

```
    IF NEW.event_date < fest_start_date OR NEW.event_date > fest_end_date
    ↪  THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Event date must be between festival start and end
        ↪  dates';
    END IF;
END
```

ενεργοποιείται με την εισαγωγή μίας παράστασης σε ένα φεστιβάλ. Σκοπός του είναι ο έλεγχος της ημερομηνίας που θα πραγματοποιηθεί, η οποία πρέπει να είναι εντός των ημερομηνιών διεξαγωγής του φεστιβάλ στο οποίο ανήκει. Αρχικά αποθηκεύονται οι ημερομηνίες έναρξης και λήξης του φεστιβάλ `fest_start_date`, `fest_end_date`. Αν η ημερομηνία της παράστασης είναι εντός του εύρους που δημιουργείται, αυτή εισάγεται επιτυχώς στο φεστιβάλ. Αντίθετα, εμφανίζεται το μήνυμα που δηλώνει την απόρριψή της.

To trigger

```
CREATE TRIGGER check_break_range
BEFORE INSERT ON PERFORMANCES
FOR EACH ROW
BEGIN
    DECLARE prev_end_seconds INT;
    DECLARE current_start_seconds INT;

    -- Convert new performance time to seconds since midnight
    SET current_start_seconds = TIME_TO_SEC(NEW.performance_time);

    -- Get the most recent performance's end time (not just previous
    ↪  order_in_show)
    SELECT TIME_TO_SEC(ADDTIME(performance_time, SEC_TO_TIME(duration * 60)))
    INTO prev_end_seconds
    FROM PERFORMANCES
    WHERE event_id = NEW.event_id
    ORDER BY performance_time DESC
    LIMIT 1;

    -- Only check if there was a previous performance
    IF prev_end_seconds IS NOT NULL THEN
        SET @time_diff_minutes = (current_start_seconds - prev_end_seconds) /
        ↪  60;

        IF @time_diff_minutes < 5 OR @time_diff_minutes > 30 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Break must be 5-30 minutes.';
        END IF;
    END IF;
END
```

θα ενεργοποιηθεί με την εισαγωγή μιας νέας εμφάνισης σε μία παράσταση και θα εξασφαλίσει πως το διάλειμμα από την προηγούμενή της είναι μεταξύ 5 και 30 λεπτών. Αρχικά, μετατρέπεται η ώρα έναρξής της σε δευτερόλεπτα μετά τα μεσάνυχτα (current_start_seconds). Έπειτα βρίσκεται το τέλος της προηγούμενης παράστασης, με βάση την ώρα που ξεκινάει και το χρόνο που διαρκεί (prev_end_seconds). Η διαφορά της έναρξης current_start_seconds και της λήξης prev_end_seconds μετατρέπεται σε λεπτά και γίνεται έλεγχος αν βρίσκεται εντός του ζητούμενου εύρους 5 με 30 λεπτών. Αν βγαίνει εκτός αυτού δε γίνεται δεκτή η εισαγωγή και εμφανίζεται μήνυμα σφάλματος.

To trigger

```
CREATE TRIGGER block_fest_delete BEFORE DELETE ON FESTIVALS
FOR EACH ROW
BEGIN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FESTIVAL delete cancelled';
END
```

εκτελείται κατά την προσπάθεια διαγραφής ενός φεστιβάλ, καθώς αυτό απαγορεύεται και εμφανίζει μήνυμα σφάλματος.

To trigger

```
CREATE TRIGGER block_event_delete BEFORE DELETE ON FESTIVAL_EVENTS
FOR EACH ROW
BEGIN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FESTIVAL_EVENTS delete
  ↪  cancelled';
END
```

ενεργοποιείται κατά την διαγραφή μίας παράστασης, καθώς αυτό δεν επιτρέπεται.

To view

```
CREATE VIEW staff_coverage_view AS

SELECT
  fe.event_id,
  fe.stage_id,
  s.max_capacity,

  SUM(IF(sc.technical_id IS NOT NULL, 1, 0)) AS technical_assigned,
  SUM(IF(sc.staff_category_desc = 'Security', 1, 0))  AS security_assigned,
  SUM(IF(sc.staff_category_desc = 'Assistant', 1, 0)) AS general_assigned,


  CEIL(s.max_capacity * 0.05)                         AS security_required,
  CEIL(s.max_capacity * 0.02)                         AS general_required

FROM FESTIVAL_EVENTS fe
```

```
JOIN STAGES s ON fe.stage_id = s.stage_id
LEFT JOIN STAFF st ON st.event_id = fe.event_id       -- Left join because we
↪  need to see the event even if it has no staff
LEFT JOIN STAFF_CATEGORIES sc ON st.category_id = sc.staff_category_id

GROUP BY fe.event_id, fe.stage_id, s.max_capacity;
```

δημιουργεί μία συνολική αναφορά για την κάλυψη προσωπικού σε κάθε φεστιβάλ, με βάση τους διαθέσιμους υπαλλήλους και τη χωρητικότητα κάθε σκηνής. Αναλυτικότερα, στην αρχή μετρόνται οι υπάλληλοι που θα καλύψουν τις ανάγκες των εκδηλώσεων, ανά την κατήγορία που ανήκουν, δηλαδή τεχνικούς, υπαλλήλους ασφαλείας και γενικούς βοηθούς [technical_assigned, security_assigned, general_assigned]. Στη συνέχεια υπολογίζεται το απαιτούμενο προσωπικό με βάση τη χωρητικότητα κάθε σκηνής, το 5% της για προσωπικό ασφαλείας και το 2% της για υποστηρικτικό προσωπικό. Σημειώνεται πως το LEFT JOIN χρησιμοποιείται για να εμφανίζονται όλες οι εκδήλώσεις, ακόμα και αυτές στις οποίες δεν έχουν ανατεθεί υπάλληλοι. Η ομαδοποίηση γίνεται με βάση την παράσταση, τη σκηνή και τη μέγιστη χωρητικότητά της.

1.4 Ευρετήρια (Indexes)

Τα **ευρετήρια (indexes)** είναι δομές που επιταχύνουν τις αναζητήσεις και την εκτέλεση των ερωτημάτων, λειτουργώντας παρόμοια με ένα ευρετήριο βιβλίου. Στην παρούσα εργασία υλοποιούνται με B+ δέντρα, τα οποία εξυπηρετούν ιδανικά λόγω της ιεραρχικής τους δομής. Κάθε κόμβος τους περιέχει πολλαπλά κλειδιά κι δείκτες, επιτρέποντας λογαριθμικό χρόνο πρόσβασης (ακόμα και για μεγάλο όγκο δεδομένων) και αποδοτική διαχείριση ερωτημάτων, καθώς διατηρούνται ταξινομημένες οι τιμές και οι διασυνδέσεις μεταξύ των φύλλω. Στη συγκεκριμένη βάσης, έχουν σχεδιαστεί για τη βελτιστοποίηση των ερωτημάτων που εκτελούνται πιο συχνά, την επιτάχυνση των JOIN operations μεταξύ πινάκων και τη βελτίωση στις αναζητήσεις μέσω ξένων κλειδιών.

Επισημαίνεται ότι κατά τη δημιουργία της βάσης στο MySQL Workbench για κάθε πίνακα δημιουργείται αυτόματα ευρετήριο στο πρωτεύον κλειδί του.

Παρακάτω φαίνεται η αναλυτική εξήγηση των ευρετηρίων:

Σχετικά με φεστιβάλ και παραστάσεις:

```
CREATE INDEX idx_events_festivals ON FESTIVAL_EVENTS(festival_id);
```

για τη γρηγορότερη σύνδεση εκδηλώσεων.

```
CREATE INDEX idx_events_performances_events ON PERFORMANCES(event_id);
```

για επιτάχυνση αναζητήσεων παραστάσεων ανά εκδήλωση.

```
CREATE INDEX idx_events_tickets ON TICKETS(event_id);
```

για καλύτερη απόδοση στην αναζήτηση εισητηρίων ανά εκδήλωση.

```
CREATE INDEX idx_events_staff ON STAFF(event_id);
```

για την εύρεση προσωπικού που έχει ανατεθεί σε συγκεκριμένες εκδηλώσεις

Σχετικά με καλλιτέχνες και συγκροτήματα:

```
CREATE INDEX idx_ab_artist ON ARTISTS_X_BANDS(artist_id);
CREATE INDEX idx_ab_band ON ARTISTS_X_BANDS(band_id);
```

επιταχύνουν ερωτήματα τα οποία ζητούν τα συγκροτήματα που ανήκει ένας καλλιτέχνης και αντίστροφα, το σύνολο των καλλιτεχνών που αποτελούν ένα συγκρότημα. Βοηθούν στον γρηγορότερο έλεγχο για συμμετοχές και σχέσεις.

Σχετικά με κριτικές:

```
CREATE INDEX idx_reviews_performance ON REVIEWS(performance_id);
CREATE INDEX idx_reviews_visitors_performances ON REVIEWS(visitor_id,
↪  performance_id);
```

αφορούν κριτικές που έχουν γίνει σε παραστάσεις, τον έλεγχο αν ένας επισκέπτης έχει ήδη αξιολογήσει μία παράσταση και συνδέουν τους επισκέπτες με τις κριτικές τους.

Σχετικά με προσωπικό:

```
CREATE INDEX idx_staff_event_level ON STAFF(event_id, category_id);
```

επιταχύνει τα ερωτήματα τα οποία βρίσκουν υπαλλήλους ανά εκδήλωση και κατηγορία, ελέγχουν την κάλυψη του προσωπικού και αναλύουν την κατανομή του.

Σχετικά με παραστάσεις και εμφανιζόμενους καλλιτέχνες:

```
CREATE INDEX idx_events_performers ON PERFORMANCES(event_id, performer_id);
CREATE INDEX idx_performances_performers ON PERFORMANCES(performance_id,
↪  performer_id, is_solo);
```

συχνή χρήση για την σύνδεση παραστάσεων με καλλιτέχνες, ώστε να βρεθεί αν εμφανίζεται μεμονομένος καλλιτέχνης ή συγκρότημα.

Σχετικά με εισητήρια και επισκέπτες:

```
CREATE INDEX idx_tickets_visitors_events ON TICKETS(visitor_id, event_id);
```

καθιστά αποδοτικότερη την εύρεση των επισκεπτών στο μεγάλο όγκο δεδομένων των εισητηρίων.

1.5 Δεδομένα Ελέγχου

Για την επαλήθευση της ορθής λειτουργίας του συστήματος, κατασκευάστηκαν δεδομένα ελέγχου με χρήση της βιβλιοθήκης Faker της Python. Τα δεδομένα αυτά εισάγονται μέσω του αρχείου `load.sql` και περιλαμβάνουν ενδεικτικά 100 φεστιβάλ (εκ των οποίων και μελλοντικά), 100 τοποθεσίες, 300 παραστάσεις (events), 2000 εμφανίσεις (performances), 2000 μέλη προσωπικού, 300 καλλιτέχνες, 100 επισκέπτες, 4000 εισιτήρια και 300 αξιολογήσεις.

Κεφάλαιο 2

Ερωτήματα (Queries)

Τα **ερωτήματα (queries)** είναι εντολές που ζητούν την ανάκληση πληροφοριών από την βάση δεδομένων.

2.1 Q01

Ζητούνται τα έσοδα του φεστιβάλ από την πώληση εισιτηρίων ανά έτος παρέχοντας ανάλυση ανά είδος πληρωμής. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

Festival_Year	Credit_Card	Debit_Card	...	Cash	Total

Οι διαφορετικές χρονολογίες διεξαγωγής του φεστιβάλ βρίσκονται στη σχέση `FESTIVALS`. Οι παραστάσεις που πραγματοποιήθηκαν στο πλαίσιο ενός φεστιβάλ μπορούν να βρεθούν από τη σχέση `FESTIVAL_EVENTS` χρησιμοποιώντας το `festival_id` του φεστιβάλ αυτού. Τα εισιτήρια που πωλήθηκαν στο πλαίσιο μίας παράστασης ενός φεστιβάλ μπορούν να βρεθούν από τη σχέση `TICKETS` χρησιμοποιώντας το `event_id` της παράστασης αυτής. Τέλος, ο (περιγραφικός) τρόπος πληρωμής για κάποιο εισιτήριο μπορεί να βρεθεί από τη σχέση `PAYMENT_MENTHODS` χρησιμοποιώντας το `payment_method_id` του εισιτηρίου αυτού. Πραγματοποιούμε `JOIN` των παραπάνω σχέσεων ούτως ώστε να ενώσουμε τις πληροφορίες που χρειαζόμαστε.

Χρησιμοποιούμε τη συνοπτική συνάρτηση αθροίσματος (SUM) ούτως ώστε να υπολογίσουμε τα έσοδα για κάθε είδος πληρωμής αλλά και συνολικά.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SELECT
    f.festival_id,
    YEAR(f.date_starting) AS Festival_Year,

    -- Conditional SUMs for each payment method
```

```
    SUM(CASE WHEN pm.payment_method = 'Credit Card' THEN t.cost ELSE 0 END)
    ↪  AS Credit_Card,
    SUM(CASE WHEN pm.payment_method = 'Debit Card' THEN t.cost ELSE 0 END) AS
    ↪  Debit_Card,
    SUM(CASE WHEN pm.payment_method = 'Bank Transfer' THEN t.cost ELSE 0 END)
    ↪  AS Bank_Transfer,
    SUM(CASE WHEN pm.payment_method = 'Mobile Wallet' THEN t.cost ELSE 0 END)
    ↪  AS Mobile_Wallet,
    SUM(CASE WHEN pm.payment_method = 'Online Banking' THEN t.cost ELSE 0
    ↪  END) AS Online_Banking,
    SUM(CASE WHEN pm.payment_method = 'Prepaid Card' THEN t.cost ELSE 0 END)
    ↪  AS Prepaid_Card,
    SUM(CASE WHEN pm.payment_method = 'Cash' THEN t.cost ELSE 0 END) AS Cash,

    -- Total of all ticket sales
    SUM(t.cost) AS Total_Income

FROM FESTIVALS f
JOIN FESTIVAL_EVENTS fe ON fe.festival_id = f.festival_id
JOIN TICKETS t ON t.event_id = fe.event_id
JOIN PAYMENT_METHODS pm ON t.payment_method_id = pm.payment_method_id

GROUP BY f.festival_id, YEAR(f.date_starting)
ORDER BY Festival_Year DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q01_out.txt`.

2.2 Q02

Ζητούνται όλοι οι καλλιτέχνες που ανήκουν σε ένα συγκεκριμένο είδος (εδώ Jazz) με ένδειξη αν συμμετείχαν σε εκδηλώσεις του φεστιβάλ για το ένα συγκεκριμένο έτος (εδώ 2005). Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

artist_id	first_name	last_name	year	music_type	participated_in_year

Για την εκτέλεση του ερωτήματος χρησιμοποιούνται δύο παραμετρικές μεταβλητές: η `@choose_year`, η οποία καθορίζει το έτος ενδιαφέροντος, και η `@music_type`, που προσδιορίζει το μουσικό είδος των καλλιτεχνών που μας ενδιαφέρουν.

Οι διαφορετικοί καλλιτέχνες βρίσκονται στη σχέση `ARTISTS`. Το μουσικό είδος στο οποίο ανήκει ένας καλλιτέχνης μπορεί να βρεθεί από τη σχέση `ARTISTS_X_MUSIC` χρησιμοποιώντας το `artist_id` του καλλιτέχνη. Το είδος μουσικής ονομαστικά μπορεί να βρεθεί από τη σχέση `MUSIC_TYPES` χρησιμοποιώντας το `music_type_id` του μουσικού είδους αυτού. Όπως και πριν, πραγματοποιούμε `JOIN` των παραπάνω σχέσεων ούτως ώστε να ενώσουμε τις πληροφορίες που χρειαζόμαστε.

Το ερώτημα απαιτεί πρόσθετη επεξεργασία μέσω ένθετων υποερωτημάτων (subqueries) τα οποία ελέγχουν αν ο καλλιτέχνης συμμετείχε ως solo ή ως μέλος συγκροτήματος. Ουσιαστικά, μπορούμε

να βρούμε τον performer που συμμετείχε μία συγκεκριμένη χρονιά στο φεστιβάλ, είτε αυτός συμμετείχε ως solo καλλιτέχνης είτε ως μέλος συγκροτήματος, από το attribute `performance_id` της σχέσης `PERFORMANCE` (συνδέοντάς τον, όπως και προηγουμένως, με τις αντίστοιχες σχέσεις `EVENTS` και `FESTIVALS`). Στην περίπτωση που ο performer είναι συγκρότημα, απαιτείται επιπλέον σύνδεση με τη σχέση `ARTISTS_X_BANDS` ώστε να εντοπιστούν τα μέλη του.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SET @choose_year = 2005;
SET @music_type = 'Jazz';

SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    @choose_year AS year,
    mt.music_type,
    CASE
        WHEN participation.artist_id IS NOT NULL THEN 'Yes'
        ELSE 'No'
    END AS participated_in_year

FROM ARTISTS a
JOIN ARTISTS_X_MUSIC am ON am.artist_id = a.artist_id
JOIN MUSIC_TYPES mt ON mt.music_type_id = am.music_type_id
LEFT JOIN (
    -- For solo artists
    SELECT DISTINCT p.performer_id AS artist_id
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN FESTIVALS f ON f.festival_id = fe.festival_id
    WHERE p.is_solo = 1 AND YEAR(f.date_starting) = @choose_year

    UNION

    -- For bands
    SELECT DISTINCT ab.artist_id
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN FESTIVALS f ON f.festival_id = fe.festival_id
    JOIN ARTISTS_X_BANDS ab ON ab.band_id = p.performer_id
    WHERE p.is_solo = 0 AND YEAR(f.date_starting) = @choose_year
) AS participation ON participation.artist_id = a.artist_id

WHERE mt.music_type = @music_type
```

```
ORDER BY participated_in_year DESC, a.last_name;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q02_out.txt`.

2.3 Q03

Ζητούνται όλοι οι καλλιτέχνες που έχουν εμφανιστεί ως warm up περισσότερες από δύο φορές στο ίδιο φεστιβάλ. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

artist_id	first_name	last_name	festival_id	warmup_count

Οι εμφανίσεις εντοπίζονται μέσω της σχέσης `PERFORMANCES`, συνδέονται με τα αντίστοιχα events μέσω της σχέσης `FESTIVAL_EVENTS` και συνδέονται με τα αντίστοιχα φεστιβάλ μέσω της σχέσης `FESTIVALS`.

Για την υλοποίηση του ερωτήματος ελέγχεται η παρουσία του καλλιτέχνη σε παραστάσεις τύπου warm up, είτε συμμετείχε ως solo είτε ως μέλος συγκροτήματος. Στην περίπτωση που ο performer είναι συγκρότημα, απαιτείται επιπλέον σύνδεση με τη σχέση `ARTISTS_X_BANDS`, ώστε να εντοπιστούν τα επιμέρους μέλη του. Με την ένωση των δύο περιπτώσεων (`UNION ALL`), συγκεντρώνονται όλες οι εμφανίσεις τύπου `warm up` ανά καλλιτέχνη και φεστιβάλ.

Με ομαδοποίηση (`GROUP BY`) εντοπίζονται μόνο όσοι έχουν πραγματοποιήσει περισσότερες από δύο.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    f.festival_id,
    COUNT(*) AS warmup_count

FROM (
    -- For solo artists
    SELECT
        p.performer_id AS artist_id,
        fe.festival_id
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN PERFORMANCE_TYPES pt ON pt.performance_type_id =
    ↪  p.performance_type_id
    WHERE p.is_solo = 1 AND pt.performance_type = 'Warm Up'

    UNION ALL
```

```
    -- For bands
    SELECT
        ab.artist_id,
        fe.festival_id
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN PERFORMANCE_TYPES pt ON pt.performance_type_id =
    ↪  p.performance_type_id
    JOIN ARTISTS_X_BANDS ab ON ab.band_id = p.performer_id
    WHERE p.is_solo = 0 AND pt.performance_type = 'Warm Up'
) AS artist_festival_warmups

JOIN ARTISTS a ON a.artist_id = artist_festival_warmups.artist_id
JOIN FESTIVALS f ON f.festival_id = artist_festival_warmups.festival_id

GROUP BY a.artist_id, f.festival_id
HAVING COUNT(*) > 2
ORDER BY warmup_count DESC;
```

Επισημαίνεται ότι χρησιμοποιήθηκε UNION ALL ούτως ώστε να κρατήσουμε τα διπλότυπα καθότι θέλουμε τον συνολικό αριθμό των warm ups.

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q03_out.txt`.

2.4 Q04

Ζητείται να βρεθεί για κάποιο καλλιτέχνη ο μέσος όρος αξιολογήσεων (Ερμηνεία καλλιτεχνών) και εμφάνισης (Συνολική εντύπωση). Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

artist_id	first_name	last_name	avg_interpretation	avg_impression

Για την εκτέλεση του ερωτήματος χρησιμοποιείτα η παραμετρική μεταβλητή `@artist_id`, η οποία καθορίζει τον καλλιτέχνη για τον οποίο ενδιαφερόμαστε.

Οι αξιολογήσεις εντοπίζονται μέσω της σχέσης `REVIEWS` και συνδέονται με μία συγκεκριμένη παράσταση μέσω της σχέσης `PERFORMANCES`. Ένας καλλιτέχνης εντοπίζεται μέσω της σχέσης `ARTISTS` και σχετίζεται μέσω του attribute `performer_id` με μία συγκεκριμένη παράσταση.

Το τελικό αποτέλεσμα προκύπτει με χρήση της `AVG()` για κάθε τύπο βαθμολογίας και στρογγυλοποίηση με `ROUND(..., 2)`. Η ομαδοποίηση γίνεται ως προς τον `artist_id` για συγκέντρωση όλων των αξιολογήσεων του συγκεκριμένου καλλιτέχνη.
Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SET @artist_id = 41;

-- simple query
```

```sql
SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    ROUND(AVG(r.interpretation_rating), 2) AS avg_interpretation,
    ROUND(AVG(r.overall_impression_rating), 2) AS avg_impression
FROM REVIEWS r
JOIN PERFORMANCES p ON r.performance_id = p.performance_id
JOIN ARTISTS a ON p.performer_id = a.artist_id AND p.is_solo = 1
WHERE a.artist_id = @artist_id
GROUP BY a.artist_id;
```

Επιπλέον, με τη χρήση του `EXPLAIN ANALYZE` στην αρχή του query, μπορούμε να παρακολουθήσουμε το query plan και την εκτιμώμενη απόδοση του ερωτήματος.

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q06_out.txt`.

2.4.1 Εναλλακτικό Query Plan

Για διαφορετικό query plan θα αξιοποιήσουμε `FORCE INDEX` και `STRAIGHT_JOIN` για να θέσουμε την σειρά που γίνονται τα joins. Το query φαίνεται παρακάτω:

```sql
USE pulse_uni_db;

SET @artist_id = 41;

-- query with FORCE INDEX
EXPLAIN ANALYZE
SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    ROUND(AVG(r.interpretation_rating), 2) AS avg_interpretation,
    ROUND(AVG(r.overall_impression_rating), 2) AS avg_impression
FROM REVIEWS r FORCE INDEX (idx_reviews_performance)
JOIN PERFORMANCES p FORCE INDEX (idx_performances_performer)
    ON r.performance_id = p.performance_id
STRAIGHT_JOIN ARTISTS a ON p.performer_id = a.artist_id AND p.is_solo = 1
WHERE a.artist_id = @artist_id
GROUP BY a.artist_id;
```

Επιπλέον, με τη χρήση του `EXPLAIN ANALYZE` στην αρχή του query, μπορούμε να παρακολουθήσουμε το query plan και την εκτιμώμενη απόδοση του ερωτήματος.

2.4.2 Traces

Το trace του αρχικού query:

```
EXPLAIN
-> Group aggregate: avg(r.overall_impression_rating), avg(r.interpretation_rating)  (cost=1.09 rows=0.92) (actual time=0.0437..0.0438 rows=1 loops=1)
-> Nested loop inner join  (cost=1 rows=0.92) (actual time=0.0222..0.0267 rows=1 loops=1)
-> Filter: (p.is_solo = 1)  (cost=0.333 rows=0.8) (actual time=0.0101..0.0118 rows=3 loops=1)
-> Covering index lookup on p using idx_performances_performers (performer_id=(@artist_id))  (cost=0.333 rows=8) (actual time=0.00695..0.00954 rows=8 loops=1)
-> Index lookup on r using idx_reviews_performance (performance_id=p.performance_id)  (cost=0.862 rows=1.15) (actual time=0.00386..0.00425 rows=0.333 loops=3)
```

Το trace του εναλλακτικού query:

```
EXPLAIN
-> Group aggregate: avg(r.overall_impression_rating), avg(r.interpretation_rating)  (cost=1.38 rows=0.92) (actual time=0.0897..0.0898 rows=1 loops=1)
-> Nested loop inner join  (cost=1.29 rows=0.92) (actual time=0.079..0.0834 rows=1 loops=1)
-> Nested loop inner join  (cost=1 rows=0.92) (actual time=0.0684..0.0726 rows=1 loops=1)
-> Filter: (p.is_solo = 1)  (cost=0.333 rows=0.8) (actual time=0.0347..0.0368 rows=3 loops=1)
-> Covering index lookup on p using idx_performances_performers (performer_id=(@artist_id))  (cost=0.333 rows=8) (actual time=0.0302..0.0334 rows=8 loops=1)
-> Index lookup on r using idx_reviews_performance (performance_id=p.performance_id)  (cost=0.862 rows=1.15) (actual time=0.0109..0.0112 rows=0.333 loops=3)
-> Single-row index lookup on a using PRIMARY (artist_id=(@artist_id))  (cost=0.326 rows=1) (actual time=0.00909..0.00916 rows=1 loops=1)
```

Τα traces βρίσκονται στα αρχεία `Q04_trace.txt` και `Q04_alt_trace.txt` αντίστοιχα.

2.4.3 Συμπεράσματα

Παρατηρούμε πως η "αυστηρή" `STRAIGHT_JOIN` επιφέρει αλλαγή στο query plan καθώς εκετελείται ένα παραπάνω nested join. Επιπλέον δεν εκτελείται filter για το `is_solo` λόγω του αντίστοιχου index. Παρ'όλα αυτά, οι χρόνοι εκτέλεσης είναι πολύ μικροί για να έχουμε έγκυρη σύγκριση (μεταξύ πειραμάτων είχαμε αλλαγή στο γρηγορότερο query). Αυτό συμβαίνει καθώς το συγκεκριμένο query αναζητά σε σχετικά λίγα δεδομένα.

2.5 Q05

Ζητούνται οι καλλιτέχνες ηλικίας μικρότερης των 30 ετών που έχουν συμμετάσχει τις περισσότερες φορές στο φεστιβάλ. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

artist_id	first_name	last_name	birthday	festival_count

Οι εμφανίσεις εντοπίζονται μέσω της σχέσης `PERFORMANCES`, συνδέονται με τα αντίστοιχα events μέσω της σχέσης `FESTIVAL_EVENTS` και συνδέονται με τα αντίστοιχα φεστιβάλ μέσω της σχέσης `FESTIVALS`.

Το ερώτημα απαιτεί πρόσθετη επεξεργασία μέσω ένθετων υποερωτημάτων (subqueries) τα οποία ελέγχουν αν ο καλλιτέχνης συμμετείχε ως solo ή ως μέλος συγκροτήματος. Ουσιαστικά, μπορούμε να βρούμε τον performer που συμμετείχε μία συγκεκριμένη χρονιά στο φεστιβάλ, είτε αυτός συμμετείχε ως solo καλλιτέχνης είτε ως μέλος συγκροτήματος, από το attribute `performance_id` της

σχέσης PERFORMANCE (συνδέοντάς τον, όπως και προηγουμένως, με τις αντίστοιχες σχέσεις EVENTS και FESTIVALS). Στην περίπτωση που ο performer είναι συγκρότημα, απαιτείται επιπλέον σύνδεση με τη σχέση ARTISTS_X_BANDS ώστε να εντοπιστούν τα μέλη του. Με την ένωση των δύο περιπτώσεων (UNION ALL), συγκεντρώνονται όλες οι εμφανίσεις ανά καλλιτέχνη και φεστιβάλ.

Το παραπάνω αποτέλεσμα συνδέεται με τον πίνακα ARTISTS μέσω LEFT JOIN, ώστε να συμπεριληφθούν όλοι οι καλλιτέχνες, ακόμη και εκείνοι που ενδεχομένως να μην έχουν ξανασυμμετάσχει.

Ο ηλικιακός περιορισμός εφαρμόζεται με χρήση της συνάρτησης TIMESTAMPDIFF, η οποία υπολογίζει τη διαφορά σε έτη μεταξύ της ημερομηνίας γέννησης του καλλιτέχνη (birthday) και της ημερομηνίας εμφάνισής του στο φεστιβάλ (event_date). Με τη χρήση του φίλτρου TIMESTAMPDIFF(YEAR, a.birthday, artist_festivals.event_date) < 30 AND > 0, διασφαλίζεται ότι περιλαμβάνονται μόνο συμμετοχές καλλιτεχνών ηλικίας μεταξύ 1 και 29 ετών.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    a.birthday,
    COUNT(DISTINCT artist_festivals.festival_id) AS festival_count

FROM ARTISTS a
LEFT JOIN (
    -- For solo artists
    SELECT p.performer_id AS artist_id, fe.festival_id, fe.event_date
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN FESTIVALS f on fe.festival_id = f.festival_id
    WHERE p.is_solo = 1

    UNION ALL

    -- For bands
    SELECT ab.artist_id as artist_id, fe.festival_id, fe.event_date
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN ARTISTS_X_BANDS ab ON ab.band_id = p.performer_id
    JOIN FESTIVALS f on fe.festival_id = f.festival_id
    WHERE p.is_solo = 0
) AS artist_festivals ON artist_festivals.artist_id = a.artist_id

WHERE TIMESTAMPDIFF(YEAR, a.birthday, artist_festivals.event_date) < 30 AND
↪  TIMESTAMPDIFF(YEAR, a.birthday, artist_festivals.event_date) > 0
GROUP BY a.artist_id
```

```
ORDER BY festival_count DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q05_out.txt`.

2.6 Q06

Ζητείται να βρεθούν για έναν επισκέπτη όλες παραστάσεις που έχει παρακολουθήσει και ο μέσος όρος της αξιολόγησης του. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

visitor_id	performance_id	event_id	is_solo	avg_rating

Για την εκτέλεση του ερωτήματος χρησιμοποιείται η παραμετρική μεταβλητή `@visitor_id` για να καθοριστεί ο επισκέπτης για τον οποίο θέλουμε να ανακτήσουμε τις πληροφορίες.

Οι αξιολογήσεις προέρχονται από τη σχέση `REVIEWS`, ενώ οι πληροφορίες για την παράσταση και τον τύπο εμφάνισης (solo ή όχι) προέρχονται από τη σχέση `PERFORMANCES`. Οι δύο σχέσεις συνδέονται μέσω του `performance_id`.

Ο μέσος όρος της συνολικής αξιολόγησης υπολογίζεται αθροίζοντας τα πέντε επιμέρους πεδία και διαιρώντας το αποτέλεσμα με το 5. Το `AVG(...)` χρησιμοποιείται σε συνδυασμό με `ROUND(..., 2)` για στρογγυλοποίηση σε δύο δεκαδικά ψηφία.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SET @visitor_id = 42;

-- simple query
SELECT
    r.visitor_id,
    r.performance_id,
    p.event_id,
    p.is_solo,
    ROUND(AVG(
        r.interpretation_rating +
        r.sound_lighting_rating +
        r.stage_presence_rating +
        r.organization_rating +
        r.overall_impression_rating
    ) / 5, 2) AS avg_rating
FROM REVIEWS r
JOIN PERFORMANCES p ON r.performance_id = p.performance_id
WHERE r.visitor_id = @visitor_id
GROUP BY r.visitor_id, r.performance_id, p.event_id, p.is_solo;
```

Επιπλέον, με τη χρήση του `EXPLAIN ANALYZE` στην αρχή του query, μπορούμε να παρακολουθήσουμε το query plan και την εκτιμώμενη απόδοση του ερωτήματος.

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q06_out.txt`.

2.6.1 Εναλλακτικό Query Plan

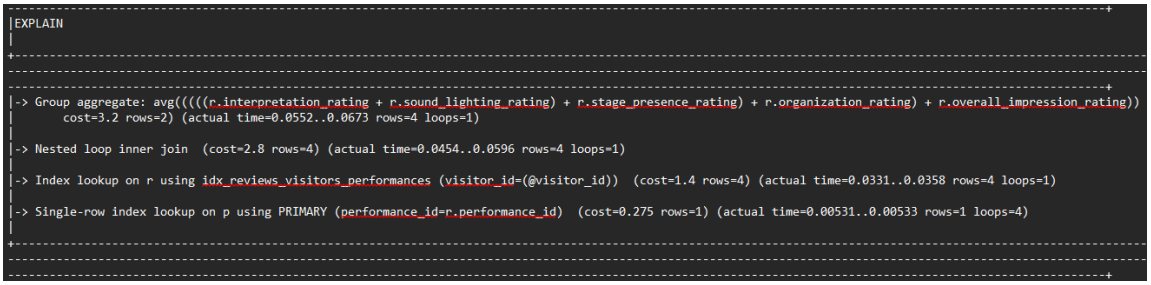
Για να αλλάξουμε το query plan, θα χρησιμοποιήσουμε `FORCE INDEX()` πάνω στο index `idx_reviews_visitors_performances`. Το query διατυπώνεται ως εξής:

```sql
USE pulse_uni_db;

SET @visitor_id = 42;

-- query with FORCE INDEX
EXPLAIN ANALYZE
SELECT
    r.visitor_id,
    r.performance_id,
    p.event_id,
    p.is_solo,
    ROUND(AVG(
        r.interpretation_rating +
        r.sound_lighting_rating +
        r.stage_presence_rating +
        r.organization_rating +
        r.overall_impression_rating
    ) / 5, 2) AS avg_rating
FROM REVIEWS r FORCE INDEX (idx_reviews_visitors_performances)
JOIN PERFORMANCES p ON r.performance_id = p.performance_id
WHERE r.visitor_id = @visitor_id
GROUP BY r.visitor_id, r.performance_id;
```

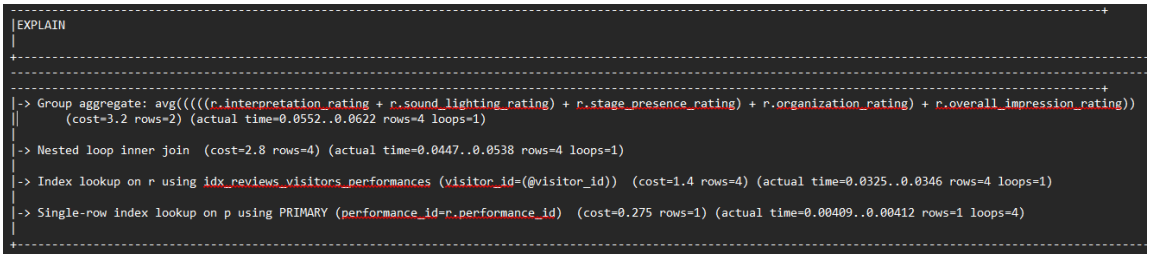
Επιπλέον, με τη χρήση του `EXPLAIN ANALYZE` στην αρχή του query, μπορούμε να παρακολουθήσουμε το query plan και την εκτιμώμενη απόδοση του ερωτήματος.

2.6.2 Traces

Το trace αρχικού query:



Το trace του εναλλακτικού query:



Τα traces βρίσκονται στα αρχεία `Q06_trace.txt` και `Q06_alt_trace.txt` αντίστοιχα.

2.6.3 Συμπεράσματα

Όπως φαίνεται από τα traces ο optimizer δεν βρίσκει διαφορετικό query plan με χρήση `FORCE INDEX ()`. Αυτό συμβαίνει καθώς ο index είναι ήδη βέλτιστος, εξ' αυτού και οι παρεμφερείς χρόνοι στους οποίους εκτελέστηκαν τα queries κατά τον πειραματισμό μας.

2.7 Q07

Ζητείται να βρεθεί ποιο φεστιβάλ είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

festival_id	avg_experience

Τα μέλη του προσωπικού προκύπτουν από τη σχέση `STAFF`, κατηγοριοποιούνται μέσω της σχέσης `STAFF_CATEGORIES`, και συνδέονται με το αντίστοιχο επίπεδο εμπειρίας τους μέσω της σχέσης `LEVELS_OF_EXPERTISE`. Κάθε μέλος του προσωπικού συνδέεται με ένα event μέσω της σχέσης `FESTIVAL_EVENTS`, και τελικά με το φεστιβάλ στο οποίο ανήκει, μέσω της σχέσης `FESTIVALS`.

Το ερώτημα απαιτεί τη δημιουργία προσωρινής όψης (`CTE` - Common Table Expression) με το όνομα `festival_experience`, στην οποία αποθηκεύεται ο μέσος όρος επιπέδου εμπειρίας του τεχνικού προσωπικού που εργάστηκε στα events κάθε φεστιβάλ.

Το φίλτρο `sc.technical_id IS NOT NULL` εξασφαλίζει ότι λαμβάνεται υπόψη μόνο το προσωπικό που ανήκει σε τεχνικές κατηγορίες (π.χ. ηχολήπτες, φωτιστές, κ.λπ.), το οποίο είναι σχετικό με τον υπολογισμό εμπειρίας σε τεχνικό επίπεδο.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

WITH festival_experience AS(
    SELECT
        f.festival_id,
        AVG(l.level_id) AS avg_experience
    FROM STAFF s
```

```
    JOIN STAFF_CATEGORIES sc ON s.category_id = sc.staff_category_id
    JOIN LEVELS_OF_EXPERTISE l ON s.level_id = l.level_id
    JOIN FESTIVAL_EVENTS fe ON s.event_id = fe.event_id
    JOIN FESTIVALS f ON fe.festival_id = f.festival_id
    WHERE sc.technical_id IS NOT NULL
    GROUP BY f.festival_id
)

SELECT *
FROM festival_experience
WHERE avg_experience = (
    SELECT MIN(avg_experience) FROM festival_experience
)
ORDER BY festival_id;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q07_out.txt`.

2.8 Q08

Ζητείται να βρεθεί το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία (εδώ στις 2022-07-29). Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

staff_id	first_name	last_name

Για την εκτέλεση του ερωτήματος χρησιμοποιείται μία παραμετρική μεταβλητή: η `@event_date`, η οποία καθορίζει την ημέρα ενδιαφέροντος.

Τα μέλη του προσωπικού προκύπτουν από τη σχέση `STAFF` και κατηγοριοποιούνται μέσω της σχέσης `STAFF_CATEGORIES`.

Αρχικά, ελέγχεται εάν το μέλος του προσωπικού που εξετάζεται είναι τεχνικό ή όχι μέσω του `technical_id`. Το ερώτημα απαιτεί, επίσης, πρόσθετη επεξεργασία μέσω ένθετου υποερωτήματος (subquery) το οποίο ελέγχει αν το συγκεκριμένο μέλος προσωπικού είχε προγραμματισμένη εργασία τη συγκεκριμένη ημερομηνία.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SET @event_date = '2022-07-29';

SELECT
    s.staff_id,
    s.first_name,
    s.last_name

FROM STAFF s
```

```
JOIN STAFF_CATEGORIES sc ON s.category_id = sc.staff_category_id

WHERE sc.technical_id IS NULL
  AND s.staff_id NOT IN (
    SELECT s2.staff_id
    FROM STAFF s2
    JOIN FESTIVAL_EVENTS fe ON s2.event_id = fe.event_id
    WHERE fe.event_date = @event_date
)
ORDER BY s.last_name;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q08_out.txt`.

2.9 Q09

Ζητείται να βρεθούν ποιοι επισκέπτες έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με περισσότερες από 3 παρακολουθήσεις. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

festival_id	visitor_id	first_name	last_name	year	num_performances

Τα εισιτήρια προέρχονται από τη σχέση `TICKETS` και οι επισκέπτες από τη σχέση `VISITORS` μέσω του `visitor_id`. Η σύνδεση με τις παραστάσεις και τα φεστιβάλ γίνεται μέσω της σχέσης `FESTIVAL_EVENTS`. Για να εξασφαλίσουμε ότι οι επισκέπτες έχουν παρακολουθήσει την εκάστοτε παράσταση ελέγχουμε εάν το αντίστοιχο εισιτήριο έχει σκαναριστεί μέσω του attribute (`is_scanned = 1`).

Τα αποτελέσματα για κάθε επισκέπτη και έτος διεξαγωγής φεστιβάλ αθροίζονται και, στη συνέχεια, κρατούνται μόνο οι επισκέπτες που παρακολούθησαν περισσότερες από τρεις παραστάσεις σε συγκεκριμένο φεστιβάλ μέσω του `HAVING COUNT(*) > 3`.

Η ταξινόμηση γίνεται κατά φθίνουσα σειρά πλήθους παραστάσεων (`num_performances`), ώστε οι πιο ενεργοί επισκέπτες να εμφανίζονται πρώτοι.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SELECT
    fe.festival_id,
    t.visitor_id,
    v.first_name,
    v.last_name,
    YEAR(fe.event_date) AS year,
    COUNT(*) AS num_performances

FROM TICKETS t
JOIN VISITORS v on t.visitor_id = v.visitor_id
```

```
JOIN FESTIVAL_EVENTS fe ON t.event_id = fe.event_id

WHERE t.is_scanned = 1

GROUP BY t.visitor_id, YEAR(fe.event_date), fe.festival_id
HAVING COUNT(*) > 3
ORDER BY num_performances DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q09_out.txt`.

2.10 Q10

Ζητείται να βρεθούν τα 3 κορυφαία (top-3) ζεύγη μουσικών ειδών (π.χ. ροκ, τζαζ) που είναι κοινά στους καλλιτέχνες που εμφανίστηκαν σε ένα φεστιβάλ. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

festival_id	type1	type2	pair_count

Το ερώτημα απαιτεί τη δημιουργία δύο προσωρινών όψεων (CTEs - Common Table Expressions): η `artist_types`, η οποία αντιστοιχίζει κάθε καλλιτέχνη στα μουσικά είδη στα οποία ανήκει με βάση τις σχέσεις `ARTISTS_X_MUSIC` και `MUSIC_TYPES`, και η `festival_artists`, η οποία εντοπίζει όλους τους καλλιτέχνες που συμμετείχαν σε φεστιβάλ είτε ως solo είτε ως μέλη συγκροτήματος. Η επιλογή του σωστού id με χρήση της `CASE`.

Καθώς ένας καλλιτέχνης μπορεί να ανήκει σε περισσότερα από ένα μουσικά είδη, πραγματοποιείται `JOIN` μεταξύ των δύο ειδών με την συνθήκη `at1.music_type < at2.music_type`, ώστε να αποφεύγεται η διπλή καταμέτρηση συμμετρικών ζευγών.

Τα αποτελέσματα ομαδοποιούνται ως προς το `festival_id` και το ζεύγος μουσικών τύπων, με χρήση των συναρτήσεων `LEAST()` και `GREATEST()` για τη σταθερή σειρά εμφάνισης των ζευγών.

Η ταξινόμηση γίνεται βάσει του πλήθους (`pair_count`) και του πρώτου μουσικού είδους αλφαβητικά και επιστρέφονται τα τρία πιο συχνά εμφανιζόμενα ζεύγη.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

WITH artist_types AS(
    SELECT DISTINCT am.artist_id, mt.music_type
    FROM ARTISTS_X_MUSIC am
    JOIN MUSIC_TYPES mt ON am.music_type_id = mt.music_type_id
),
festival_artists AS(
    SELECT DISTINCT
        CASE
            WHEN p.is_solo = 1 THEN p.performer_id
            ELSE ab.artist_id
```

```
        END AS artist_id,
        fe.festival_id as festival_id
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe on p.event_id = fe.event_id
    LEFT JOIN ARTISTS_X_BANDS ab on ab.band_id = p.performer_id
)

SELECT
    fa.festival_id,
    LEAST(at2.music_type,at1.music_type) AS type1,
    GREATEST(at2.music_type,at1.music_type) AS type2,
    COUNT(*) AS pair_count
FROM artist_types at1
JOIN artist_types at2 ON at2.artist_id = at1.artist_id AND at1.music_type <
↪  at2.music_type
JOIN festival_artists fa on at1.artist_id = fa.artist_id
GROUP BY fa.festival_id, type1, type2
ORDER BY pair_count DESC
LIMIT 3
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q10_out.txt`.

2.11 Q11

Ζητείται να βρεθούν όλοι οι καλλιτέχνες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον καλλιτέχνη με τις περισσότερες συμμετοχές στο φεστιβάλ. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

artist_id	first_name	last_name	total_participations

Το ερώτημα απαιτεί τη δημιουργία δύο προσωρινών όψεων (CTEs - Common Table Expressions): η `artist_participations`, η οποία εντοπίζει όλες τις συμμετοχές καλλιτεχνών σε φεστιβάλ (είτε συμμετείχαν ως solo είτε ως μέλος συγκροτήματος) και η `total_counts`, που συγκεντρώνει τον συνολικό αριθμό συμμετοχών ανά καλλιτέχνη, χρησιμοποιώντας `SUM()` επί των μετρήσεων που παρήχθησαν από την `artist_participations`.

Τα δεδομένα συνολικού πλήθους συμμετοχών που προκύπτουν συγκρίνονται με τη μέγιστη τιμή όλων των συμμετοχών, μειωμένη κατά 5 χάρη στη συνθήκη `WHERE total_participations <= MAX(...) - 5`.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

WITH artist_participations AS (
    SELECT p.performer_id AS artist_id, COUNT(DISTINCT fe.festival_id) AS
    ↪  festivals_count
```

```
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    WHERE p.is_solo = 1
    GROUP BY p.performer_id

    UNION ALL

    SELECT ab.artist_id, COUNT(DISTINCT fe.festival_id) AS festivals_count
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON fe.event_id = p.event_id
    JOIN ARTISTS_X_BANDS ab ON ab.band_id = p.performer_id
    WHERE p.is_solo = 0
    GROUP BY ab.artist_id
),
total_counts AS (
    SELECT artist_id, SUM(festivals_count) AS total_participations
    FROM artist_participations
    GROUP BY artist_id
)

SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    tc.total_participations
FROM total_counts tc
JOIN ARTISTS a ON a.artist_id = tc.artist_id
WHERE tc.total_participations <= (
    SELECT MAX(total_participations) FROM total_counts
) - 5
ORDER BY tc.total_participations DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q11_out.txt`.

2.12 Q12

Ζητείται να βρεθεί το προσωπικό που απαιτείται για κάθε ημέρα του φεστιβάλ, παρέχοντας ανάλυση ανά κατηγορία (τεχνικό, προσωπικό ασφαλείας, βοηθητικό προσωπικό). Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

event_date	technical_count	security_count	assistant_count	total_count

Οι παραστάσεις προέρχονται από τη σχέση FESTIVAL_EVENTS και συνδέονται με τις σκηνές μέσω της σχέσης STAGES, προκειμένου να ληφθεί υπόψη η χωρητικότητα κάθε σκηνής στην εκτίμηση των απαιτήσεων προσωπικού.

Για κάθε παράσταση που πραγματοποιείται σε συγκεκριμένη ημερομηνία, ορίζεται σταθερός αριθμός 5 τεχνικών ανά event, με χρήση της μεταβλητής `@tech_req`. Οι ανάγκες για υπεύθυνους ασφαλείας και βοηθητικό προσωπικό καθορίζονται ως ποσοστά της μέγιστης χωρητικότητας της σκηνής: 5% και 2% αντίστοιχα. Για να διασφαλιστεί ακέραιος αριθμός ατόμων, χρησιμοποιείται η συνάρτηση `CEIL()` για στρογγυλοποίηση προς τα πάνω.

Τα αποτελέσματα ομαδοποιούνται ως προς το `event_date` και ταξινομούνται κατά φθίνουσα σειρά ημερομηνίας.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

SET @tech_req = 5;

SELECT
    fe.event_date,
    SUM(@tech_req) AS technical_count,
    SUM(CEIL(st.max_capacity * 0.05)) AS security_count,
    SUM(CEIL(st.max_capacity * 0.02)) AS assistant_count,
    SUM(@tech_req+CEIL(st.max_capacity * 0.05)+CEIL(st.max_capacity * 0.02))
    ↪  as total_count
FROM FESTIVAL_EVENTS fe
JOIN STAGES st ON st.stage_id = fe.stage_id

GROUP BY fe.event_date
ORDER BY fe.event_date DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q12_out.txt`.

2.13 Q13

Ζητείται να βρεθούν οι καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

artist_id	first_name	last_name	continent_count

Το ερώτημα απαιτεί τη δημιουργία της προσωρινής όψης (CTE - Common Table Expression) `artist_continents`, η οποία αντιστοιχίζει κάθε καλλιτέχνη με τις ηπείρους όπου έχει εμφανιστεί είτε ως solo είτε ως μέλος συγκροτήματος.

Οι εμφανίσεις των καλλιτεχνών εντοπίζονται από τη σχέση `PERFORMANCES`, ενώ η σύνδεση με τις αντίστοιχες παραστάσεις και φεστιβάλ γίνεται μέσω των σχέσεων `FESTIVAL_EVENTS` και `FESTIVALS`. Η τοποθεσία διεξαγωγής του κάθε φεστιβάλ αντλείται από τη σχέση `LOCATIONS`, από όπου προκύπτει και η αντίστοιχη ήπειρος (`continent`). Για να ληφθούν υπόψη και οι συμμετοχές μέσω συγκροτήματος, πραγματοποιείται σύνδεση με τη σχέση `ARTISTS_X_BANDS`, επιτρέποντας την αναγνώριση των μελών του κάθε συγκροτήματος.

Για κάθε καλλιτέχνη υπολογίζεται το πλήθος των μοναδικών ηπείρων και με τη συνθήκη `HAVING COUNT(...) >= 3` τα αποτελέσματα περιορίζονται μόνο σε όσους έχουν εμφανιστεί σε φεστιβάλ που διεξήχθησαν σε τρεις ή περισσότερες διαφορετικές ηπείρους. Τέλος, τα αποτελέσματα ταξινομούνται κατά φθίνουσα σειρά πλήθους ηπείρων.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

WITH artist_continents AS (
    SELECT DISTINCT
        p.performer_id AS artist_id,
        l.continent
    FROM PERFORMANCES p
    JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
    JOIN FESTIVALS f ON fe.festival_id = f.festival_id
    JOIN LOCATIONS l ON f.location_id = l.location_id
    WHERE p.is_solo = 1

    UNION

    SELECT DISTINCT
        ab.artist_id,
        l.continent
    FROM PERFORMANCES p
    JOIN ARTISTS_X_BANDS ab ON ab.band_id = p.performer_id
    JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
    JOIN FESTIVALS f ON fe.festival_id = f.festival_id
    JOIN LOCATIONS l ON f.location_id = l.location_id
    WHERE p.is_solo = 0
)
SELECT
    a.artist_id,
    a.first_name,
    a.last_name,
    COUNT(DISTINCT ac.continent) AS continent_count
FROM artist_continents ac
JOIN ARTISTS a ON a.artist_id = ac.artist_id
GROUP BY a.artist_id
HAVING COUNT(DISTINCT ac.continent) >= 3
ORDER BY continent_count DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q13_out.txt`.

2.14 Q14

Ζητείται να βρεθούν τα μουσικά είδη που είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές με τουλάχιστον τρεις εμφανίσεις ανά έτος. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

music_type	year1	year2	appearance_count

Το ερώτημα απαιτεί τη δημιουργία δύο προσωρινών όψεων (CTEs - Common Table Expressions): η `appearances_per_year`, η οποία υπολογίζει για κάθε είδος μουσικής και για κάθε έτος, τον αριθμό εμφανίσεων καλλιτεχνών που ανήκουν σε αυτό το είδος είτε εμφανίστηκαν ως solo είτε ως μέλος συγκροτήματος και ελέγχει αν οι εμφανίσεις ήταν τουλάχιστον τρεις και η `consecutive_years`, που εντοπίζει τα είδη μουσικής για τα οποία παρατηρείται το ίδιο πλήθος εμφανίσεων σε δύο συνεχόμενα έτη.

Τα αποτελέσματα ταξινομούνται ως προς το πλήθος εμφανίσεων κατά φθίνουσα σειρά.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;

WITH appearances_per_year AS (
    SELECT
        mt.music_type_id,
        mt.music_type,
        YEAR(f.date_starting) AS year,
        COUNT(*) AS appearance_count
    FROM PERFORMANCES p
    JOIN ARTISTS a ON p.performer_id = a.artist_id AND p.is_solo = 1
    JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
    JOIN FESTIVALS f ON fe.festival_id = f.festival_id
    JOIN ARTISTS_X_MUSIC am on a.artist_id = am.artist_id
    JOIN MUSIC_TYPES mt ON am.music_type_id = mt.music_type_id
    GROUP BY mt.music_type_id, mt.music_type, YEAR(f.date_starting)
    HAVING COUNT(*) >= 3

    UNION

    SELECT
        mt.music_type_id,
        mt.music_type,
        YEAR(f.date_starting) AS year,
        COUNT(*) AS appearance_count
    FROM PERFORMANCES p
    JOIN ARTISTS_X_BANDS ab ON p.performer_id = ab.band_id AND p.is_solo = 0
    JOIN ARTISTS a ON ab.artist_id = a.artist_id
    JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
    JOIN FESTIVALS f ON fe.festival_id = f.festival_id
```

```
    JOIN ARTISTS_X_MUSIC am on a.artist_id = am.artist_id
    JOIN MUSIC_TYPES mt ON am.music_type_id = mt.music_type_id
    GROUP BY mt.music_type_id, mt.music_type, YEAR(f.date_starting)
    HAVING COUNT(*) >= 3
),
consecutive_years AS (
    SELECT
        a1.music_type,
        a1.year AS year1,
        a2.year AS year2,
        a1.appearance_count
    FROM appearances_per_year a1
    JOIN appearances_per_year a2
      ON a1.music_type_id = a2.music_type_id
     AND a2.year = a1.year + 1
     AND a1.appearance_count = a2.appearance_count
)
SELECT
    music_type,
    year1,
    year2,
    appearance_count
FROM consecutive_years
ORDER BY appearance_count DESC;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q14_out.txt`.

2.15 Q15

Ζητούνται οι top-5 επισκέπτες που έχουν δώσει συνολικά την υψηλότερη βαθμολόγηση σε ένα καλλιτέχνη. Ουσιαστικά, θέλουμε ως αποτέλεσμα έναν πίνακα της μορφής:

visitor_name	artist_name	total_score

Το ερώτημα απαιτεί τη δημιουργία τριών προσωρινών όψεων (CTEs - Common Table Expressions): η `solo_scores`, η οποία υπολογίζει τις συνολικές βαθμολογίες (άθροισμα `interpretation_rating` και `overall_impression_rating`) που έδωσε κάθε επισκέπτης σε solo καλλιτέχνες, η `band_scores`, που λειτουργεί με αντίστοιχο τρόπο για περιπτώσεις όπου ο καλλιτέχνης συμμετείχε ως μέλος συγκροτήματος και η `all_scores`, που προκύπτει ως ένωση `UNION ALL` των δύο προηγούμενων

Τα αποτελέσματα ομαδοποιούνται ανά επισκέπτη και καλλιτέχνη και υπολογίζεται το συνολικό σκορ με `SUM(score)`. Τελικά, επιστρέφονται οι πέντε υψηλότερες περιπτώσεις.

Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

```
USE pulse_uni_db;
```

```sql
WITH solo_scores AS(
    SELECT
        r.visitor_id AS visitor_id,
        p.performer_id AS artist_id,
        SUM(r.interpretation_rating + r.overall_impression_rating) AS score
    FROM REVIEWS r
    JOIN PERFORMANCES p ON r.performance_id = p.performance_id
    WHERE p.is_solo = 1
    GROUP BY visitor_id, artist_id
),
band_scores AS(
    SELECT
        r.visitor_id AS visitor_id,
        ab.artist_id AS artist_id,
        SUM(r.interpretation_rating + r.overall_impression_rating) AS score
    FROM REVIEWS r
    JOIN PERFORMANCES p ON r.performance_id = p.performance_id
    JOIN ARTISTS_X_BANDS ab ON ab.band_id = p.performer_id
    WHERE p.is_solo = 0
    GROUP BY visitor_id, ab.artist_id
),
all_scores AS(
    SELECT * FROM band_scores
    UNION ALL
    SELECT * FROM solo_scores
)
SELECT
    CONCAT(v.first_name, ' ', v.last_name) AS visitor_name,
    CONCAT(a.first_name, ' ', a.last_name) AS artist_name,
    SUM(score) AS total_score
FROM all_scores s
JOIN VISITORS v ON s.visitor_id = v.visitor_id
JOIN ARTISTS a ON s.artist_id = a.artist_id
GROUP BY s.visitor_id, s.artist_id
ORDER BY total_score DESC
LIMIT 5;
```

Το αποτέλεσμα του ερωτήματος βρίσκεται στο αρχείο `Q15_out.txt`.

Παράρτημα Α΄

Παραδοχές

Για τη ζητούμενη βάση δεδομένων έχουν γίνει κάποιες συγκεκριμένες παραδοχές, με τις οποίες έγινε η υλοποίηση και καθορίστηκε η λειτουργία της. Για την αναλυτικότερη επεξήγησή της παρατίθενται παρακάτω:

- Η ουρά για την προπώληση των εισητηρίων ανοίγει όταν τελειώσουν όλα τα εισητήρια και όχι κάποια συγκεκριμένη κατηγορία τους.
- Το υποστηρικτικό προσωπικό είναι το μη τεχνικό, δηλαδή βοηθητικό και ασφαλείας.
- Στον πίνακα των Performances έχει γίνει εισαγωγή του πεδίου `order_in_show`, και έχει υλοποιηθεί το trigger που ελέγχει πως τα διαλείμματα είναι εντός του εύρους των 5 έως 30 λεπτών, επομένως εξασφαλίζεται πως δεν υπάρχει overlap μεταξύ των εμφανίσεων.
- Το duration στα πεδία των πινάκων μετράται σε λεπτά.
- Θεωρούμε πως ένα event κατοχυρώνει μία σκηνή για όλη τη δεδομένη ημέρα (επίσης απ'ο την εκφώνηση ισχύει η σχέση 1:1 μεταξύ σκηνών και παραστάσεων)
- Για το Q12 έγινε η παραδοχή πως κάθε event (στη σκηνή που πραγματοποιείται) χρειάζεται 5 άτομα τεχνικού προσωπικού
- Υπάρχει ο βασικός πίνακας Artists ο οποίος αφορά καλλιτέχνες. Στον πίνακα των εμφανίσεων εισάγεται το performer_id και επίσης υπάρχει μία μεταβλητή boolean is_solo. Αν το is_solo έχει τιμή ίση με μηδέν, τότε αυτό υποδεικνύει πως ο καλλιτέχνης εμφανίζεται μόνος του και θα καταχωρηθεί το id του, `artist_id`. Αν is_solo έχει τιμή ίση με ένα, τότε αυτό σημαίνει πως στην εμφάνιση συμμετέχει συγκρότημα, επομένως θα γίνει εισαγωγή του αντίστοιχου id του, `band_id`. Η ανάκτηση των συμμετοχόντων καλλιτεχνών γίνεται με βάση του πίνακα `artists_x_bands`
- Θεωρούμε πως τα είδη σε κάθε κατηγορία (για παράδειγμα `payment_methods`, `music_types`, `staff_categories` κλπ) είναι αυτά που γίνονται στην αρχική εισαγωγή και δεν πρόκειται να διαγραφούν. Γι' αυτό το λόγο δεν τίθεται το ζήτημα υλοποίησης triggers για να διαγράφονται τα ξένα κλειδιά στις σχέσεις, καθώς θα παραμένουν αμετάβλητα.

Παράρτημα Β΄

DDL Script

Το DDL Script που κατασκευάζει τους πίνακες της βάσης δεδομένων, τα αντίστοιχα ευρετήρια (indexes) και τους πυροδότες (triggers) βρίσκεται στο GitHub repository, στο παρακάτω μονοπάτι:

`sql/scripts/01_install.sql`

Επιπλέον, το hyperlink προς το DDL Script στο αποθετήριο είναι:

https://github.com/gballos/pulse-university-rdbms/blob/main/sql/scripts/01_install.sql

Ακολουθεί το DDL Script:

Listing Β΄.1: DDL Script

```
-- DATABASE CREATION

DROP DATABASE IF EXISTS pulse_uni_db;
CREATE DATABASE pulse_uni_db;
USE pulse_uni_db;

-- TABLES

DROP TABLE IF EXISTS LOCATIONS;
CREATE TABLE LOCATIONS (
        location_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    address VARCHAR(100),
    city VARCHAR(50),
    country VARCHAR(50),
    continent VARCHAR(50),
    longtitude NUMERIC(5, 2),
    latitude NUMERIC(5, 2),
    image VARCHAR(100), CHECK(image like 'https://%'),
    PRIMARY KEY(location_id)
);

DROP TABLE IF EXISTS FESTIVALS;
CREATE TABLE FESTIVALS (
```

```
    festival_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    date_starting DATE,
    date_ending DATE,
    duration INT,
    location_id INT UNSIGNED,
    image VARCHAR(100), CHECK(image like 'https://%'),
    festival_year INT GENERATED ALWAYS AS (YEAR(date_starting)) STORED,
    PRIMARY KEY(festival_id),
    FOREIGN KEY(location_id) REFERENCES LOCATIONS(location_id),
    UNIQUE (festival_year),
    UNIQUE (location_id),
    CHECK (date_ending > date_starting)
);

DROP TABLE IF EXISTS STAGES;
CREATE TABLE STAGES (
        stage_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    stage_name VARCHAR(50),
    stage_description VARCHAR(150),
    max_capacity INT,
    image VARCHAR(100), CHECK(image like 'https://%'),
    PRIMARY KEY(stage_id)
);

DROP TABLE IF EXISTS TECHNICAL_SUPPLY;
CREATE TABLE TECHNICAL_SUPPLY(
        technical_supply_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        technical_supply_description VARCHAR(150),
    image VARCHAR(100), CHECK(image like 'https://%'),
        PRIMARY KEY(technical_supply_id)
);

DROP TABLE IF EXISTS STAGES_X_TECHNICAL_SUPPLY;
CREATE TABLE STAGES_X_TECHNICAL_SUPPLY(
        amount_of_supply INT UNSIGNED,
        stage_id INT UNSIGNED,
        technical_supply_id INT UNSIGNED,
        PRIMARY KEY(stage_id, technical_supply_id),
        FOREIGN KEY(stage_id) REFERENCES STAGES(stage_id),
        FOREIGN KEY(technical_supply_id) REFERENCES TECHNICAL_SUPPLY(
            technical_supply_id)
);

DROP TABLE IF EXISTS FESTIVAL_EVENTS;
CREATE TABLE FESTIVAL_EVENTS (
        event_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        festival_id INT UNSIGNED NOT NULL,
    stage_id INT UNSIGNED NOT NULL,
    event_date DATE,
    duration INT,
```

```
    image VARCHAR(100), CHECK(image like 'https://%'),
    PRIMARY KEY(event_id),
    FOREIGN KEY(festival_id) REFERENCES FESTIVALS(festival_id),
    FOREIGN KEY(stage_id) REFERENCES STAGES(stage_id)
);

DROP TABLE IF EXISTS MUSIC_TYPES;
CREATE TABLE MUSIC_TYPES(
        music_type_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        music_type VARCHAR(20),
        PRIMARY KEY(music_type_id)
);

DROP TABLE IF EXISTS MUSIC_SUBTYPES;
CREATE TABLE MUSIC_SUBTYPES(
        music_subtype_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        music_subtype VARCHAR(20),
        PRIMARY KEY(music_subtype_id)
);

DROP TABLE IF EXISTS ARTISTS;
CREATE TABLE ARTISTS(
        artist_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        first_name VARCHAR(25),
        last_name VARCHAR(25),
        nickname VARCHAR(25),
        birthday DATE,
        website VARCHAR(100) CHECK(website LIKE 'https://%' OR website LIKE '
            http://%'),
        instagram VARCHAR(50),
    image VARCHAR(100), CHECK(image like 'https://%'),
        PRIMARY KEY(artist_id)
);

DROP TABLE IF EXISTS BANDS;
CREATE TABLE BANDS(
        band_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        name VARCHAR(25),
        date_of_creation DATE,
        website VARCHAR(100) CHECK(website LIKE 'https://%' OR website LIKE '
            http://%'),
        instagram VARCHAR(50),
    image VARCHAR(100), CHECK(website LIKE 'https://%' OR website LIKE 'http
        ://%'),
        PRIMARY KEY(band_id)
);

DROP TABLE IF EXISTS ARTISTS_X_BANDS;
CREATE TABLE ARTISTS_X_BANDS(
        artist_id INT UNSIGNED,
```

```
        band_id INT UNSIGNED,
        PRIMARY KEY(artist_id, band_id),
        FOREIGN KEY(artist_id) REFERENCES ARTISTS(artist_id),
        FOREIGN KEY(band_id) REFERENCES BANDS(band_id)
);

DROP TABLE IF EXISTS ARTISTS_X_MUSIC;
CREATE TABLE ARTISTS_X_MUSIC (
    artist_id INT UNSIGNED NOT NULL,
    music_type_id INT UNSIGNED,
    music_subtype_id INT UNSIGNED,
    PRIMARY KEY(artist_id, music_type_id, music_subtype_id),
    FOREIGN KEY (artist_id) REFERENCES ARTISTS(artist_id),
    FOREIGN KEY (music_type_id) REFERENCES MUSIC_TYPES(music_type_id),
    FOREIGN KEY (music_subtype_id) REFERENCES MUSIC_SUBTYPES(music_subtype_id)
);

DROP TABLE IF EXISTS BANDS_X_MUSIC;
CREATE TABLE BANDS_X_MUSIC (
    band_id INT UNSIGNED NOT NULL,
    music_type_id INT UNSIGNED,
    music_subtype_id INT UNSIGNED,
    PRIMARY KEY(band_id, music_type_id, music_subtype_id),
    FOREIGN KEY (band_id) REFERENCES BANDS(band_id),
    FOREIGN KEY (music_type_id) REFERENCES MUSIC_TYPES(music_type_id),
    FOREIGN KEY (music_subtype_id) REFERENCES MUSIC_SUBTYPES(music_subtype_id)
);

DROP TABLE IF EXISTS PERFORMANCE_TYPES;
CREATE TABLE PERFORMANCE_TYPES (
        performance_type_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    performance_type VARCHAR(50),
    PRIMARY KEY(performance_type_id)
);

DROP TABLE IF EXISTS PERFORMANCES;
CREATE TABLE PERFORMANCES (
        performance_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    performance_type_id INT UNSIGNED NOT NULL,
    event_id INT UNSIGNED NOT NULL,
    performance_time TIME,
    duration INT CHECK(duration <= 180),                    -- duration
        in minutes
    order_in_show INT,
    is_solo BOOLEAN,
    performer_id INT UNSIGNED,
    image VARCHAR(100), CHECK(image like 'https://%'),
    PRIMARY KEY(performance_id),
    FOREIGN KEY(performance_type_id)  REFERENCES PERFORMANCE_TYPES(
        performance_type_id),
```

```
    FOREIGN KEY(event_id) REFERENCES FESTIVAL_EVENTS(event_id)
);

DROP TABLE IF EXISTS TECHNICAL_ROLES;
CREATE TABLE TECHNICAL_ROLES(
        technical_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        technical_description VARCHAR(40),
        PRIMARY KEY(technical_id)
);

DROP TABLE IF EXISTS STAFF_CATEGORIES;
CREATE TABLE STAFF_CATEGORIES(
        staff_category_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        staff_category_desc VARCHAR(20),
        technical_id INT UNSIGNED DEFAULT NULL,                 -- NOT NULL for
            technical, NULL for security/assistance
        PRIMARY KEY(staff_category_id),
        FOREIGN KEY(technical_id) REFERENCES TECHNICAL_ROLES(technical_id)
);

DROP TABLE IF EXISTS LEVELS_OF_EXPERTISE;
CREATE TABLE LEVELS_OF_EXPERTISE(
        level_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        level_description VARCHAR(20),
        PRIMARY KEY(level_id)
);

DROP TABLE IF EXISTS STAFF;
CREATE TABLE STAFF (
        staff_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    category_id INT UNSIGNED,
    level_id INT UNSIGNED,
    event_id INT UNSIGNED NOT NULL,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    age INT,
    image VARCHAR(100), CHECK(image like 'https://%'),
    PRIMARY KEY(staff_id),
    FOREIGN KEY(event_id) REFERENCES FESTIVAL_EVENTS(event_id),
        FOREIGN KEY(category_id) REFERENCES STAFF_CATEGORIES(staff_category_id)
            ,
        FOREIGN KEY(level_id) REFERENCES LEVELS_OF_EXPERTISE(level_id)
);

DROP TABLE IF EXISTS VISITORS;
CREATE TABLE VISITORS(
        visitor_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        first_name VARCHAR(20),
        last_name VARCHAR(20),
        phone_number VARCHAR(40),
```

```
        email VARCHAR(20),
        age INT,
        PRIMARY KEY(visitor_id)
);

DROP TABLE IF EXISTS TICKET_TYPES;
CREATE TABLE TICKET_TYPES(
        ticket_type_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        ticket_type VARCHAR(25),
        PRIMARY KEY(ticket_type_id)
);

DROP TABLE IF EXISTS PAYMENT_METHODS;
CREATE TABLE PAYMENT_METHODS(
        payment_method_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        payment_method VARCHAR(50),
        PRIMARY KEY(payment_method_id)
);

DROP TABLE IF EXISTS TICKETS;
CREATE TABLE TICKETS(
        ticket_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        event_id INT UNSIGNED,
        visitor_id INT UNSIGNED,
        ticket_type_id INT UNSIGNED,
        payment_method_id INT UNSIGNED,
        ean_code CHAR(13),
        is_scanned BOOLEAN,
        date_bought DATE,
        cost INT,
        PRIMARY KEY(ticket_id),
    FOREIGN KEY(event_id) REFERENCES FESTIVAL_EVENTS(event_id),
        FOREIGN KEY(ticket_type_id) REFERENCES TICKET_TYPES(ticket_type_id),
        FOREIGN KEY(visitor_id) REFERENCES VISITORS(visitor_id),
        FOREIGN KEY(payment_method_id) REFERENCES PAYMENT_METHODS(
            payment_method_id),
    UNIQUE (event_id, visitor_id)
);

DROP TABLE IF EXISTS LIKERT_RATINGS;
CREATE TABLE LIKERT_RATINGS(
        rating_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        rating_number INT UNSIGNED CHECK (rating_number BETWEEN 1 AND 5),
        rating_description VARCHAR(25),
        PRIMARY KEY(rating_id)
);

DROP TABLE IF EXISTS REVIEWS;
CREATE TABLE REVIEWS(
        review_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
        visitor_id INT UNSIGNED NOT NULL,
        performance_id INT UNSIGNED NOT NULL,
        interpretation_rating INT UNSIGNED,
        sound_lighting_rating INT UNSIGNED,
        stage_presence_rating INT UNSIGNED,
        organization_rating INT UNSIGNED,
        overall_impression_rating INT UNSIGNED,
        PRIMARY KEY(review_id),
        FOREIGN KEY(visitor_id) REFERENCES VISITORS(visitor_id),
        FOREIGN KEY(performance_id) REFERENCES PERFORMANCES(performance_id),
        FOREIGN KEY(interpretation_rating) REFERENCES LIKERT_RATINGS(rating_id)
            ,
        FOREIGN KEY(sound_lighting_rating) REFERENCES LIKERT_RATINGS(rating_id)
            ,
        FOREIGN KEY(stage_presence_rating) REFERENCES LIKERT_RATINGS(rating_id)
            ,
        FOREIGN KEY(organization_rating) REFERENCES LIKERT_RATINGS(rating_id),

        FOREIGN KEY(overall_impression_rating) REFERENCES LIKERT_RATINGS(
            rating_id)
);

DROP TABLE IF EXISTS BUYERS;
CREATE TABLE BUYERS(
        buyer_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        event_id INT UNSIGNED,
        ticket_type_id INT UNSIGNED,
        ticket_id INT UNSIGNED,
    requested_at DATETIME,
        PRIMARY KEY(buyer_id),
        FOREIGN KEY(event_id) REFERENCES FESTIVAL_EVENTS(event_id),
        FOREIGN KEY(ticket_id) REFERENCES TICKETS(ticket_id),
        FOREIGN KEY(ticket_type_id) REFERENCES TICKET_TYPES(ticket_type_id)
);

DROP TABLE IF EXISTS TICKETS_FOR_RESALE;
CREATE TABLE TICKETS_FOR_RESALE(
        ticket_for_resale_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        ticket_id INT UNSIGNED,
        event_id INT UNSIGNED,
        ticket_type_id INT UNSIGNED,
    listed_at DATETIME,
        PRIMARY KEY(ticket_for_resale_id),
        FOREIGN KEY(ticket_id) REFERENCES TICKETS(ticket_id),
        FOREIGN KEY(ticket_type_id) REFERENCES TICKET_TYPES(ticket_type_id),
        FOREIGN KEY(event_id) REFERENCES FESTIVAL_EVENTS(event_id)
);

-- TRIGGERS

```

```sql
-- Delete trigger for performances
DROP TRIGGER IF EXISTS delete_performance_after_artist;
DELIMITER //
CREATE TRIGGER delete_performance_after_artist
AFTER DELETE ON ARTISTS
FOR EACH ROW
BEGIN
    DELETE FROM PERFORMANCES
    WHERE performer_id = OLD.artist_id AND is_solo = 1;
END;

//
DROP TRIGGER IF EXISTS delete_performance_after_band;
CREATE TRIGGER delete_performance_after_band
AFTER DELETE ON BANDS
FOR EACH ROW
BEGIN
    DELETE FROM PERFORMANCES
    WHERE performer_id = OLD.band_id AND is_solo = 0;
END;
//

-- Check review eligibility
DROP TRIGGER IF EXISTS check_review_ticket_scanned;

CREATE TRIGGER check_review_ticket_scanned
BEFORE INSERT ON REVIEWS
FOR EACH ROW
BEGIN
    DECLARE event_of_perf INT;
    DECLARE ticket_scanned BOOL;

    -- Get the event for the performance being reviewed
    SELECT event_id INTO event_of_perf
    FROM PERFORMANCES
    WHERE performance_id = NEW.performance_id;

    -- Check if visitor has a scanned ticket for that event
    SELECT COUNT(*) > 0 INTO ticket_scanned
    FROM TICKETS
    WHERE visitor_id = NEW.visitor_id
      AND event_id = event_of_perf
      AND is_scanned = TRUE;

    -- If not, block the review
    IF NOT ticket_scanned THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You can only review performances you attended (
            ticket must be scanned).';
    END IF;
```

```
END

//

-- Performer ID Trigger
DROP TRIGGER IF EXISTS check_performer;
CREATE TRIGGER check_performer
BEFORE INSERT ON PERFORMANCES
FOR EACH ROW
BEGIN
   DECLARE artist_count INT;
   DECLARE band_count INT;

   IF NEW.is_solo = TRUE THEN
       SELECT COUNT(*) INTO artist_count FROM ARTISTS WHERE artist_id = NEW.
           performer_id;
       IF artist_count = 0 THEN
           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No matching artist for
               solo performance.';
       END IF;
   ELSE
       SELECT COUNT(*) INTO band_count FROM BANDS WHERE band_id = NEW.
           performer_id;
       IF band_count = 0 THEN
           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No matching band for
               ensemble performance.';
       END IF;
   END IF;
END
//

-- Check 3 consecutive years
DROP TRIGGER IF EXISTS check_4th_year;
CREATE TRIGGER check_4th_year
BEFORE INSERT ON PERFORMANCES
FOR EACH ROW
BEGIN
    DECLARE performer_year INT;
    DECLARE prev_years INT;

    -- Get the year of the current performance
    SELECT YEAR(event_date) INTO performer_year
    FROM FESTIVAL_EVENTS
    WHERE event_id = NEW.event_id;

    -- Count how many times this performer performed in the 3 years before this
        one
    IF NEW.is_solo = TRUE THEN
        SELECT COUNT(DISTINCT YEAR(fe.event_date)) INTO prev_years
        FROM PERFORMANCES p
```

```
        JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
        WHERE p.performer_id = NEW.performer_id
          AND p.is_solo = 1
          AND YEAR(fe.event_date) BETWEEN performer_year - 3 AND performer_year
              - 1;
    ELSE
        SELECT COUNT(DISTINCT YEAR(fe.event_date)) INTO prev_years
        FROM PERFORMANCES p
        JOIN FESTIVAL_EVENTS fe ON p.event_id = fe.event_id
        WHERE p.performer_id = NEW.performer_id
          AND p.is_solo = 0
          AND YEAR(fe.event_date) BETWEEN performer_year - 3 AND performer_year
              - 1;
    END IF;

    IF prev_years = 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Performer cannot take part in the festival for more
             than 3 consecutive years.';
    END IF;
END

//

-- Check stage capacity
DROP TRIGGER IF EXISTS check_stage_capacity;
CREATE TRIGGER check_stage_capacity
BEFORE INSERT ON TICKETS
FOR EACH ROW
BEGIN
    DECLARE cap INT;
    DECLARE ticket_count INT;
    DECLARE vip_count INT;
    DECLARE ticket_type_name VARCHAR(20);

    SELECT ticket_type INTO ticket_type_name
    FROM TICKET_TYPES
    WHERE ticket_type_id = NEW.ticket_type_id;

    SELECT s.max_capacity INTO cap
    FROM FESTIVAL_EVENTS fe
    JOIN STAGES s ON s.stage_id = fe.event_id
    WHERE NEW.event_id = fe.event_id;

    SELECT COUNT(*) INTO ticket_count
    FROM TICKETS
    WHERE event_id = NEW.event_id;

    SELECT COUNT(*) INTO vip_count
    FROM TICKETS
```

```
    WHERE event_id = NEW.event_id AND ticket_type_name = 'VIP';

    IF ticket_count >= cap THEN
        INSERT INTO BUYERS (event_id, ticket_type_id, ticket_id, requested_at)
        VALUES (NEW.event_id, NEW.ticket_type_id, NULL, NOW());
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Exceeded stage capacity. No more tickets available.
             Added buyer to queue';
    END IF;

    IF vip_count >= 0.1*cap THEN
        INSERT INTO BUYERS (event_id, ticket_type_id, ticket_id, requested_at)
        VALUES (NEW.event_id, NEW.ticket_type_id, NULL, NOW());
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No more VIP tickets available. Added buyer to queue
            ';
    END IF;
END

-- Resale Queue | Matching
//
DROP PROCEDURE IF EXISTS match_resale_queue;
CREATE PROCEDURE match_resale_queue()
BEGIN
    DECLARE b_id INT;
    DECLARE b_event INT;
    DECLARE b_ticket INT;
    DECLARE b_ticket_type INT;
    DECLARE r_id INT;
    DECLARE r_ticket INT;

    match_loop: LOOP
        SELECT buyer_id, event_id, ticket_type_id, ticket_id
        INTO b_id, b_event, b_ticket_type, b_ticket
        FROM BUYERS
        ORDER BY requested_at
        LIMIT 1;

        -- No buyers left
        IF b_id IS NULL THEN
            LEAVE match_loop;
        END IF;

        -- Check if requested by id:
        IF b_ticket IS NOT NULL THEN
            SELECT ticket_for_resale_id
            INTO r_id
            FROM TICKETS_FOR_RESALE
            WHERE ticket_id = b_ticket
            ORDER BY listed_at
```

```
            LIMIT 1;

            IF r_id IS NOT NULL THEN
                DELETE FROM TICKETS_FOR_RESALE WHERE ticket_for_resale_id =
                    r_id;
                DELETE FROM BUYERS WHERE buyer_id = b_id;
                ITERATE match_loop;
            END IF;
        END IF;

        -- Otherwise match by event + type
        SELECT ticket_for_resale_id, ticket_id
        INTO r_id, r_ticket
        FROM TICKETS_FOR_RESALE
        WHERE event_id = b_event AND ticket_type_id = b_ticket_type
        ORDER BY listed_at
        LIMIT 1;

        IF r_id IS NOT NULL THEN
          DELETE FROM TICKETS_FOR_RESALE WHERE ticket_for_resale_id = r_id;
          DELETE FROM BUYERS WHERE buyer_id = b_id;
          ITERATE match_loop;
        END IF;

        -- If here no match found
        LEAVE match_loop;
    END LOOP match_loop;
END //

-- Run match for new entries
//
DROP TRIGGER IF EXISTS after_ticket_resale_insert;
CREATE TRIGGER after_ticket_resale_insert
AFTER INSERT ON TICKETS_FOR_RESALE
FOR EACH ROW
BEGIN
    DECLARE not_valid BOOL;

    SELECT t.is_scanned INTO not_valid
    FROM TICKETS t
    WHERE (NEW.ticket_id = t.ticket_id);

    IF not_valid = 0 THEN
        CALL match_resale_queue();
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You cannot list scanned ticket.';
    END IF;
END;
//
```

```
DROP TRIGGER IF EXISTS after_buyer_insert;
CREATE TRIGGER after_buyer_insert
AFTER INSERT ON BUYERS
FOR EACH ROW
BEGIN
    CALL match_resale_queue();
END;
//

-- trigger to check if an event is within the date range of the festival
DROP TRIGGER IF EXISTS event_dates_in_festival;
CREATE TRIGGER event_dates_in_festival
BEFORE INSERT ON FESTIVAL_EVENTS
FOR EACH ROW
BEGIN
    DECLARE fest_start_date DATE;
    DECLARE fest_end_date DATE;

    SELECT date_starting, date_ending
        INTO fest_start_date, fest_end_date
        FROM FESTIVALS
    WHERE festival_id = NEW.festival_id;

    IF NEW.event_date < fest_start_date OR NEW.event_date > fest_end_date THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Event date must be between festival start and end
            dates';
    END IF;
END
//

-- create trigger to check that break is within range of 5-30 mins
DROP TRIGGER IF EXISTS check_break_range;
//
CREATE TRIGGER check_break_range
BEFORE INSERT ON PERFORMANCES
FOR EACH ROW
BEGIN
    DECLARE prev_end_seconds INT;
    DECLARE current_start_seconds INT;

    -- Convert new performance time to seconds since midnight
    SET current_start_seconds = TIME_TO_SEC(NEW.performance_time);

    -- Get the most recent performance's end time (not just previous
        order_in_show)
    SELECT TIME_TO_SEC(ADDTIME(performance_time, SEC_TO_TIME(duration * 60)))
    INTO prev_end_seconds
    FROM PERFORMANCES
    WHERE event_id = NEW.event_id
```

```
    ORDER BY performance_time DESC
    LIMIT 1;

    -- Only check if there was a previous performance
    IF prev_end_seconds IS NOT NULL THEN
        SET @time_diff_minutes = (current_start_seconds - prev_end_seconds) /
            60;

        IF @time_diff_minutes < 5 OR @time_diff_minutes > 30 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Break must be 5-30 minutes.';
        END IF;
    END IF;
END
//
DROP TRIGGER IF EXISTS block_fest_delete;
CREATE TRIGGER block_fest_delete BEFORE DELETE ON FESTIVALS
FOR EACH ROW
BEGIN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FESTIVAL delete cancelled';
END
//
DROP TRIGGER IF EXISTS block_event_delete;
CREATE TRIGGER block_event_delete BEFORE DELETE ON FESTIVAL_EVENTS
FOR EACH ROW
BEGIN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FESTIVAL_EVENTS delete cancelled'
      ;
END
//
DROP VIEW IF EXISTS staff_coverage_view;
CREATE VIEW staff_coverage_view AS

SELECT
  fe.event_id,
  fe.stage_id,
  s.max_capacity,

  SUM(IF(sc.technical_id IS NOT NULL, 1, 0)) AS technical_assigned,
  SUM(IF(sc.staff_category_desc = 'Security', 1, 0))  AS security_assigned,
  SUM(IF(sc.staff_category_desc = 'Assistant', 1, 0)) AS general_assigned,


  CEIL(s.max_capacity * 0.05)                         AS security_required,
  CEIL(s.max_capacity * 0.02)                         AS general_required

FROM FESTIVAL_EVENTS fe
JOIN STAGES s ON fe.stage_id = s.stage_id
LEFT JOIN STAFF st ON st.event_id = fe.event_id       -- Left join because we
    need to see the event even if it has no staff
```

```
643 LEFT JOIN STAFF_CATEGORIES sc ON st.category_id = sc.staff_category_id
644
645 GROUP BY fe.event_id, fe.stage_id, s.max_capacity;
646
647 //
648
649 DELIMITER ;
650
651 -- indexes for the most common joins / foreign keys
652
653 -- festival_events
654 CREATE INDEX idx_events_festivals ON FESTIVAL_EVENTS(festival_id);
655 CREATE INDEX idx_events_performances_events ON PERFORMANCES(event_id);
656 CREATE INDEX idx_events_tickets ON TICKETS(event_id);
657 CREATE INDEX idx_events_staff ON STAFF(event_id);
658
659 -- artist / band relationships
660 CREATE INDEX idx_ab_artist ON ARTISTS_X_BANDS(artist_id);
661 CREATE INDEX idx_ab_band ON ARTISTS_X_BANDS(band_id);
662
663 -- composite indexes
664
665 -- for queries 2, 3, 4, 10, 15  /  require the connection both events and
        performers
666 CREATE INDEX idx_events_performers ON PERFORMANCES(event_id, performer_id);
667
668 -- for queries 6, 9, 15  /  require both visitors and events
669 -- useful for the big number of tickets
670 CREATE INDEX idx_tickets_visitors_events ON TICKETS(visitor_id, event_id);
671
672 -- performance-review related queries 4, 15
673 CREATE INDEX idx_performances_performers ON PERFORMANCES(performance_id,
        performer_id);
674
675 -- review related queries 4, 6, 9, 15
676 CREATE INDEX idx_reviews_performance ON REVIEWS(performance_id);
677 CREATE INDEX idx_reviews_visitors_performances ON REVIEWS(visitor_id,
        performance_id);
678
679 -- staff queries 7, 12   /   filter event & categories
680 CREATE INDEX idx_staff_event_level ON STAFF(event_id, category_id);
```

Παράρτημα Γ΄

DML Script

Το DML Script που εισάγει τα δεδομένα ελέγχου στη βάση δεδομένων — τα οποία δημιουργήθηκαν σύμφωνα με τη διαδικασία που περιγράφεται στην ενότητα Δεδομένα Ελέγχου — βρίσκεται στο GitHub repository, στο παρακάτω μονοπάτι:

`sql/scripts/02_load.sql`

Επιπλέον, το hyperlink προς το DML Script στο αποθετήριο είναι:

https://github.com/gballos/pulse-university-rdbms/blob/main/sql/scripts/02_load.sql

Επισημαίνεται ότι η γεννήτρια τυχαίων αριθμών έχει ρυθμιστεί με σταθερή τιμή `seed`*, ώστε τα αποτελέσματα να είναι ντετερμινιστικά και επαναλήψιμα.*