



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ - ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ

Ιωάννου Βασίλης, AM: 03121663

Τσάκαλος Θεόδωρος, AM: 03121668

Μαρκίδης Δημήτρης, AM: 03121670

Περιεχόμενα:

1. Βάση Δεδομένων

1.1 Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship Diagram)

1.2 Σχεσιακό σχήμα (Relational Schema)

1.3 Indexes

1.4 Triggers και Procedures

1.5 Εισαγωγή δεδομένων

2. Queries

Σύνδεσμος GitHub repository: <https://github.com/VasilisIoannou/DatabaseNTUA>

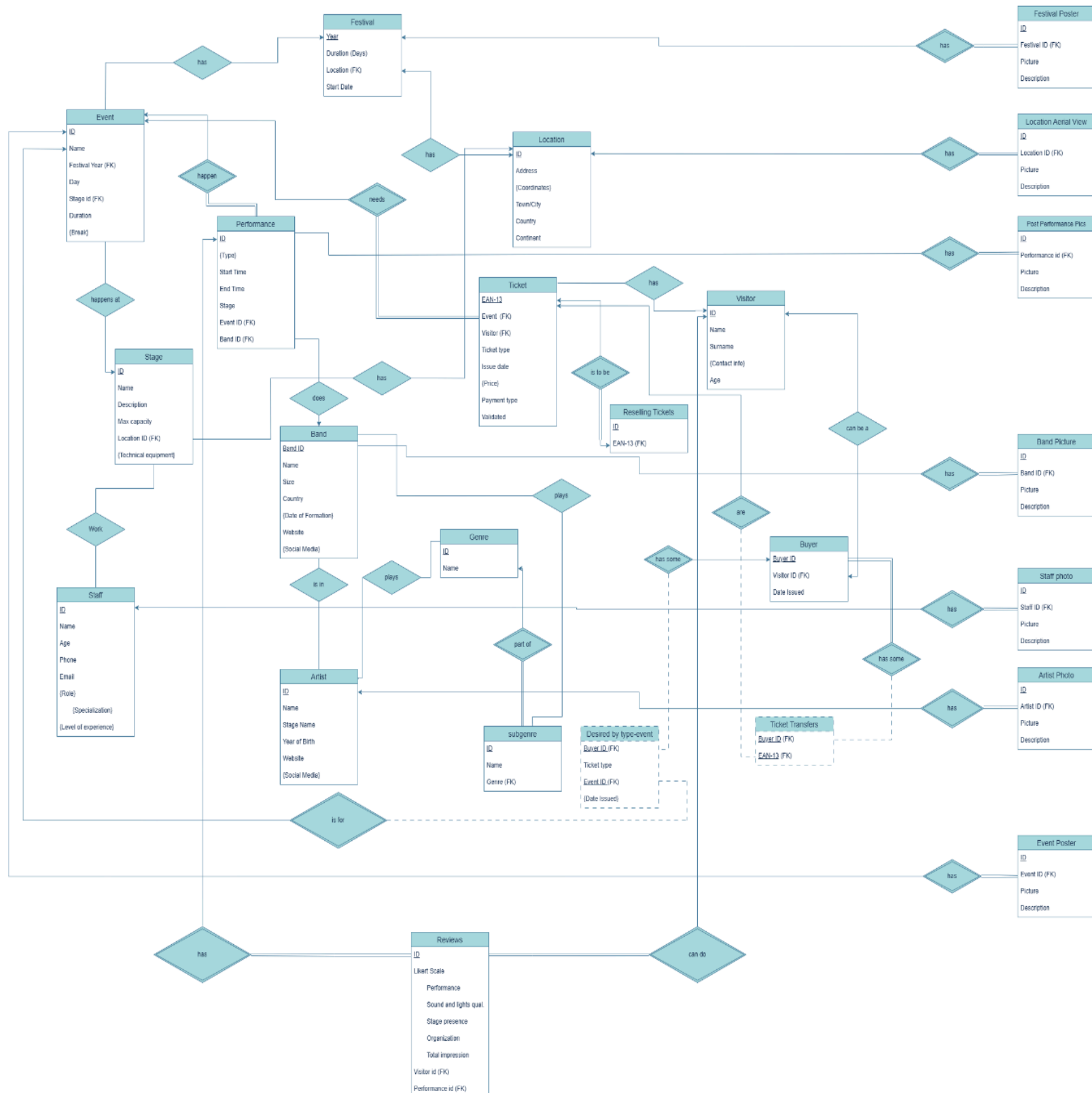
Πριν εξηγήσουμε τη βάση δεδομένων που υλοποιήσαμε, παραθέτουμε μερικές υποθέσεις / παραδοχές που κάναμε σχετικά με τη λειτουργία του φεστιβάλ.

- Θεωρούμε ότι το φεστιβάλ του 2025 δεν έχει πραγματοποιηθεί ακόμα.
- Θεωρούμε ότι όταν ένας επισκέπτης αγοράζει εισιτήριο για ημέρα/ες, αγοράζει ένα εισιτήριο για κάθε event της/των συγκεκριμένης/ων ημέρας/ων.
- Θεωρούμε ότι το κοινό για κάθε event θα είναι ίσο με τον αριθμό εισιτηρίων που πουλήθηκαν για το συγκεκριμένο event. Επομένως, αν για παράδειγμα δύο events διεξάγονται ταυτόχρονα σε διαφορετικές σκηνές, αναγκαζόμαστε να θεωρήσουμε ότι ότι κάποιος που έχει ημερήσιο εισιτήριο θα πάει και στα δύο events. Προφανώς αυτό δεν είναι δυνατόν, όμως επειδή δεν μπορούμε να ξέρουμε αν θα πάει ή όχι, πρέπει να υπολογίσουμε τα μέτρα ασφαλείας στην περίπτωση που θα πάει σε οποιοδήποτε από τα δύο.
- Θεωρούμε ότι η διάρκεια ενός event είναι καθορισμένη πριν οριστούν οι εμφανίσεις και τα διαλείμματα του συγκεκριμένου event.
- Θεωρούμε ότι κάθε σκηνή χρειάζεται 2 τεχνικούς.
- Θεωρούμε ότι οι καλλιτέχνες ανήκουν όλοι σε τουλάχιστον μία μπάντα, επομένως και οι σόλο καλλιτέχνες ανήκουν σε μπάντα με 1 άτομο.
- Θεωρούμε ότι το συνολικό κοινό σε μια παράσταση είναι ίσο με το άθροισμα των θεατών, του βοηθητικού προσωπικού και του προσωπικού ασφαλείας.
- Θεωρούμε ότι υπάρχουν οι κατηγορίες εισιτηρίων “φοιτητικό” και “μειωμένο εισιτήριο ηλικιωμένων”.
- Θεωρούμε ότι κάθε οντότητα μπορεί να έχει πολλές φωτογραφίες, για αυτό δημιουργούμε νέους πίνακες για κάθε μια από αυτές.
- Γενικά, σε όλα τα queries που είναι εξεφρασμένα σε παρελθοντικό χρόνο, θεωρούμε ότι αφορούν μόνο φεστιβάλ που έχουν γίνει, άρα σε φεστιβάλ πριν το 2025.

1. ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

1.1 Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity - Relationship Diagram)

Αρχικά σχεδιάσαμε το διάγραμμα οντοτήτων-συσχετίσεων που φαίνεται παρακάτω:



Στο **ER** μας χρησιμοποιήσαμε τις εξής σημασιολογικές συμβάσεις:

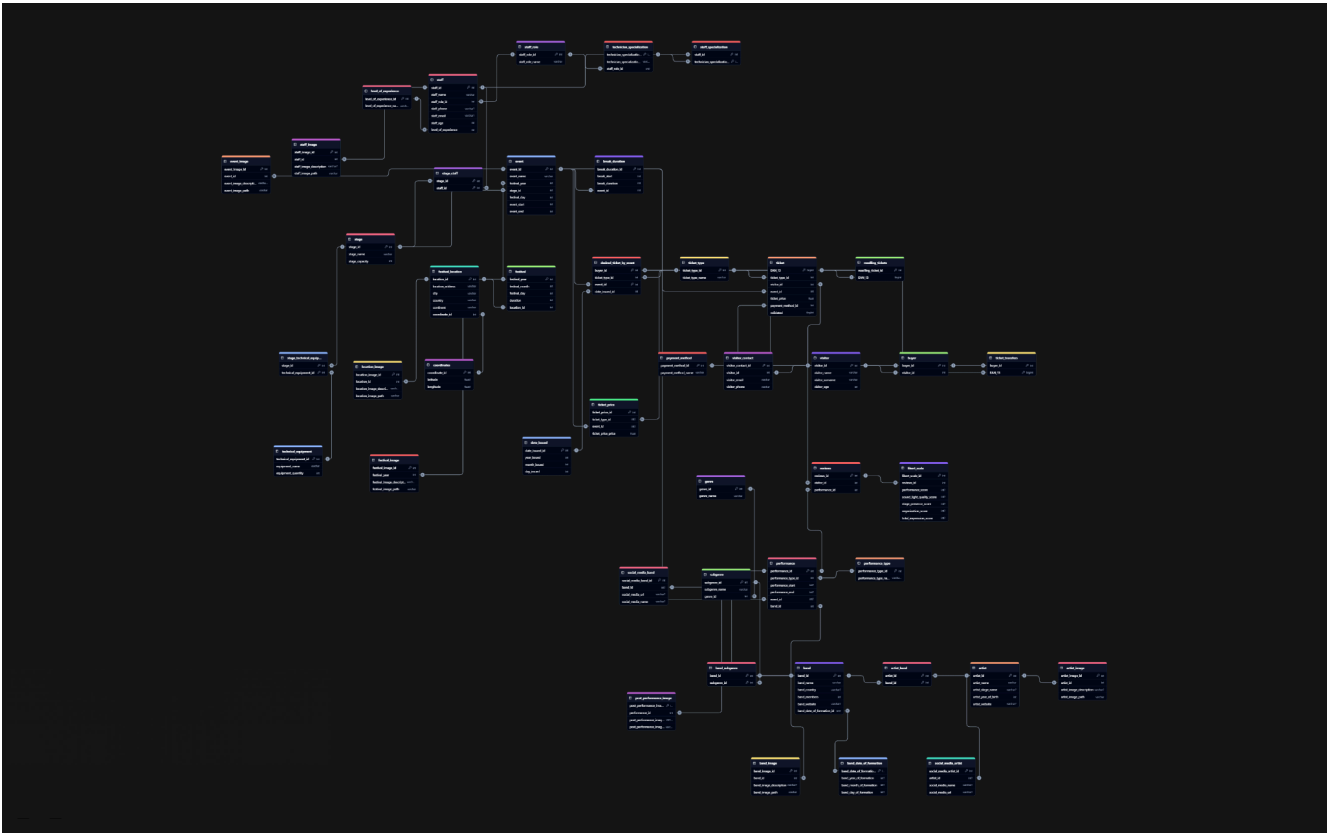
- Όλες οι οντότητες συνδέονται με άλλες οντότητες μέσω ρόμβων που αναγράφουν μονολεκτικά την εννοιολογική τους σύνδεση
- Οι σχέσεις (*one to one*, *one to many*, etc.) συμβολίζονται μέσω των γραμμών που συνδέουν την οντότητα με τον ρόμβο:

1. Γραμμή χωρίς βέλος: Σχέση *many*
2. Γραμμή με βέλος: Σχέση *one*

Άρα για παράδειγμα μια σχέση *one to many* συμβολίζεται με βέλος να “βλέπει” την οντότητα που είναι “το *one*” και γραμμή χωρίς βέλος να “βλέπει” την οντότητα που είναι “το *many*”

- Σε κάθε οντότητα το υπογραμμισμένο χαρακτηριστικό είναι το **Primary Key** της
- Τα χαρακτηριστικά μίας οντότητας που είναι foreign key για κάποια άλλη οντότητα το αναγράφουν σε παρένθεση “**(FK)**”
- Στις οντότητες που είναι το “*many*” της σχέσης η αντίστοιχη γραμμή “φεύγει” από τον τίτλο της οντότητας, ενώ σε οντότητες που είναι το “*one*” της σχέσης η γραμμή φεύγει από το **Foreign Key** τους
- Οντότητες που αντιστοιχούν στην κατηγορία *weak entities* (δεν μπορούν να υπάρχουν χωρίς την οντότητα στην οποία υπάγονται (*identifying entity*)), έχουν δύο χαρακτηριστικά μαζί σαν **PK** και έχουν διακεκομμένο περίγραμμα. Ενώνονται με τις συνδεόμενες οντότητες με διπλό ρόμβο.
- Οντότητες που έχουν σχέση *total participation* με άλλες οντότητες (δηλαδή που κάθε εγγραφή της οντότητας πρέπει να σχετίζεται με μία εγγραφή της οντότητας στην οποία υπάγεται - πχ. κάθε εμφάνιση πρέπει να σχετίζεται με μία παράσταση) ενώνονται με τις συνδεόμενες οντότητες με διπλές γραμμές και διπλούς ρόμβους
- Χαρακτηριστικά που στη βάση δεδομένων υλοποιήσαμε ως ξεχωριστούς βοηθητικούς πίνακες, αναγράφονται στην οντότητα με αγκύλες (πχ. {*Performance Type*}).
- Βοηθητικοί πίνακες που δείχνουν σε άλλους βοηθητικούς πίνακες σημειώνονται με indented χαρακτηριστικά σε αγκύλες (πχ. *Staff Role* -> *Staff Role Specialization*)
- Βοηθητικοί πίνακες που όμως τα χαρακτηριστικά τους είναι αρκετά σημαντικά για να αναγραφούν στο ER αναγράφονται με indentation (πχ. *Likert Scale*)

1.2 Σχεσιακό σχήμα (Relational Schema)



Από το σχεσιακό διάγραμμα μπορούμε να δούμε ακριβώς τους πίνακες και τις σχέσεις μεταξύ τους όπως υλοποιήθηκαν σε SQL. Ο κώδικας φαίνεται στο αρχείο *install.sql*.

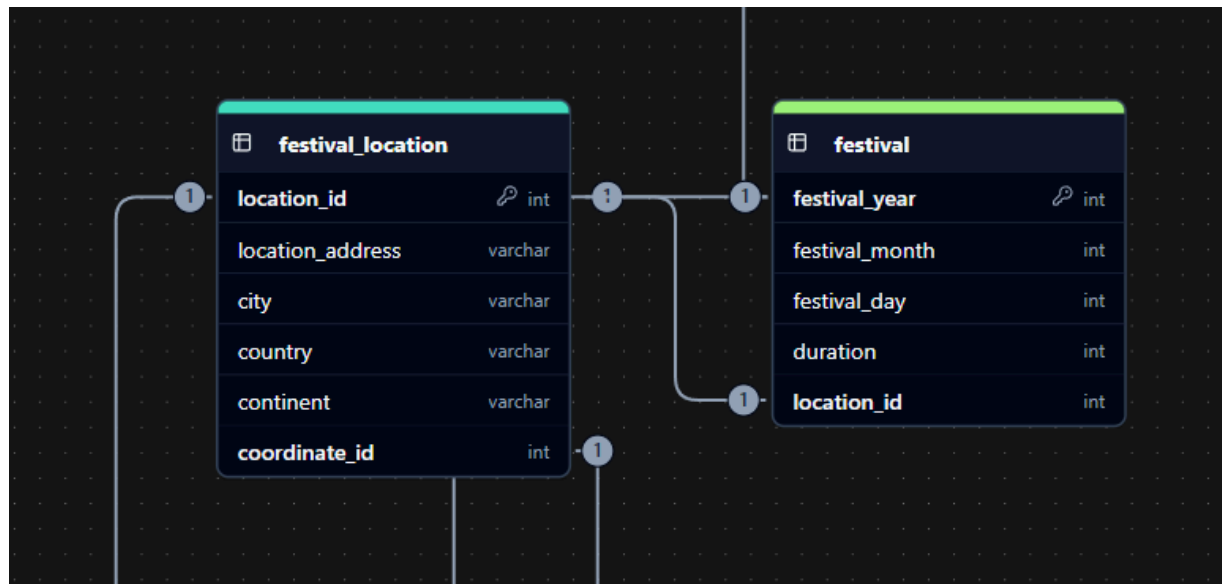
Για παράδειγμα, πιο κάτω βλέπουμε τη δημιουργία του πίνακα **festival**:

```
38 CREATE TABLE festival(  
39     festival_year int NOT NULL CHECK (festival_year > 0),  
40     festival_month int NOT NULL CHECK (festival_month > 0 AND festival_month < 13),  
41     festival_day int NOT NULL CHECK (festival_day > 0), -- A Trigger will check the festival_day < N  
42     duration int NOT NULL CHECK (duration > 0), -- Duration of the festival in days  
43     location_id int NOT NULL UNIQUE,  
44     PRIMARY KEY(festival_year),  
45     FOREIGN KEY(location_id) REFERENCES festival_location(location_id)  
46 );
```

Όπως βλέπουμε δημιουργούμε τα attributes *festival_year*, *festival_month*, *festival_day*, τα οποία ορίζουν την ημερομηνία του festival. Ο λόγος που αποφύγαμε τη χρήση του datatype DATE είναι το γεγονός ότι η χρήση integers είναι συνεπής σε όλα τα συστήματα βάσεων, αλλά και για λόγους απλότητας και ταχύτητας υπολογισμών. Για τις ίδιες μεταβλητές χρησιμοποιούμε τους περιορισμούς NOT NULL, που σημαίνει πως δεν πρέπει να μείνουν κενές, αλλά και άλλους περιορισμούς ακεραιότητας χρησιμοποιώντας την εντολή CHECK. Η εντολή αυτή κάνει τον έλεγχο που την ακολουθεί όταν εισαχθούν δεδομένα στον πίνακα. Ωστόσο, για να διασφαλίσουμε ότι π.χ δεν μπορούμε να εισάγουμε 31 μέρες σε μήνα που δεν έχει, χρησιμοποιούμε *Triggers*, στα οποία θα αναφερθούμε παρακάτω.

Ορίζουμε ακόμη τα attributes *duration* και *location_id* τα οποία αντιστοιχούν στη διάρκεια σε μέρες και την τοποθεσία του festival. Η τοποθεσία πρέπει να είναι ξεχωριστή για κάθε festival και για αυτό χρησιμοποιούμε την εντολή UNIQUE η οποία δείχνει στη βάση ότι δεν μπορούν να υπάρχουν δύο festival entities με ίδιο *location_id*. Τέλος, ορίζουμε ως PRIMARY KEY το έτος του φεστιβάλ, δηλαδή το *festival_year*, εφόσον υπάρχει μόνο ένα φεστιβάλ ανά έτος, και ως FOREIGN KEY το *location_id* το οποίο συνδέει τον πίνακα **festival** με τον πίνακα **festival_location** μέσω του attribute *location_id*.

Η λειτουργία και η σχέση μεταξύ των δύο αυτών πινάκων φαίνεται και στο διάγραμμα:



Τα PRIMARY KEYS φαίνονται με bold γράμματα και με ένα κλειδί δίπλα τους, ενώ τα FOREIGN KEYS συνδέονται με τον πίνακα που αναφέρουν μέσω γραμμών. Η σχέση μεταξύ των πινάκων φαίνεται με μικρούς κύκλους και περιέχει τον αριθμό 1 για 'one' και N για 'many'. Όπως βλέπουμε, σε αυτή την περίπτωση, η σχέση είναι 'one-to-one'.

Για την υλοποίηση των σχέσεων 'many-to-many', χρησιμοποιήσαμε ενδιάμεσους πίνακες όπως στην περίπτωση του πίνακα **stage_staff**, ο οποίος συνδέει τις σκηνές με το προσωπικό. Ο πίνακας αυτός δείχνει σε ποια σκηνή έχει ανατεθεί ο κάθε υπάλληλος.

```

Run | Select | Ask AI
CREATE TABLE stage_staff(
  stage_id int,
  staff_id int,
  PRIMARY KEY(stage_id, staff_id),
  FOREIGN KEY(stage_id) REFERENCES stage(stage_id) ON DELETE CASCADE,
  FOREIGN KEY(staff_id) REFERENCES staff(staff_id) ON DELETE CASCADE
);
  
```

Παρατηρούμε ότι ο πίνακας έχει COMPOSITE PRIMARY KEY, το συνδυασμό *stage_id* και *staff_id* τα οποία αναφέρουν το αντίστοιχο attribute στον αντίστοιχο πίνακα.

Στο διάγραμμα επαληθεύουμε ότι οι δύο πίνακες έχουν σχέση 'many-to-many' αφού πρώτα περάσουν από τον ενδιάμεσο **stage_staff**.

Οι υπόλοιποι πίνακες υλοποιούνται με παρόμοιο τρόπο. Γενικά προσπαθήσαμε να ορίσουμε τους πίνακες λαμβάνοντας υπόψη τις 5 μορφές κανονικοποίησης, αν και υπήρξαν περιπτώσεις που τις παραβιάσαμε.

Στην παρακάτω περίπτωση, για παράδειγμα, στους πίνακες **ticket** και **ticket_price** αποθηκεύουμε δύο φορές την τιμή εισιτηρίου, στα attributes *ticket_price* και *ticket_price_price* αντίστοιχα. Το γεγονός ότι το *ticket_price* εξαρτάται από δύο άλλα attributes (*ticket_type_id*, *event_id*) τα οποία δεν αποτελούν κλειδί, παραβιάζει την 3η κανονική μορφή (και το BCNF).

```
296 CREATE TABLE ticket(  
297     EAN_13 bigint NOT NULL CHECK (EAN_13 > 0),  
298     ticket_type_id int NOT NULL,  
299     visitor_id int NOT NULL,  
300     event_id int NOT NULL,  
301     ticket_price float NOT NULL CHECK(ticket_price >= 0),  
302     payment_method_id int NOT NULL,  
303     validated boolean NOT NULL,  
304     PRIMARY KEY(EAN_13),  
305     FOREIGN KEY(ticket_type_id) REFERENCES ticket_type(ticket_type_id),  
306     FOREIGN KEY(event_id) REFERENCES event(event_id) ON DELETE CASCADE,  
307     FOREIGN KEY(visitor_id) REFERENCES visitor(visitor_id) ON DELETE CASCADE,  
308     FOREIGN KEY(payment_method_id) REFERENCES payment_method(payment_method_id),  
309     UNIQUE KEY(visitor_id, event_id)  
310 );  
311  
312 > Run | Select | Ask AI  
312 CREATE TABLE ticket_price(  
313     ticket_price_id int AUTO_INCREMENT,  
314     ticket_type_id int,  
315     event_id int,  
316     ticket_price_price float NOT NULL CHECK(ticket_price_price >= 0),  
317     PRIMARY KEY(ticket_price_id),  
318     FOREIGN KEY(event_id) REFERENCES event(event_id) ON DELETE CASCADE,  
319     FOREIGN KEY(ticket_type_id) REFERENCES ticket_type(ticket_type_id) ON DELETE CASCADE  
320 );  
321
```

1.3 Indexes

Δημιουργήσαμε τα indexes στα attributes *event_id*, *EAN_13*, *visitor_id*, καθώς παρατηρήσαμε ότι οι αντίστοιχοι πίνακες και πιο συγκεκριμένα αυτά τα attributes προσπελαύνονται συχνά. Ακόμη δημιουργήσαμε index στο *artist_year_of_birth* επειδή στο query 5 ζητείται να φιλτράρουμε τους καλλιτέχνες κάτω των 30 ετών και στο *ticket_price* επειδή στο query 1 καλούμαστε να αθροίσουμε πολλές τιμές εισιτηρίων. Ακόμη δημιουργούμε άλλα 2 indexes, τα οποία θα αναλύσουμε αργότερα, για την υλοποίηση των ερωτημάτων 4 και 6.

1.4 Triggers και Procedures

Όπως αναφέραμε και προηγουμένως, για να διασφαλίσουμε τη σωστή εισαγωγή και αποθήκευση των δεδομένων, χρειάστηκε σε αρκετά σημεία να πραγματοποιήσουμε δικούς μας ελέγχους. Για να το επιτύχουμε αυτό, δημιουργήσαμε Triggers και Procedures.

Triggers:

Τα triggers είναι ένα κομμάτι κώδικα που εκτελείται αυτόματα όταν εισάγουμε (INSERT), τροποποιούμε (UPDATE) ή διαγράφουμε (DELETE), σε/από έναν πίνακα.

Για να κατανοήσουμε καλύτερα την έννοια των triggers και το λόγο που τα χρησιμοποιήσαμε, θα αναλύσουμε εκτενώς 2 από αυτά.

1. prevent_festival_deletion

```

6 CREATE TRIGGER prevent_festival_deletion
7 BEFORE DELETE ON festival
8 FOR EACH ROW
9 BEGIN
10     SIGNAL SQLSTATE '45000'
11     SET MESSAGE_TEXT = 'Festivals cannot be deleted from the system';
12 END//
13
```

Το trigger που φαίνεται πιο πάνω εμποδίζει τον χρήστη από το να διαγράψει κάποιο φεστιβάλ από τη βάση δεδομένων. Οι εντολή BEFORE DELETE ON **festival** ορίζει ότι το *trigger* θα τρέξει όταν πάμε να διαγράψουμε κάτι από τον πίνακα **festival** (αλλά πριν το κάνουμε). Η εντολή FOR EACH ROW, ορίζει ότι τα παραπάνω ισχύουν για οποιαδήποτε γραμμή επιχειρήσουμε να διαγράψουμε.

Ο κώδικας που θα εκτελεστεί τοποθετείται ανάμεσα στα BEGIN και END. Στην περίπτωση αυτή, η βάση πετάει error, με το μήνυμα που βλέπουμε στην εικόνα, ενημερώνοντας έτσι τον χρήστη ότι δεν μπορεί να διαγράψει φεστιβάλ.

2. assign_security_if_needed

```
272  /* Trigger to assign security staff to stages based on ticket sales. */
273  > Run | Select | Ask AI
274  CREATE TRIGGER assign_security_if_needed
275  BEFORE INSERT ON ticket
276  FOR EACH ROW
277  BEGIN
278      DECLARE ticket_count INT;
279      DECLARE security_count INT;
280      DECLARE stage_id_val INT;
281      DECLARE staff_to_assign INT;
282
283      SELECT e.stage_id INTO stage_id_val
284      FROM event e WHERE e.event_id = NEW.event_id;
285
286      SELECT COUNT(*) INTO ticket_count
287      FROM ticket t
288      JOIN event e ON e.event_id = t.event_id
289      WHERE e.stage_id = stage_id_val;
290
291      SELECT COUNT(*) INTO security_count
292      FROM stage_staff ss
293      JOIN staff s ON s.staff_id = ss.staff_id
294      WHERE ss.stage_id = stage_id_val AND s.staff_role_id = 2;
295
296      IF ticket_count + 1 > security_count * 20 THEN
297          SELECT s.staff_id INTO staff_to_assign
298          FROM staff s
299          WHERE s.staff_role_id = 2
300          AND s.staff_id NOT IN (
301              SELECT ss.staff_id
302              FROM stage_staff ss
303              JOIN event e1 ON ss.stage_id = e1.stage_id
304              JOIN event e2 ON e2.event_id = NEW.event_id
305              WHERE e1.festival_year = e2.festival_year
306              AND e1.festival_day = e2.festival_day
307              AND (
308                  (e2.event_start BETWEEN e1.event_start AND e1.event_end) OR
309                  (e2.event_end BETWEEN e1.event_start AND e1.event_end) OR
310                  (e2.event_start <= e1.event_start AND e2.event_end >= e1.event_end)
311              )
312          )
313          AND s.staff_id NOT IN (
314              SELECT staff_id FROM stage_staff WHERE stage_id = stage_id_val
315          )
316          LIMIT 1;
317
318          IF staff_to_assign IS NOT NULL THEN
319              INSERT INTO stage_staff(stage_id, staff_id)
320              VALUES (stage_id_val, staff_to_assign);
321          ELSE
322              SIGNAL SQLSTATE '45000'
323              SET MESSAGE_TEXT = 'Cannot add ticket: No available security staff without schedule conflict.';
324          END IF;
325      END IF;
326  END;
327  //
```

Το παραπάνω trigger χειρίζεται τον περιορισμό “Το προσωπικό ασφαλείας πρέπει να καλύπτει τουλάχιστον το 5% του συνολικού αριθμού θεατών σε κάθε σκηνή”. Όποτε πουληθεί ένα νέο εισιτήριο για κάποια παράσταση, η βάση αυτόματα ελέγχει εάν πρέπει να αναθέσει κάποιο προσωπικό ασφαλείας στη σκηνή όπου θα λάβει μέρος η εν λόγω παράσταση (event).

Αρχικά δηλώνουμε μερικές μεταβλητές που θα μας βοηθήσουν στη συνέχεια. Με τις παρακάτω εντολές βρίσκουμε τη σκηνή στην οποία θα λάβει μέρος το event, από τον πίνακα **event**, και την αποθηκεύουμε στη μεταβλητή *stage_id_val*.

```
SELECT e.stage_id INTO stage_id_val
FROM event e WHERE e.event_id = NEW.event_id;
```

Ακολουθώντας, μετράμε με την COUNT(*), τον αριθμό των εισιτηρίων που έχουν πουληθεί για το συγκεκριμένο event. Για να φτάσουμε από τον πίνακα **ticket** στον πίνακα **event**, κάνουμε “JOIN” με τη συνθήκη ότι ταιριάζει το *event_id* στον πίνακα **ticket** με το *event_id* στον πίνακα **event**. Κρατάμε τα events που έχουν ως σκηνή αυτή που βρήκαμε πριν.

```
SELECT COUNT(*) INTO ticket_count
FROM ticket t
JOIN event e ON e.event_id = t.event_id
WHERE e.stage_id = stage_id_val;
```

Στη συνέχεια, μετράμε τον αριθμό προσωπικού που έχει ανατεθεί στη συγκεκριμένη σκηνή και έχει ρόλο με id = 2, δηλαδή που είναι προσωπικό ασφαλείας. Κάνουμε JOIN όπως και πιο πάνω.

```
SELECT COUNT(*) INTO security_count
FROM stage_staff ss
JOIN staff s ON s.staff_id = ss.staff_id
WHERE ss.stage_id = stage_id_val AND s.staff_role_id = 2;
```

Έπειτα, ελέγχουμε εάν μια σκηνή θα χρειαστεί επιπλέον προσωπικό ασφαλείας με τη συνθήκη στο IF. Ψάχνουμε τον πίνακα **staff** και κρατάμε το προσωπικό ασφαλείας το οποίο δεν έχει άλλη δουλειά τη συγκεκριμένη μέρα, ώρα και χρόνο και δεν είναι ήδη στη σκηνή αυτή. Για να κάνουμε αυτούς τους ελέγχους βλέπουμε στον ενδιαμέσο πίνακα **stage_staff**:

```

IF ticket_count + 1 > security_count * 20 THEN

    SELECT s.staff_id INTO staff_to_assign
    FROM staff s
    WHERE s.staff_role_id = 2
        AND s.staff_id NOT IN (
            SELECT ss.staff_id
            FROM stage_staff ss
            JOIN event e1 ON ss.stage_id = e1.stage_id
            JOIN event e2 ON e2.event_id = NEW.event_id
            WHERE e1.festival_year = e2.festival_year
                AND e1.festival_day = e2.festival_day
                AND (
                    (e2.event_start BETWEEN e1.event_start AND e1.event_end) OR
                    (e2.event_end BETWEEN e1.event_start AND e1.event_end) OR
                    (e2.event_start <= e1.event_start AND e2.event_end >= e1.event_end)
                )
        )
        AND s.staff_id NOT IN (
            SELECT staff_id FROM stage_staff WHERE stage_id = stage_id_val
        )
    LIMIT 1;

```

Με το LIMIT 1, κρατάμε μόνο 1 staff_id.

Τέλος, ελέγχουμε αν βρήκαμε κάποιον. Εάν ναι, τον αναθέτουμε στη σκηνή μέσω ενός INSERT στον **stage_staff**. Αλλιώς, η βάση πετάει error και μας ενημερώνει ότι δεν μπορούμε να πουλήσουμε περισσότερα εισιτήρια καθώς δεν υπάρχει αρκετό προσωπικό ώστε να πληρούνται οι προϋποθέσεις.

```

IF staff_to_assign IS NOT NULL THEN
    INSERT INTO stage_staff(stage_id, staff_id)
    VALUES (stage_id_val, staff_to_assign);
ELSE
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot add ticket: No available security staff without schedule conflict.';
END IF;
END IF;
END;
//

```

Τα υπόλοιπα triggers λειτουργούν παρόμοια και είναι στην επόμενη σελίδα:

Triggers

- **prevent_event_deletion:** Δεν επιτρέπει τη διαγραφή των events
- **prevent_invalid_review:** Ελέγχει ότι ένας επισκέπτης έχει χρησιμοποιημένο εισιτήριο για ένα event, προτού τον αφήσει να αξιολογήσει τις εμφανίσεις του.
- **prevent_artist_stage_conflict:** Ελέγχει εάν ένας καλλιτέχνης έχει ανατεθεί σε πολλές σκηνές την ίδια ώρα
- **prevent_event_overlapping:** Ελέγχει εάν μια παράσταση συμπίπτει με μια άλλη
- **check_event_start_end:** Ελέγχει ότι η ώρα έναρξης είναι μικρότερη από την ώρα λήξης
- **check_event_day:** Ελέγχει ότι η μέρα μιας παράστασης δεν υπερβαίνει τη διάρκεια του φεστιβάλ
- **set_band_members_to_zero:** Θέτει αυτόματα το attribute *band_members* σε 0, όταν εισάγουμε μια μπάντα.
- **increment_band_members / decrement_band_members:** Αυξάνει/Μειώνει αυτόματα κατά ένα το attribute *band_members*, όταν εισαχθεί/διαγραφεί ένας καλλιτέχνης από μια μπάντα.
- **delete_break_after_performance:** Διαγράφει το διάλειμμα μετά από μια εμφάνιση, όταν η εμφάνιση διαγραφεί
- **check_band_formation_before_performance:** Ελέγχει ότι μια μπάντα έχει ιδρυθεί πριν την εμφάνισή της.
- **check_band_members_before_performance:** Ελέγχει ότι μια μπάντα έχει μέλη πριν την εμφάνισή της.
- **check_technician_specialization:** Διασφαλίζει ότι μόνο οι τεχνικοί έχουν ειδίκευση
- **auto_set_role_id:** Θέτει αυτόματα την τιμή του attribute *staff_role* σε 1, όταν εισάγουμε ειδίκευση
- **assign_secondary_if_needed:** Αναθέτει αυτόματα βοηθητικό προσωπικό στις σκηνές
- **assign_technicians_on_stage_insert:** Αναθέτει αυτόματα τεχνικούς στη σκηνή όταν εισάγουμε σκηνή
- **assign_technicians_on_staff_insert:** Αναθέτει αυτόματα τεχνικούς στη σκηνή όταν εισάγουμε τεχνικούς
- **check_festival_day / check_date_issued / check_band_date_of_formation:** Ελέγχει την ακεραιότητα των ημερών ανά μήνα
- **remove_validated_tickets:** Διαγράφει εισιτήρια από την ουρά επαναπώλησης όταν αυτά ενεργοποιηθούν.
- **date_issued_check_for_desired_ticket:** Ελέγχει ότι το η μέρα εκδήλωσης ενδιαφέροντος για αγορά εισιτηρίου είναι πριν τη μέρα της παράστασης.
- **check_matches_reselling_tickets / check_matches_desired_ticket_by_event:** Ελέγχουν αν ενημερώθηκε κάποιος πίνακας που αφορά την ουρά μεταπώλησης, και καλούν το αντίστοιχο procedure που χειρίζεται την συναλλαγή.

Procedures:

Τα stored procedures είναι προ-μεταγλωττισμένα κομμάτια κώδικα τα οποία εκτελούνται στην κλήση του ονόματός τους και εκτελούν συγκεκριμένες λειτουργίες.

Δημιουργήσαμε αρκετά procedures, κυρίως για την εισαγωγή δεδομένων σε πολλούς πίνακες ταυτόχρονα, χωρίς την χρήση insert statements στον καθένα ξεχωριστά. Μαζί με την εισαγωγή δεδομένων, τα procedures χρησιμεύουν και στον έλεγχο των δεδομένων που είναι προς εισαγωγή, και την παρεμπόδιση της εισαγωγής τους αν είναι λανθασμένα, μέσω errors. Τα procedures που δημιουργήσαμε, καθώς και η αντίστοιχη λειτουργία τους, είναι τα εξής:

- **insert_performance_break:** Χρησιμοποιείται για την εισαγωγή δεδομένων στους πίνακες **performance** και **break duration**, χωρίς να χρειαστεί να δώσουμε εντολές και για τους δύο αυτούς πίνακες. Ταυτόχρονα, για να διασφαλίσει την ορθότητα των δεδομένων, κάνει τους εξής ελέγχους:
 1. Ότι το νέο performance συμπίπτει με τις ώρες που διεξάγεται το event στο οποίο λαμβάνει χώρα και ότι, μαζί με το διάλειμμα μετά την τελευταία εμφάνιση στο event, το performance δεν ξεπερνά το συνολικό duration του event.
 2. Ότι στο συγκεκριμένο event εκείνη την ώρα δεν έχει άλλο performance.
 3. Ότι οι εμφανίσεις τύπου “warmup” είναι πρώτες στο event και οι εμφανίσεις τύπου “closing” είναι τελευταίες.
 4. Ότι το διάλειμμα είναι διάρκειας 5 με 30 λεπτών.
 5. Ότι ο καλλιτέχνης του νέου performance δεν συμμετάσχει για τρίτη συνεχόμενη χρονιά σε φεστιβάλ.
 6. Ότι το performance δεν ξεπερνάει τις 3 ώρες
- **insert_visitor_with_ticket:** Χρησιμοποιείται για την εισαγωγή στους πίνακες **ticket**, **visitor** και **visitor_contact**. Καλώντας το procedure αυτό, δίνοντας το contact information του επισκέπτη, το event για το οποίο αγοράζει εισιτήριο, τον τύπο του εισιτηρίου και τον κωδικό EAN-13 του εισιτηρίου, γεμίζονται και οι τρεις προεναφερθέντες πίνακες με τα σωστά δεδομένα. Ταυτόχρονα, για να διασφαλίσει την ορθότητα των δεδομένων, κάνει τους εξής ελέγχους:
 1. Ότι ο κωδικός EAN-13 του εισιτηρίου είναι όντως έγκυρος κωδικός (13 ψηφία και σωστό check digit).
 2. Ότι έχει γραφτεί σωστά στην αίτηση ο τύπος του εισιτηρίου
 3. Ότι έχουν δοθεί εισιτήρια με αυτό τον τύπο για το συγκεκριμένο event (ότι υπάρχει δηλαδή entry για το συγκεκριμένο event και εισιτήριο στον πίνακα **ticket_price**)
 4. Ότι το event δεν είναι ήδη sold out.
 5. Αν το εισιτήριο είναι VIP, ότι δεν ξεπερνάει τον περιορισμό για το 10% επί των συνολικών εισιτηρίων.
 6. Αν το εισιτήριο είναι Senior Citizen, ότι ο επισκέπτης που το αιτείται είναι όντως άνω των 65 χρονών. Αντίστοιχο έλεγχο **δεν** έχουμε για τα φοιτητικά, γιατί δεν αντιστοιχούν μόνο σε νεαρή ηλικία.

- **insert_ticket_with_existing_visitor:** Χρησιμοποιεί στην αγορά δεύτερου εισιτηρίου για κάποιον επισκέπτη που έχει ήδη ένα εισιτήριο και επομένως υπάρχει ήδη καταχωρημένος στους τρεις πίνακες και πρέπει απλά να προστεθεί ακόμα μία εγγραφή του στον πίνακα **ticket**. Όπως είπαμε, θεωρούμε ότι ένας επισκέπτης μπορεί να έχει στο όνομά του μόνο ένα εισιτήριο ανά event. Επομένως, το procedure αυτό μπορεί να χρησιμοποιηθεί μόνο για αγορά εισιτηρίου για καινούργιο event. Εκτελεί τους ίδιους ελέγχους με το procedure **insert_visitor_with_ticket**.
- **insert_buyer_visitor:** Εισάγει αγοραστή επισκέπτη, δηλαδή βάζοντας τα στοιχεία επικοινωνίας του, τον εισάγουμε στην λίστα υποψήφιων αγοραστών από την ουρά μεταπώλησης. Το procedure επηρεάζει μόνο τον πίνακα **visitor** και τον πίνακα **buyer**, μέχρι να τοποθετηθεί προς πώληση το εισιτήριο που επιθυμεί και να τοποθετηθεί η συναλλαγή αυτόματα στον πίνακα **ticket_transfers** (μέσω του procedure **process_ticket_matches**)
- **insert_reselling_ticket:** Εισάγει ένα εισιτήριο στην ουρά μεταπώλησης (πίνακας **reselling_tickets**). Προτού το εισάγει, κάνει τους εξής ελέγχους:
 1. Ότι όντως τα εισιτήρια για το συγκεκριμένο event έχουν πουληθεί όλα.
 2. Ότι το εισιτήριο δεν έχει χρησιμοποιηθεί ήδη το εισιτήριο (ότι δεν είναι επικυρωμένο)
- **reselling_desired_by_id:** Καλώντας το procedure με παράμετρο συγκεκριμένο εισιτήριο (με το **reselling_ticket_id** που συνδέεται με το EAN-13 του) που πωλείται (δηλαδή είναι στον πίνακα **reselling_tickets**), το procedure πραγματοποιεί αυτόματα την συναλλαγή, δηλαδή, αφαιρεί το συγκεκριμένο εισιτήριο από τον πίνακα **reselling_tickets**, αλλάζει τα στοιχεία επικοινωνίας του κατόχου του εισιτηρίου σε αυτά του νέου κατόχου (στον πίνακα **ticket**), και εγγράφει την συναλλαγή στον πίνακα **ticket_transfers**.
- **process_ticket_matches:** Το procedure αυτό καλείται μέσω του trigger **check_matches_reselling_tickets** στην εγγραφή νέου εισιτηρίου προς πώληση (στον πίνακα **reselling_tickets**) ή μέσω του trigger **check_matches_desired_ticket_by_event** στην εισαγωγή νέου επιθυμητού εισιτηρίου για συγκεκριμένο event και τύπο προς αγορά (πίνακας **desired_ticket_by_event**).

Αν ένας από τους δύο πίνακες ενημερωθεί, δηλαδή είτε προστέθηκε νέο εισιτήριο προς πώληση ή προστέθηκε καινούργια αίτηση για εισιτήριο που πληροί τις προδιαγραφές, το procedure αυτό συγκρίνει τα εισιτήρια και αν ταιριάζουν (δηλαδή αν υπάρχει στην λίστα με τα εισιτήρια που πωλούνται αυτό που ζητείται), χειρίζεται την συναλλαγή, εκτελώντας τις ίδιες λειτουργίες με το προηγούμενο. Λειτουργεί σαν λίστα *FIFO*, δηλαδή *First In First Out*, ήτοι το εισιτήριο θα το πάρει ο πρώτος που εκδήλωσε ενδιαφέρον.

1.4 Εισαγωγή Δεδομένων

Για την εισαγωγή δεδομένων στην βάση μας, χρησιμοποιήσαμε κυρίως έτοιμες λίστες από το διαδίκτυο (πχ. Top Μουσικοί 2025, Top Τενίστες 2025, Top έλληνες πολιτικοί κτλπ) και απαντήσεις από LLMs και μετά χρησιμοποιήσαμε κυρίως scripts στην python που να δημιουργούν τα *insert statements* ή τα *procedure calls* που χρειάζονται για την εισαγωγή τους. Ένα παράδειγμα python script που χρησιμοποιήσαμε είναι το κάτωθι, για την δημιουργία 1340 αντικειμένων τεχνικού εξοπλισμού, δουλειά που χειρωνακτικά θα μας έπαιρνε ώρες:

```
# Equipment ID ranges
speakers = list(range(1, 301))           # 300 total
lights = list(range(301, 901))           # 600 total
microphones = list(range(901, 1111))     # 210 total
consoles = list(range(1111, 1211))       # 100 total
special_effects = list(range(1211, 1341)) # 130 total

stage_ids = list(range(1, 39))           # Stage IDs from 1 to 38

# Requirements per stage
req_per_stage = {
    "speakers": 7,
    "lights": 15,
    "microphones": 5,
    "consoles": 2,
    "special_effects": 3
}

# Map to equipment lists
equipment_lists = {
    "speakers": speakers,
    "lights": lights,
    "microphones": microphones,
    "consoles": consoles,
    "special_effects": special_effects
}

# Assignments list
assignments = []

# Equipment pointers
eq_index = {
    "speakers": 0,
    "lights": 0,
    "microphones": 0,
    "consoles": 0,
    "special_effects": 0
}

# Generate insert pairs
for stage_id in stage_ids:
    for eq_type in req_per_stage:
        needed = req_per_stage[eq_type]
        eq_list = equipment_lists[eq_type]
        idx = eq_index[eq_type]

        # Take 'needed' items
        for _ in range(needed):
            if idx >= len(eq_list):
                raise ValueError(f"Not enough {eq_type} for all stages.")
            assignments.append(f"({stage_id}, {eq_list[idx]})")
            idx += 1

        # Update pointer
        eq_index[eq_type] = idx

# Output
print("INSERT INTO stage_technical_equipment (stage_id, technical_equipment_id) VALUES")
print(",\n".join(assignments) + ";")
```

Άλλο παράδειγμα χρήσης της python για την διευκόλυνση της εισαγωγής δεδομένων είναι το κάτωθι πρόγραμμα, το οποίο παίρνει ως είσοδο έναν δωδεκαψήφιο αριθμό και παράγει το 13ο ψηφίο (*check digit*) με τρόπο που η έξοδος είναι έγκυρος κωδικός EAN-13. Το χρησιμοποιήσαμε, με βάση το 100000000000, για την παραγωγή όλων των EAN-13 κωδικών που χρησιμοποιήσαμε στην βάση δεδομένων μας.

```
def calculate_ean13_check_digit(number):  
    sum_odd = sum(int(number[i]) for i in range(0, 12, 2))  
    sum_even = sum(int(number[i]) for i in range(1, 12, 2))  
    total = sum_odd + sum_even * 3  
    check_digit = (10 - (total % 10)) % 10  
    return str(check_digit)  
  
def generate_ean13_codes(start, count):  
    ean13_codes = []  
    for i in range(count):  
        base_number = str(start + i).zfill(12)  
        check_digit = calculate_ean13_check_digit(base_number)  
        ean13_code = base_number + check_digit  
        ean13_codes.append(ean13_code)  
    return ean13_codes  
  
if __name__ == "__main__":  
    start_number = 100000000100 # Starting 12-digit number  
    total_codes = 400 # Number of EAN-13 codes to generate  
    codes = generate_ean13_codes(start_number, total_codes)  
    for code in codes:  
        print(code)
```


Οδηγίες για τη σωστή Εισαγωγή Δεδομένων

Για την εισαγωγή δεδομένων μέσω του DML file (*sql/load.sql*), χρησιμοποιούμε εκτός από insert statements, έτοιμα stored procedures και triggers, τα μεν για να γεμίσουμε πολλούς πίνακες ταυτόχρονα και με ελεγμένα δεδομένα και τα δε για την αυτόματη ενημέρωση πινάκων που επηρεάζονται από άλλα δεδομένα που θα δώσουμε.

Συγκεκριμένα, με procedures γεμίζουν οι πίνακες:

1. **performance** , **break_duration** (proc. **insert_performance_break**)
2. **ticket** , **visitor** , **visitor_contact** (proc. **insert_visitor_with_ticket**)
3. **buyer** (proc. **insert_buyer_visitor**)
4. **ticket_transfers** (procs. **reselling_desired_by_id** , **process_ticket_matches**)

Επίσης, με triggers ενημερώνονται αυτόματα τα κάτωθι δεδομένα:

1. Στην προσθήκη ή διαγραφή σε μπάντα καλλιτέχνη από τον ενδιαμέσο πίνακα **artist_band**, αυξάνεται ή μειώνεται αντίστοιχα αυτόματα ο αριθμός μελών της αντίστοιχης μπάντας στον πίνακα **band** (triggers *decrement_band_members* και *increment_band_members*)
2. Ο πίνακας **stage_staff**, δηλαδή ο ενδιαμέσος πίνακας που αναθέτει το προσωπικό στις σκηνές που χρειάζεται, γεμίζει αυτόματα (triggers **assign_security_if_needed**, **assign_technicians_on_staff_insert**, **assign_technicians_on_stage_insert**, **assign_secondary_if_needed**) για να πληρούνται οι προϋποθέσεις (2% βοηθητικό, 5% ασφαλείας και 2 τεχνικοί ανά σκηνή)

Επίσης, μερικοί βοηθητικοί πίνακες γεμίζουν αυτόματα, με σταθερά δεδομένα, ήδη κατά την δημιουργία των πινάκων:

performance_type: “Warm up”, “Headline”, “Special guest”, “Closing act”

level_of_experience: “Beginner”, “Intermediate”, “Advanced”, “Expert”, “Master”

staff_role: “Technician”, “Security”, “Secondary”

payment_method: “Credit card”, “Debit card”, “Paypal”, “Cash”

ticket_type: “VIP”, “Regular”, “Student”, “Senior Citizen”, “Backstage”

Κατά την εκτέλεση του dml μας, load.sql, τα δεδομένα που δημιουργούνται είναι:

12 χρονιές του φεστιβάλ (κατ’επέκταση και 12 ζευγάρια συντεταγμένων, 12 διευθύνσεις και 12 ημερομηνίες) από το **2016 έως και το 2027** (δύο μελλοντικά + 2025 που δεν έγινε ακόμα)

150 καλλιτέχνες, εκ των οποίων οι **130** ανήκουν σε συγκρότημα και οι **20** είναι ανεξάρτητοι. Με μέσο όρο περίπου 4 μέλη ανά συγκρότημα, ο αριθμός συγκροτημάτων που βάλαμε είναι **30**. Όπως είπαμε, θεωρούμε τον ανεξάρτητο καλλιτέχνη ως μπάντα με 1 άτομο, άρα ο συνολικός αριθμός “συγκροτημάτων” ανέρχεται στα **50 συγκροτήματα**. Ένας καλλιτέχνης μπορεί να ανήκει σε πάνω από ένα συγκροτήματα, για αυτό

180 λογαριασμοί social media, που αντιστοιχούν στους 150 καλλιτέχνες και τις 30 μπάντες. Ο κώδικάς μας επιτρέπει ένας καλλιτέχνης να έχει δύο ή περισσότερους λογαριασμούς στα σόσιαλ παρόλο που στα δεδομένα δεν βάλαμε κάποιον να έχει πάνω από έναν.

10 genres μουσικής, με **37 subgenres** που απορρέουν από αυτά

38 σκηνές, με χωρητικότητες από 15 έως 30 άτομα

50 events, που το καθένα αντιστοιχεί σε μία σκηνή, αλλά μια σκηνή μπορεί να χρησιμοποιηθεί σε πολλά events. Αντιστοιχίσαμε **4 events** το **2019**, **4 events** το **2020**, **3 events** το **2018**, **5 events** το **2019**, **3 events** το **2020**, **5 events** το **2021**, **4 events** το **2022**, **5 events** το **2023**, **5 events** το **2024**, **6 events** το **2025**, **3 events** το **2026** και **3 events** το **2027**.

106 performances που αντιστοιχούν σε events, με κάθε event να έχει μέσο όρο 2 performances.

1340 αντικείμενα τεχνικού εξοπλισμού, δηλαδή **300 ηχεία**, **600 συστήματα φωτισμού**, **210 μικρόφωνα**, **100 κονσόλες** και **130 συστήματα Special Effects**.

Αναθέσαμε σε κάθε σκηνή **7 ηχεία**, **15 συστήματα φωτισμού**, **5 μικρόφωνα**, **2 κονσόλες** και **3 συστήματα Special Effects**. Επομένως, στην “αποθήκη” μένουν **34 ηχεία**, **30 συστήματα φωτισμού**, **20 μικρόφωνα**, **24 κονσόλες** and **16 συστήματα Special Effects**.

50 άτομα προσωπικό: 19 τεχνικοί, 23 για ασφάλεια και 8 βοηθητικοί.

Από τους 19 τεχνικούς, οι **7** είναι **Sound Engineers**, οι **5** είναι **Light Engineers** και από **2** είναι οι **Network Engineers** και οι **Pyro Technicians**.

Οι τιμές των εισιτηρίων διαφέρουν ανά μέρα, event και χρόνο. Κατά κανόνα, αν θεωρήσουμε **x** την τιμή των κανονικών εισιτηρίων, τα εισιτήρια VIP είναι **x + 40**, τα φοιτητικά **x - 10**, τα μειωμένα για ηλικιωμένους **x - 20** και τα backstage **x + 80**.

Βάλαμε συνολικά **254 εισιτήρια**, που αντιστοιχούν σε 20 εισιτήρια ανά έτος, συν 14 επισκέπτες που ξανααγοράζουν εισιτήριο. Σύνολο έχουμε **260 επισκέπτες**, 20 ανά έτος συν 20 που έχουν μπει στην λίστα για την ουρά μεταπώλησης.

Βάλαμε **12 εισιτήρια προς μεταπώληση**, ένα για κάθε έτος, τα οποία τέθηκαν όλα προς πώληση πριν την ημέρα διεξαγωγής του αντίστοιχου φεστιβάλ (έλεγχος που γίνεται μέσω trigger).

Επίσης, βάλαμε άλλα **12 εισιτήρια που ζητώνται**.

Βάλαμε **81 αξιολογήσεις**, δηλαδή περίπου 9 ανά έτος για 9 έτη (αξιολογήσεις έχουν γίνει μόνο για φεστιβάλ που έχουν πραγματοποιηθεί).

Βάλαμε ενδεικτικά **17 φωτογραφίες**, καλλιτεχνών, τοποθεσιών, εμφανίσεων κτλπ.

2. QUERIES

Views:

Για τον καλύτερο έλεγχο των αποτελεσμάτων των queries που ζητούνται, δημιουργήσαμε τα απαραίτητα **Views**. Ένα view είναι ένας εικονικός πίνακας που παρουσιάζει δεδομένα από διάφορους πίνακες με βάση τα αποτελέσματα κάποιων queries.

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

Query (αρχείο Query/Q01.sql):

```
SELECT
    f.festival_year,
    pm.payment_method_name,
    SUM(t.ticket_price) AS total_earnings
FROM
    festival f
JOIN event e ON e.festival_year = f.festival_year
JOIN ticket t ON t.event_id = e.event_id
JOIN payment_method pm ON pm.payment_method_id = t.payment_method_id
GROUP BY
    f.festival_year,
    pm.payment_method_name
ORDER BY
    f.festival_year,
    pm.payment_method_name;
```

Αποτέλεσμα (αρχείο Query/Q01_out.txt):

(Φαίνονται τα έσοδα των πρώτων 3 φεστιβάλ ανά τρόπο πληρωμής)

festival_year int	payment_method_name varchar	total_earnings double
2016	Credit card	135
2016	Debit card	335
2017	Credit card	256
2017	Debit card	301
2018	Credit card	245
2018	Debit card	220
2018	Paypal	10

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

Query (αρχείο Query/Q02.sql):

```
SELECT DISTINCT
    f.festival_year,
    b.band_name,
    g.genre_name
FROM
    band b
JOIN band_subgenre bs ON b.band_id = bs.band_id
JOIN subgenre sg ON sg.subgenre_id = bs.subgenre_id
JOIN genre g ON g.genre_id = sg.genre_id
JOIN performance p ON p.band_id = b.band_id
JOIN event e ON e.event_id = p.event_id
JOIN festival f ON f.festival_year = e.festival_year
ORDER BY
    f.festival_year,
    b.band_name;
```

Αποτέλεσμα (αρχείο Query/Q02_out.txt):

(Φαίνονται τα πρώτα 7 αποτελέσματα του query για το είδος 'Jazz')










<input type="checkbox"/>	<input type="checkbox"/>	* festival_year int(11)	<input type="checkbox"/>	* band_name varchar(255)	<input type="checkbox"/>	* genre_name varchar(255)
<input type="checkbox"/>	>	2016		Neon Howl		Jazz
<input type="checkbox"/>	>	2016		Zayn		Jazz
<input type="checkbox"/>	>	2018		Arcade Fire		Jazz
<input type="checkbox"/>	>	2019		Muse		Jazz
<input type="checkbox"/>	>	2022		Ariana Grande		Jazz
<input type="checkbox"/>	>	2022		Jacob Anderson		Jazz
<input type="checkbox"/>	>	2022		Khalid		Jazz

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

Query (αρχείο Query/Q03.sql):

```
SELECT DISTINCT
    f.festival_year,
    a.artist_stage_name,
    COUNT(p.performance_id) AS warm_up_performances
FROM
    artist a
JOIN artist_band ab ON a.artist_id = ab.artist_id
JOIN band b ON b.band_id = ab.band_id
JOIN performance p ON p.band_id = b.band_id
JOIN event e ON e.event_id = p.event_id
JOIN festival f ON f.festival_year = e.festival_year
WHERE
    p.performance_type_id = 1 AND f.festival_year < 2025
GROUP BY
    f.festival_year,
    a.artist_id
HAVING
    COUNT(p.performance_id) > 2
ORDER BY
    f.festival_year,
    b.band_name;
```

Αποτέλεσμα (αρχείο Query/Q03_out.txt):
(Το query επιστρέφει μόνο 1 αποτέλεσμα)

	 	* festival_year int(11)	 	artist_stage_name varchar(255)	 	* warm_up_performar bigint(21)	 
		> 2016		Kesha		3	

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

Query (αρχείο Query/Q04.sql):

```
SELECT DISTINCT
  b.band_name,
  ROUND(AVG(ls.performance_score),2)AS avg_performance_score,
  ROUND(AVG(ls.stage_presence_score),2) AS avg_stage_presence_score
FROM
  likert_scale ls
JOIN reviews r ON ls.reviews_id = r.reviews_id
JOIN performance p ON r.performance_id = p.performance_id
JOIN band b ON b.band_id = p.band_id
GROUP BY
  b.band_id
ORDER BY
  avg_performance_score DESC,
  avg_stage_presence_score DESC;
```

Αποτέλεσμα (αρχείο Query/Q04_out.txt):

(Φαίνονται τα πρώτα 8 αποτελέσματα του query)

Q	band_name varchar	avg_performance_sco decimal	avg_stage_presence_s decimal
>	Måneskin	5.00	4.00
>	Jacob Anderson	5.00	2.00
>	Nina Simone	4.50	3.00
>	Keane	4.33	3.33
>	The Hives	4.00	4.50
>	Hugh Masekela	4.00	4.00
>	Gojira	4.00	4.00
>	Panic! At The Disco	4.00	4.00

Χρόνος εκτέλεσης:

Cost: 7ms

Για το ερώτημα 4, μας ζητείται να χρησιμοποιήσουμε εναλλακτικό Query Plan με force indexes. Ο κώδικας και ο χρόνος εκτέλεσης φαίνονται πιο κάτω

```
SELECT DISTINCT
  b.band_name,
  ROUND(AVG(ls.performance_score), 2) AS avg_performance_score,
  ROUND(AVG(ls.stage_presence_score), 2) AS avg_stage_presence_score
FROM
  likert_scale ls FORCE INDEX (reviews_id) -- Ensure index exists
JOIN reviews r FORCE INDEX (performance_id) ON ls.reviews_id = r.reviews_id
JOIN performance p FORCE INDEX (band_id) ON r.performance_id = p.performance_id
JOIN band b FORCE INDEX (PRIMARY) ON b.band_id = p.band_id
GROUP BY
  b.band_id
ORDER BY
  avg_performance_score DESC,
  avg_stage_presence_score DESC;
```

Cost: 7ms

Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι ίδιος με πριν. Ο λόγος που συμβαίνει αυτό είναι το γεγονός ότι στο query κάνουμε JOIN μόνο μεταξύ PRIMARY KEYS, τα οποία είναι ήδη indexed.

Ακολούθως δοκιμάζουμε διαφορετικές στρατηγικές join και εκτελούμε τον ίδιο κώδικα.

Δοκιμάζουμε Hash Join:

```
SET optimizer_switch='join_cache_hashed=on';
```

Cost: 8ms

Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι μεγαλύτερος κατά πολύ λίγο, κάτι που είναι λογικό καθώς τα δεδομένα δεν είναι πολλά και τα JOIN είναι indexed.

Δοκιμάζουμε Merge Join:

```
SET optimizer_switch='index_merge=on';
```

Cost: 16ms

Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι μεγαλύτερος, καθώς οι πίνακες δεν είναι ταξινομημένοι, και το Merge Join αναγκάζει τη βάση να τους ταξινομήσει πρώτα.




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

Query (αρχείο Query/Q05.sql):

```
SELECT
  a.artist_stage_name,
  COUNT(DISTINCT f.festival_year) AS festivals_performed
FROM
  artist a
JOIN artist_band ab ON a.artist_id = ab.artist_id
JOIN band b ON b.band_id = ab.band_id
JOIN performance p ON p.band_id = b.band_id
JOIN event e ON e.event_id = p.event_id
JOIN festival f ON f.festival_year = e.festival_year
WHERE
  (YEAR(CURDATE()) - a.artist_year_of_birth) < 30 AND f.festival_year < 2025
GROUP BY
  a.artist_id
ORDER BY
  festivals_performed DESC;
```

Αποτέλεσμα (αρχείο Query/Q05_out.txt):

(Φαίνονται τα πρώτα 8 αποτελέσματα του query)

<input type="checkbox"/>		artist_stage_name varchar(255)		* festivals_performed bigint(21)	
<input type="checkbox"/>	>	Olivia Rodrigo		4	
<input type="checkbox"/>	>	Arlo Parks		4	
<input type="checkbox"/>	>	Billie Eilish		3	
<input type="checkbox"/>	>	Khalid		3	
<input type="checkbox"/>	>	Lorde		2	
<input type="checkbox"/>	>	Aurelie		2	
<input type="checkbox"/>	>	Noah Kahan		2	
<input type="checkbox"/>	>	Lorde		2	

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

Query (αρχείο Query/Q06.sql):

```
SELECT
    v.visitor_name,
    v.visitor_surname,
    e.event_id,
    ROUND(AVG((
        ls.performance_score +
        ls.sound_light_quality_score +
        ls.stage_presence_score +
        ls.organization_score +
        ls.total_impression_score
    ) / 5 ), 2) AS avg_review_score
FROM
    reviews r
JOIN likert_scale ls ON r.reviews_id = ls.reviews_id
JOIN visitor v ON r.visitor_id = v.visitor_id
JOIN performance p ON r.performance_id = p.performance_id
JOIN event e ON p.event_id = e.event_id
GROUP BY
    r.visitor_id,
    e.event_id
ORDER BY
    avg_review_score DESC;
```

Αποτέλεσμα (αρχείο Query/Q06_out.txt):

(Φαίνονται τα πρώτα 6 αποτελέσματα του query)

Q	visitor_name varchar	visitor_surname varchar	event_id int	avg_review_score decimal
>	Luka	Doncic	3	5.00
>	Nicolò	Barella	12	5.00
>	Oliver	Kahn	33	5.00
>	Christopher	Nkunku	16	4.80
>	Rivaldo	Vítor	30	4.60
>	Victor	Osimhen	13	4.60

Χρόνος εκτέλεσης:

Cost: 10ms

Για το ερώτημα 6, μας ζητείται να χρησιμοποιήσουμε εναλλακτικό Query Plan με force indexes. Ο κώδικας και ο χρόνος εκτέλεσης φαίνονται πιο κάτω.

```
SELECT
  v.visitor_name,
  v.visitor_surname,
  p.event_id,
  ROUND(AVG((
    ls.performance_score +
    ls.sound_light_quality_score +
    ls.stage_presence_score +
    ls.organization_score +
    ls.total_impression_score
  ) / 5), 2) AS avg_review_score
FROM
  likert_scale ls FORCE INDEX (reviews_id)
JOIN reviews r FORCE INDEX (performance_id) ON r.reviews_id = ls.reviews_id
JOIN visitor v ON r.visitor_id = v.visitor_id
JOIN performance p FORCE INDEX (event_id) ON r.performance_id = p.performance_id
GROUP BY
  r.visitor_id,
  p.event_id
ORDER BY
  avg_review_score DESC;
```

Cost: 9ms

Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι μικρότερος από πριν με πολύ μικρή διαφορά. Ο λόγος που συμβαίνει αυτό είναι το γεγονός ότι στο query κάνουμε JOIN μόνο μεταξύ PRIMARY KEYS, τα οποία είναι ήδη indexed.

Δοκιμάζουμε Hash Join:

```
SET optimizer_switch='join_cache_hashed=on';
```

Cost: 9ms

Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι μικρότερος κατά πολύ λίγο, κάτι που είναι λογικό καθώς τα δεδομένα δεν είναι πολλά και τα JOIN είναι indexed.

Δοκιμάζουμε Merge Join:

```
SET optimizer_switch='index_merge=on';
```

Cost: 12ms

Παρατηρούμε και πάλι ότι ο χρόνος εκτέλεσης είναι μεγαλύτερος, καθώς οι πίνακες δεν είναι ταξινομημένοι, και το Merge Join αναγκάζει τη βάση να τους ταξινομήσει πρώτα.

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

Query (αρχείο Query/Q07.sql):

```
SELECT
  f.festival_year,
  AVG(level_of_experience.level_of_experience_id) AS lvl_exp
FROM
  festival f
JOIN event e ON e.festival_year = f.festival_year
JOIN stage s ON e.stage_id = s.stage_id
JOIN stage_staff ss ON s.stage_id = ss.stage_id
JOIN staff st ON ss.staff_id = st.staff_id
JOIN staff_role sr ON st.staff_role_id = sr.staff_role_id
JOIN level_of_experience ON st.level_of_experience = level_of_experience.level_of_experience_id
WHERE
  sr.staff_role_id = 1
GROUP BY
  f.festival_year
ORDER BY
  lvl_exp ASC
LIMIT 1;
```

Αποτέλεσμα (αρχείο Query/Q07_out.txt):

	festival_year int	lvl_exp decimal
>	2019	2.2857

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

Query (αρχείο Query/Q08.sql):

```
SELECT
    f.festival_year,
    f.festival_month,
    d.festival_day,
    s.staff_id,
    s.staff_name,
    s.staff_email,
    s.staff_phone
FROM
    staff s
CROSS JOIN (
    SELECT DISTINCT e.festival_year, e.festival_day
    FROM event e
) d
JOIN festival f ON f.festival_year = d.festival_year
WHERE
    s.staff_role_id = 3
    AND NOT EXISTS (
        SELECT 1
        FROM stage_staff ss
        JOIN event e ON ss.stage_id = e.stage_id
        WHERE e.festival_year = d.festival_year
            AND e.festival_day = d.festival_day
            AND ss.staff_id = s.staff_id
    )
ORDER BY
    f.festival_year,
    f.festival_month,
    d.festival_day,
    s.staff_name; 28ms
```

Αποτέλεσμα (αρχείο Query/Q08_out.txt):

(Φαίνονται τα πρώτα 8 αποτελέσματα του query. Τα αποτελέσματα είναι ταξινομημένα με βάση την ημερομηνία σε αύξουσα σειρά)

festival_year int	festival_month int	festival_day int	staff_id int	staff_name varchar	staff_email varchar	staff_phone varchar
2016	5	1	40	Alexandra Kouris	kouris@gmail.com	99223344
2016	5	1	21	Andreas Charalambous	charalambous@gmail.com	99112233
2016	5	1	19	Costas Hadjikyriacou	hadjikyriacou@gmail.com	99999000
2016	5	1	18	Despina Nicolaou	nicolaou@gmail.com	99888999
2016	5	1	22	Elena Kallis	kallis@gmail.com	99223344
2016	5	1	39	Lebron James	savva@gmail.com	99112233
2016	5	1	20	Marina Spyrou	spyrou@gmail.com	99000111
2016	5	2	40	Alexandra Kouris	kouris@gmail.com	99223344

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

Query (αρχείο Query/Q09.sql):

```
WITH attendance_counts AS (  
    SELECT  
        v.visitor_id,  
        v.visitor_name,  
        v.visitor_surname,  
        f.festival_year,  
        COUNT(DISTINCT t.event_id) AS events_attended  
    FROM ticket t  
    JOIN visitor v ON t.visitor_id = v.visitor_id  
    JOIN event e ON t.event_id = e.event_id  
    JOIN festival f ON e.festival_year = f.festival_year  
    WHERE t.validated = TRUE  
    GROUP BY v.visitor_id, v.visitor_name, v.visitor_surname, f.festival_year  
    HAVING COUNT(DISTINCT t.event_id) > 3  
)  
grouped_attendance AS (  
    SELECT  
        festival_year,  
        events_attended,  
        COUNT(*)  
    FROM attendance_counts  
    GROUP BY festival_year, events_attended  
    HAVING COUNT(*) > 1  
)  
SELECT  
    ac.visitor_id,  
    ac.visitor_name,  
    ac.visitor_surname,  
    ac.festival_year,  
    ac.events_attended  
FROM attendance_counts ac  
JOIN grouped_attendance ga  
    ON ac.festival_year = ga.festival_year  
    AND ac.events_attended = ga.events_attended  
ORDER BY ac.events_attended, ac.festival_year, ac.visitor_surname, ac.visitor_name;
```

Σε αυτό το query αρχικά κρατάμε στη μεταβλητή `attendance_counts` τα στοιχεία όλων των επισκεπτών που έχουν παρακολουθήσει πάνω από 3 παραστάσεις σε ένα φεστιβάλ. Στη συνέχεια, κρατάμε στην `grouped_attendance` τους επισκέπτες με κοινό αριθμό παρακολουθήσεων. Τέλος, από τα αποτελέσματα των δύο αυτών queries επιλέγουμε αυτά που έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων.

Αποτέλεσμα (αρχείο Query/Q09_out.txt):

visitor_id	visitor_name	visitor_surname	festival_year	events_attended
visitor_id	varchar	varchar	int	bigint
103	Xavi	Hernández	2023	4
102	Cristiano	Ronaldo	2023	4

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

Query (αρχείο Query/Q10.sql):




```
SELECT
    LEAST(g1.genre_name, g2.genre_name) AS genre_1,
    GREATEST(g1.genre_name, g2.genre_name) AS genre_2,
    COUNT(DISTINCT bs1.band_id) AS band_count
FROM band_subgenre bs1
JOIN subgenre sg1 ON bs1.subgenre_id = sg1.subgenre_id
JOIN genre g1 ON sg1.genre_id = g1.genre_id

JOIN band_subgenre bs2 ON bs1.band_id = bs2.band_id AND bs1.subgenre_id < bs2.subgenre_id
JOIN subgenre sg2 ON bs2.subgenre_id = sg2.subgenre_id
JOIN genre g2 ON sg2.genre_id = g2.genre_id

WHERE EXISTS (
    SELECT 1
    FROM performance p
    JOIN event e ON e.event_id = p.event_id
    JOIN festival f ON f.festival_year = e.festival_year
    WHERE p.band_id = bs1.band_id AND f.festival_year < 2025
)
AND g1.genre_id != g2.genre_id

GROUP BY genre_1, genre_2
ORDER BY band_count DESC
LIMIT 3;
```

Αποτέλεσμα (αρχείο Query/Q10_out.txt):

genre_1 	genre_2 	band_count 
EDM	Rock	12
Jazz	Pop	8
Country	Pop	6

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

Query (αρχείο Query/Q11.sql):

```
WITH artist_festival_counts AS (  
    SELECT  
        a.artist_id,  
        a.artist_stage_name,  
        COUNT(DISTINCT f.festival_year) AS festivals_participated  
    FROM  
        artist a  
    JOIN artist_band ab ON a.artist_id = ab.artist_id  
    JOIN band b ON ab.band_id = b.band_id  
    JOIN performance p ON p.band_id = b.band_id  
    JOIN event e ON e.event_id = p.event_id  
    JOIN festival f ON f.festival_year = e.festival_year  
    WHERE f.festival_year  
    GROUP BY a.artist_id, a.artist_name  
),  
max_festival_count AS (  
    SELECT MAX(festivals_participated) AS max_festivals  
    FROM artist_festival_counts  
)  
SELECT  
    AFC.artist_id,  
    AFC.artist_stage_name,  
    AFC.festivals_participated  
FROM  
    artist_festival_counts AFC  
JOIN  
    max_festival_count MFC  
    ON AFC.festivals_participated < MFC.max_festivals - 5  
ORDER BY  
    AFC.festivals_participated DESC;
```

Αποτέλεσμα (αρχείο Query/Q11_out.txt):

artist_id	artist_stage_name	festivals_participated
int	varchar	bigint

Παρατηρούμε ότι το query δεν επιστρέφει κάτι. Με μια παραλλαγή στον έλεγχο που κάνουμε παρατηρούμε ότι ο λόγος που το query δεν επέστρεφε αποτελέσματα ήταν τα δεδομένα που δώσαμε στη βάση, και όχι η υλοποίηση του ερωτήματος. Αλλάζουμε το 5 σε 4 (4 λιγότερες φορές) και παίρνουμε τα κάτωθι αποτελέσματα: (Φαίνονται τα πρώτα 8)

```

JOIN
  max_festival_count mfc
ON afc.festivals_participated < mfc.max_festivals - 4

```

artist_id	artist_stage_name	festivals_participated
int	varchar	bigint
143	Wiz Khalifa	1
144	Liam Payne	1
145	Kylie Minogue	1
146	Justin Bieber	1
147	Shania Twain	1
36	Lil Xan	1
148	Slauson Malone 1	1
37	SZA	1

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

Query (αρχείο Query/Q12.sql):

```
SELECT
    f.festival_year,
    e.festival_day,
    CEIL(COUNT(t.EAN_13) * 0.05) AS required_security_staff,
    CEIL(COUNT(t.EAN_13) * 0.02) AS required_secondary_staff,
    COUNT(DISTINCT e.event_id) * 2 AS required_technicians
FROM
    ticket t
JOIN event e ON t.event_id = e.event_id
JOIN festival f ON e.festival_year = f.festival_year
GROUP BY
    f.festival_year,
    e.festival_day
ORDER BY
    f.festival_year,
    e.festival_day;
```

Αποτέλεσμα (αρχείο Query/Q12_out.txt):
(Φαίνονται τα πρώτα 8 αποτελέσματα του query).

festival_year int	festival_day int	required_security_staf decimal	required_secondary_s decimal	required_technicians bigint
2016	1	1	1	2
2016	2	1	1	2
2016	3	1	1	2
2016	4	1	1	2
2017	1	1	1	2
2017	2	1	1	2
2017	3	1	1	2
2017	4	1	1	2

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

Query (αρχείο Query/Q13.sql):

```
SELECT
  a.artist_id,
  a.artist_stage_name,
  COUNT(DISTINCT fl.continent) AS unique_continents
FROM
  artist a
JOIN artist_band ab ON a.artist_id = ab.artist_id
JOIN band b ON ab.band_id = b.band_id
JOIN performance p ON p.band_id = b.band_id
JOIN event e ON e.event_id = p.event_id
JOIN festival f ON f.festival_year = e.festival_year
JOIN festival_location fl ON f.location_id = fl.location_id
WHERE f.festival_year < 2025
GROUP BY
  a.artist_id, a.artist_name
HAVING
  COUNT(DISTINCT fl.continent) >= 3
ORDER BY
  unique_continents DESC;
```

Αποτέλεσμα (αρχείο Query/Q13_out.txt):
(Φαίνονται όλα τα αποτελέσματα)

artist_id int	artist_stage_name varchar	unique_continents bigint
136	Panic! At The Disco	3
137	Khalid	3
140	Kesha	3
131	Arlo Parks	3
132	Alice Merton	3
133	Twenty One Pilots	3
135	Raleigh Ritchie	3

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

Query (αρχείο Query/Q14.sql):

```
WITH genre_performances AS (
    SELECT
        f.festival_year,
        g.genre_name,
        COUNT(*) AS performance_count
    FROM
        performance p
    JOIN event e ON e.event_id = p.event_id
    JOIN festival f ON f.festival_year = e.festival_year
    JOIN band b ON b.band_id = p.band_id
    JOIN band_subgenre bs ON bs.band_id = b.band_id
    JOIN subgenre sg ON sg.subgenre_id = bs.subgenre_id
    JOIN genre g ON g.genre_id = sg.genre_id
    WHERE f.festival_year < 2025
    GROUP BY
        f.festival_year, g.genre_name
    HAVING COUNT(*) >= 3
),
matched_years AS (
    SELECT
        g1.genre_name,
        g1.festival_year AS year1,
        g2.festival_year AS year2,
        g1.performance_count
    FROM
        genre_performances g1
    JOIN genre_performances g2
        ON g1.genre_name = g2.genre_name
        AND g1.festival_year = g2.festival_year - 1
        AND g1.performance_count = g2.performance_count
)
SELECT
    genre_name,
    year1 AS first_year,
    year2 AS second_year,
    performance_count
FROM
    matched_years
ORDER BY
    genre_name, year1;
```

Αποτέλεσμα (αρχείο Query/Q14_out.txt):

Το μόνο είδος που πληροί τα ζητούμενα του ερωτήματος είναι το είδος *Jazz*.

genre_name	first_year	second_year	performance_count
varchar	int	int	bigint
Jazz	2022	2023	4

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

Query (αρχείο Query/Q15.sql):

```
SELECT
  CONCAT(v.visitor_name, ' ', v.visitor_surname) AS visitor_name,
  b.band_name,
  SUM(ls.performance_score + ls.stage_presence_score + ls.total_impression_score) AS total_review_score
FROM
  reviews r
JOIN
  likert_scale ls ON r.reviews_id = ls.reviews_id
JOIN
  visitor v ON r.visitor_id = v.visitor_id
JOIN
  performance p ON r.performance_id = p.performance_id
JOIN
  event e ON p.event_id = e.event_id
JOIN
  band b ON p.band_id = b.band_id
WHERE e.festival_year < 2025
GROUP BY
  v.visitor_id, b.band_id
ORDER BY
  total_review_score DESC
LIMIT 5;
```

Αποτέλεσμα (αρχείο Query/Q15_out.txt):

visitor_name varchar	band_name varchar	total_review_score decimal
Stephen Curry	Neon Howl	21
Declan Rice	Radiohead	15
Rúben Dias	Sigur Rós	15
Luka Doncic	Neon Howl	15
Oliver Kahn	Justin Bieber	15