Pulse University Music Festival Database

Βάσεις Δεδομένων - Εξαμηνιαία Εργασία

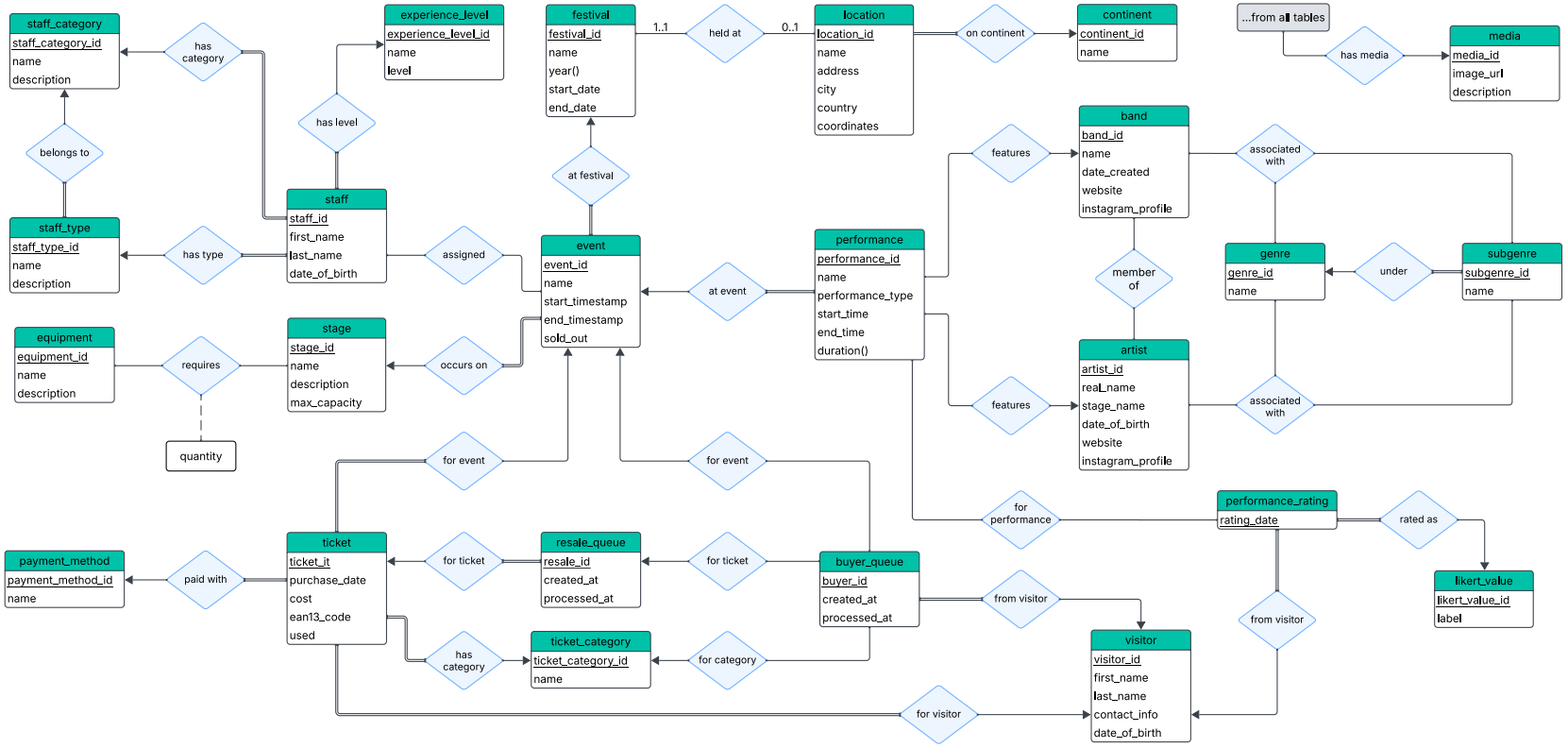
Εαρινό Εξάμηνο 2024-2025

Ομάδα Project 100

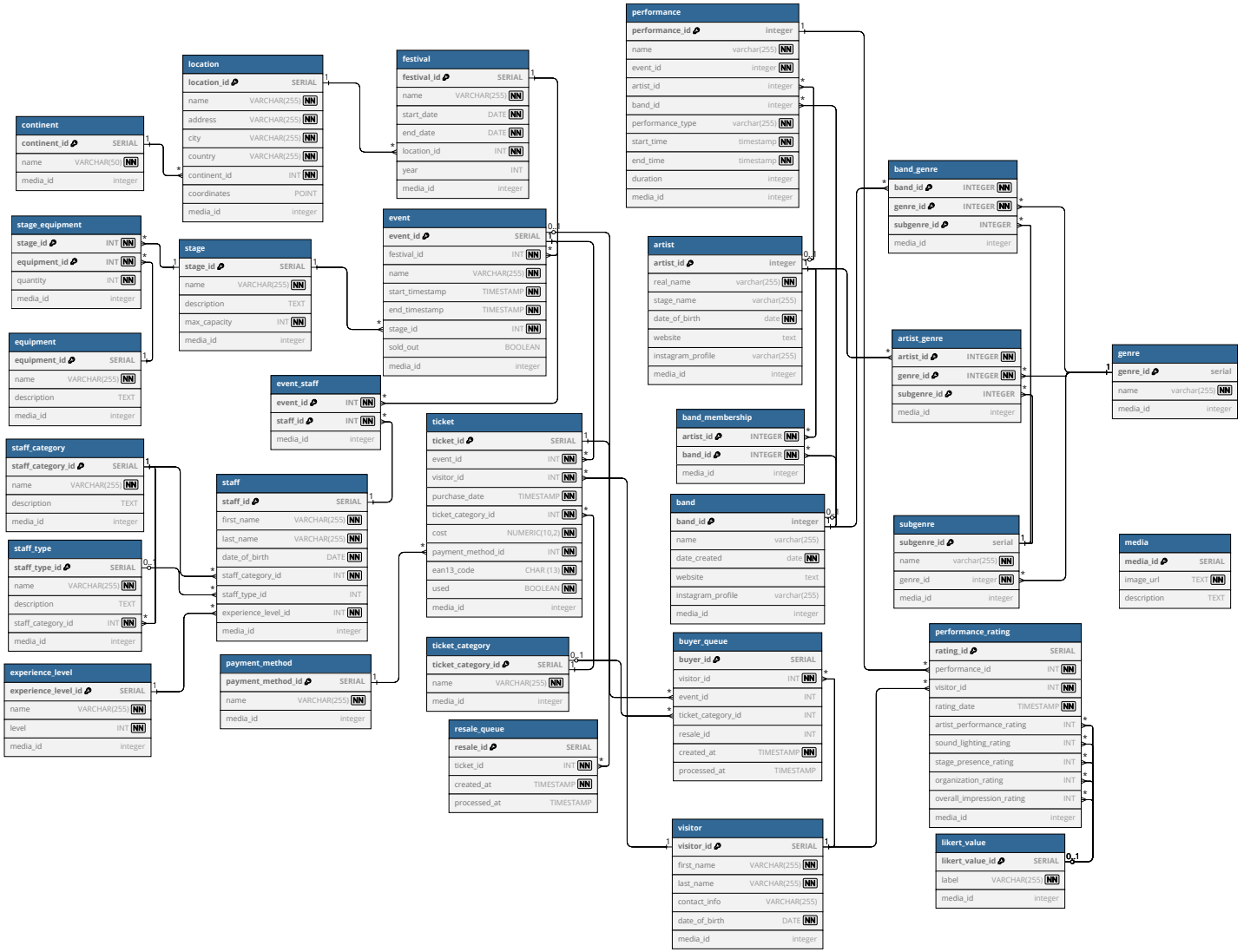
ΝΙΚΟΛΑΟΣ ΓΙΑΝΝΟΠΟΥΛΟΣ	03122086
ΣΤΑΥΡΟΣ ΠΟΝΤΙΚΗΣ	03123727
ΠΑΣΧΑΛΗΣ ΣΑΡΡΑΣ	03121642

Contents

Ενότητα 1: Διάγραμμα ER



Ενότητα 2: Σχεσιακό Διάγραμμα



Ενότητα 3: Σύντομη Περιγραφή Οντοτήτων και Σχέσεων

3.1 Οντότητες

Η βάση δεδομένων του φεστιβάλ **Pulse University** περιλαμβάνει οντότητες που μοντελοποιούν τις βασικές και βοηθητικές πληροφορίες που είναι απαραίτητες για την διεξαγωγή των εκδηλώσεων:

- **Βασικές Οντότητες**: Festival, Event, Performance, Artist, Band, Visitor, Ticket, Staff, Genre, Location, Stage, Equipment, Performance_Rating
- **Βοηθητικές Οντότητες**: Subgenre, Ticket_Category, Payment_Method, Experience_Level, Staff_Type, Staff_Category, Likert_Value, Continent, Media
- **Συνδετικές Οντότητες**: Band_Membership, Artist_Genre, Band_Genre, Stage_Equipment, Event_Staff, Resell_Queue, Buyer_Queue

Ακολουθεί περιγραφή βασικών οντοτήτων:

- **Festival**
  Κάθε φεστιβάλ χαρακτηρίζεται από μοναδικό festival_id, όνομα, έτος διεξαγωγής και ημερομηνίες έναρξης και λήξης. Συνδέεται με μία συγκεκριμένη Location.
- **Location**
  Περιλαμβάνει γεωγραφικά και διοικητικά στοιχεία, όπως όνομα, διεύθυνση, πόλη, χώρα, συντεταγμένες, και continent_id. Χρησιμοποιείται για τον εντοπισμό της τοποθεσίας κάθε φεστιβάλ.
- **Event**
  Πρόκειται για μια οργανωμένη παράσταση που γίνεται σε συγκεκριμένο stage, σχετίζεται με κάποιο Festival, και περιλαμβάνει ένα ή περισσότερα performances. Έχει όνομα, ημερομηνία, ώρα έναρξης/λήξης και ένδειξη sold_out.
- **Stage**
  Αντιστοιχεί στη φυσική σκηνή όπου λαμβάνει χώρα ένα event. Έχει όνομα, περιγραφή και max_capacity.
- **Performance**
  Κάθε εμφάνιση είναι συνδεδεμένη με είτε με έναν artist είτε με ένα Band, έχει καθορισμένο τύπο εμφάνισης (π.χ. warm-up, headline), χρονική διάρκεια και συσχέτιση με συγκεκριμένο Event.
- **Artist**
  Εκπροσωπεί μεμονωμένους καλλιτέχνες με πραγματικό και καλλιτεχνικό όνομα, ημερομηνία γέννησης, μουσικά είδη, προαιρετικά links (website, instagram).

Μπορούν να εμφανίζονται σόλο, να συμμετέχουν σε συγκροτήματα και να σχετίζονται με ένα ή πολλά μουσικά είδη.

- **Band**
  Αναπαριστά μουσικά συγκροτήματα με όνομα, ημερομηνία δημιουργίας, μουσικά είδη και προαιρετικά links. Περιέχει πολλούς καλλιτέχνες-μέλη και μπορεί να εμφανίζεται σε παραστάσεις ως σύνολο.
- **Visitor**
  Περιέχει τα προσωπικά στοιχεία (όνομα, επίθετο, ημερομηνία γέννησης, email) των επισκεπτών που έχουν αγοράσει κάποιο συγκεκριμένο Ticket και μπορούν να αξιολογήσουν Performance.
- **Ticket**
  Περιλαμβάνει κατηγορία εισιτηρίου, κόστος, τρόπο πληρωμής, κωδικό EAN-13 και ένδειξη αν χρησιμοποιήθηκε σε συγκεκριμένο event_id απο κάποιον visitor_id
- **Performance_Rating**
  Καταγράφει αξιολογήσεις γραμμένες σε μια χρονική στιγμή από επισκέπτες για συγκεκριμένες εμφανίσεις, με βάση τα πέντε κριτήρια της κλίμακας Likert: Ερμηνεία, Ήχος/Φωτισμός, Σκηνική Παρουσία, Οργάνωση, Συνολική Εντύπωση.
- **Staff**
  Περιέχει ονοματεπώνυμο, ημερομηνία γέννησης για κάθε μέλος προσωπικού, το οποίο κατηγοριοποιείται βάσει Staff_category, Staff_type και experience_level για την εύρυθμη λειτουργία κάθε Event.
- **Equipment**
  Κάθε στοιχείο τεχνικού εξοπλισμού (π.χ. φώτα, μικρόφωνα, ηχεία) καταγράφεται με equipment_id, όνομα και περιγραφή. Η σύνδεση εξοπλισμού με σκηνές γίνεται μέσω της συνδετικής οντότητας Stage_equipment, η οποία περιλαμβάνει και το πεδίο quantity, για την ποσότητα κάθε εξοπλισμού που απαιτείται.

3.2 Σχέσεις

Οι βασικές σχέσεις της βάσης περιγράφονται συνοπτικά παρακάτω:

- Κάθε **Festival** πραγματοποιείται σε ένα **Location**, και αποτελείται από πολλά **Event**.
- Κάθε **Event** διεξάγεται σε ένα **Stage** και περιλαμβάνει πολλά **Performance**.
- Κάθε **Performance** εκτελείται από **Artist** ή **Band**, αλλά όχι ταυτόχρονα σε διαφορετικά **Events**.
- Οι **Artist** μπορεί να είναι μέλη σε πολλαπλά **Band**, μέσω της **Band_Membership**.

- Κάθε **Stage** απαιτεί συγκεκριμένο **Equipment** και **Staff**, ορίζοντας έτσι σχέσεις **Stage_Equipment** και **Event_Staff**.
- Οι **Visitor** αγοράζουν **Ticket** για **Events**, και μπορούν να μεταπωλούν εισιτήρια μέσω **Resell_Queue**, ή να μπουν σε **Buyer_Queue** εφόσον το **event** είναι **sold_out**.
- Οι **Visitor** μπορούν να αξιολογούν τις **Performance** με τη βοήθεια της **Performance_Rating** εφόσον το εισιτήριο είναι used.
- Τα μουσικά χαρακτηριστικά των καλλιτεχνών καθορίζονται μέσω των **Artist_Genre** και **Band_Genre**, που συνδέονται με **Genre** και **Subgenre**.
- Οι **Ticket** κατατάσσονται σε κάποιο τύπο **Ticket_Category** και αγοράζονται μέσω συγκεκριμένου **Payment_Method**.

Ενότητα 4: Indexing

Στη βάση δεδομένων του Pulse University Festival, τα **indexes** παίζουν καθοριστικό ρόλο στην αποδοτική εκτέλεση των ερωτημάτων, ιδιαίτερα λόγω του μεγάλου αριθμού εγγραφών και των πολύπλοκων συνδέσεων (π.χ. πωλήσεις εισιτηρίων, αξιολογήσεις, σχέσεις καλλιτεχνών, ουρές μεταπώλησης κ.ά.).

- **Primary Keys**
  continent.continent_id, location.location_id, festival.festival_id, stage.stage_id, equipment.equipment_id, stage_equipment.stage_id, stage_equipment.equipment_id, staff_category.staff_category_id, staff_type.staff_type_id, experience_level.experience_level_id, staff.staff_id, event.event_id, event_staff.event_id, event_staff.staff_id, genre.genre_id, subgenre.subgenre_id, artist.artist_id, band.band_id, artist_genre.artist_id, artist_genre.genre_id, artist_genre.subgenre_id, band_genre.band_id, band_genre.genre_id, band_genre.subgenre_id, band_membership.artist_id, band_membership.band_id, performance.performance_id, ticket_category.ticket_category_id, payment_method.payment_method_id, visitor.visitor_id, ticket.ticket_id, resale_queue.resale_id, buyer_queue.buyer_id, likert_value.likert_value_id, performance_rating.rating_id

Επιπλέον, προσθέσαμε μερικά indexes τύπου unique, όπου κρίναμε πως το αντίστοιχο attribute πρέπει να είναι μοναδικό και συγκριμένα:

continent.name, festival.location_id, festival.year, subgenre.(genre_id, name), artist.instagram_profile, band.instagram_profile, ticket.ean13_code, ticket.(visitor_id, event_id), performance_rating.(performance_id, visitor_id)

Τέλος προσθέσαμε τα παρακάτω **indexes** για να έχουμε γρήγορη πρόσβαση σε στοιχεία μέσα σε queries και σε triggers που χρησιμοποιούμε συχνά.

artist.stage_name, band.name, performance.event_id, performance.start_time, performance.end_time, genre.name, subgenre.name, subgenre.genre_id, artist_genre.artist_id, artist_genre.genre_id, artist_genre.subgenre_id, band_genre.band_id, band_genre.genre_id, band_genre.subgenre_id, event_staff.staff_id, performance.artist_id, performance.band_id, event.stage_id, event.start_timestamp, event.end_timestamp, ticket.event_id, performance.event_id, performance.start_time, performance.end_time, artist_genre.genre_id, artist_genre.subgenre_id, band_genre.genre_id, band_genre.subgenre_id, buyer_queue.event_id, buyer_queue.ticket_category_id, resale_queue.ticket_id, buyer_queue.resale_id

Ενότητα 5: Constraints, Check, Triggers and Other

Η σωστή λειτουργία της βάσης διασφαλίζεται μέσω περιορισμών ακεραιότητας, λογικών ελέγχων (CHECK), και triggers όπου χρειάζεται. Ακολουθεί αναλυτική παρουσίαση των βασικών κανόνων που εφαρμόστηκαν.

5.1 Περιορισμοί κλειδιών (Primary και Foreign Keys)

- **Πρωτεύοντα Κλειδιά (Primary Keys)** ορίστηκαν σε όλες τις βασικές και συνδετικές οντότητες.
- **Ξένα Κλειδιά (Foreign Keys)** διασφαλίζουν ότι οι σχέσεις (π.χ. Performance.event_id) αναφέρονται σε υπαρκτές εγγραφές. Πολλά foreign keys ορίζονται με ON DELETE CASCADE όπου η διαγραφή λογικά συνεπάγεται διαγραφή εξαρτώμενων π.χ. Visitor → Performance_rating.

5.2 Περιορισμοί Μοναδικότητας (Unique)

Εξασφαλίζουν ότι συγκεκριμένα πεδία δεν επαναλαμβάνονται σε διαφορετικές εγγραφές.

Παράδειγμα:

```
CREATE TABLE continent (
    continent_id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);
```

Παράδειγμα προσθήκης που χρειάστηκε να γίνει αργότερα:

```
ALTER TABLE band_membership ADD CONSTRAINT unique_band_membership UNIQUE (artist_id, band_id);
```

Παραπάνω αναφέραμε όλα τα UNIQUE attributes εδώ θα κάνουμε μια μικρή αναφορά στην χρήση τους:

- UNIQUE (continent.name): Διασφαλίζει ότι δεν υπάρχουν δύο ήπειροι με το ίδιο όνομα.
- UNIQUE (festival.location_id): Κάθε τοποθεσία μπορεί να φιλοξενεί το πολύ ένα φεστιβάλ.
- UNIQUE (festival.year): Διασφαλίζει ότι δεν διεξάγονται δύο φεστιβάλ την ίδια χρονιά.
- UNIQUE (band_membership.artist_id, band_id): Εξασφαλίζει ότι δεν θα δηλωθεί δύο φορές ίδια συμμετοχή.
- UNIQUE (artist.instagram_profile): Κάθε προφίλ Instagram καλλιτέχνη μοναδικό.
- UNIQUE (band.instagram_profile): Κάθε προφίλ Instagram συγκροτήματος μοναδικό.
- UNIQUE (ticket.ean13_code): Κάθε barcode εισιτηρίου μοναδικό.
- UNIQUE (ticket.visitor_id, ticket.event_id): Κάθε επισκέπτης αγοράζει το πολύ ένα εισιτήριο ανά εκδήλωση.

5.3 Περιορισμοί Ελέγχου (CHECK Constraints)

Ελέγχουν την εγκυρότητα των δεδομένων βάσει συγκεκριμένων κανόνων.

Παράδειγμά: Διάρκεια παράστασης ≤ 3 ώρες

```
CHECK (end_time - start_time <= INTERVAL '3 hours')
```

Παρακάτω αναφέρονται Checks που χρησιμοποιούμε στον κώδικά μας και την χρήση τους:

- **CHECK (stage.max_capacity > 0):** Η μέγιστη χωρητικότητα μιας σκηνής πρέπει να είναι μεγαλύτερη από το μηδέν.
- **CHECK (stage_equipment.quantity >= 0):** Η ποσότητα εξοπλισμού δεν μπορεί να είναι αρνητική.

- **CHECK (subgenre.genre_id, subgenre.name) UNIQUE:** Κάθε όνομα υπό είδους μουσικής είναι μοναδικό μέσα στο κύριο είδος του.
- **CHECK (performance.start_time < performance.end_time):** Η ώρα έναρξης πρέπει να είναι πριν από την ώρα λήξης.
- **CHECK (performance.end_time - performance.start_time <= INTERVAL '3 hours'):** Η διάρκεια των εμφανίσεων δεν μπορεί να υπερβαίνει τις 3 ώρες.
- **CHECK ((artist_id IS NOT NULL AND band_id IS NULL) OR (band_id IS NOT NULL AND artist_id IS NULL)):** Σε μια εμφάνιση μπορεί να πάρει μέρος μόνο είτε ένας καλλιτέχνης είτε συγκρότημα, ποτέ και τα δύο μαζί.
- **CHECK (ean13_code):** Εξασφαλίζει 13 ψηφία και έγκυρο ψηφιακό άθροισμα (mod 10).
- **CHECK (buyer_queue.ck_buyer_request_exclusive):** Είτε ζητάς συγκεκριμένο resale (resale_id), είτε event+κατηγορία, όχι και τα δύο μαζί.

5.4 Triggers για Επιχειρησιακούς Κανόνες

Για τον καλύτερο έλεγχο και αποφυγή σφαλμάτων υλοποιήσαμε τα παρακάτω triggers μαζί με τα αντίστοιχα functions, είτε υλοποιούν κάποια αλλαγή σε συγκεκριμένους πίνακες τις βάσεις (Παράδειγμα sold_out column true), είτε κάνουν έλεγχο και επιστρέφουν κατάλληλα μηνύματα στον χρήστη σε περίπτωση λάθους.

Παράδειγμα trigger που ελέγχει και απαγορεύει την διαγραφή οποιουδήποτε φεστιβάλ:

```
CREATE TRIGGER trg_block_festival_delete
BEFORE DELETE ON festival
FOR EACH ROW
EXECUTE FUNCTION prevent_festival_deletion();
```

Όπως φαίνεται το trigger αυτό καλεί την συνάρτηση prevent_festival_deletion() που είναι η παρακάτω:

```
CREATE OR REPLACE FUNCTION prevent_festival_deletion()
RETURNS TRIGGER AS $$
BEGIN
  IF TRUE THEN
    RAISE EXCEPTION 'Festival cannot be deleted!';
  END IF;
  RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Στο schema της βάσης μας υλοποιήθηκαν τα παρακάτω triggers:

- **trg_block_festival_delete:** Αποτρέπει την διαγραφή οποιουδήποτε φεστιβάλ με μήνυμα σφάλματος.
- **trg_check_staff_type_category:** Ελέγχει ότι ο τύπος προσωπικού ανήκει στην ίδια κατηγορία που δηλώνεται στον πίνακα staff.
- **trg_check_event_festival_range:** Βεβαιώνει ότι οι ώρες της παράστασης εμπίπτουν στο χρονικό διάστημα του φεστιβάλ (00:00:00–23:59:59).
- **trg_block_event_delete:** Απαγορεύει τη διαγραφή παράστασης.
- **trg_check_overlap_timestamp:** Εξασφαλίζει ότι δύο παραστάσεις δεν επικαλύπτονται στον ίδιο χώρο.
- **trg_check_staff_overlap:** Ελέγχει ότι ένας υπάλληλος δεν έχει ανατεθεί σε ταυτόχρονες εκδηλώσεις.
- **trg_validate_artist_genre:** Ελέγχει ότι τα subgenre που συνδέονται με τον artist είναι υποκατηγορία του genre που επίσης έχουμε συνδέσει.
- **trg_validate_band_genre:** Ελέγχει ότι τα subgenre που συνδέονται με το band είναι υποκατηγορία του genre που επίσης έχουμε συνδέσει.
- **trg_prevent_middle_delete:** Για τον έλεγχο της λογικής που αφορά ότι η ή η τελευταία παράσταση ενός event μπορεί να διαγραφεί και οι ενδιάμεσες μπλοκάρονται.
- **trg_performer_double_booking:** Ελέγχει ότι ο καλλιτέχνης / συγκρότημα δεν εμφανίζεται σε δύο διαφορετικά events την ίδια ώρα.
- **trg_check_overlap_and_breaks**: Εξασφαλίζει ότι κάθε εμφάνιση είναι μέσα στα όρια του event και ότι τα διαλείμματα είναι τουλάχιστον 5 και το πολύ 30 λεπτά.
- **trg_check_performer_years**: Δεν επιτρέπει έναν καλλιτέχνη ή συγκρότημα να εμφανιστεί πάνω από τρεις συνεχόμενες χρονιές.
- **trg_check_ticket_capacity**: Ελέγχει ότι το πλήθος των πωλημένων εισιτηρίων δεν ξεπερνά τη χωρητικότητα της σκηνής.
- **trg_check_vip_limit**: Διασφαλίζει ότι τα VIP εισιτήρια δεν υπερβαίνουν το 10% της συνολικής χωρητικότητας.
- **trg_update_event_sold_out**: Κάνει update την στήλη event.sold_out όταν τα εισιτήρια είναι ίσα σε αριθμό με το max_capacity
- **trg_chk_rating_allowed** Ελέγχει ότι ο επισκέπτης έχει ήδη χρησιμοποιήσει εισιτήριο για το event της εμφάνισης που βαθμολογεί.
- **trg_process_resale_listing**: Ελέγχει την εγκυρότητα αίτησης μεταπώλησης του εισιτηρίου, ψάχνει στην λίστα των αγοραστών για πιθανό match και σε περίπτωση match εκτελεί τη μεταπώληση.

- **trg_cleanup_resale_on_ticket_used**: Σε περίπτωση που ένα εισιτήριο βρίσκεται στην λίστα μεταπώλησης και χρησιμοποιηθεί, τότε αυτόματα διαγράφεται από τη λίστα μεταπώλησης.
- **trg_process_buyer_request**: Ελέγχει την εγκυρότητα του ενδιαφέροντος αγοραστή, ψάχνει την λίστα μεταπώλησης, είτε σε συγκεκριμένο resale ticket είτε για συγκεκριμένο event + ticket category και σε περίπτωση match εκτελεί την μεταπώληση.

Όλα τα παραπάνω triggers, σε περίπτωση που ενεργοποιηθούν, εκτελούν το αντίστοιχο function, το οποίο κάνει τους απαραίτητους ελέγχους, επιστρέφει μηνύματα στο χρήστη και πραγματοποιεί αλλαγές στα δεδομένα.

Ιδιαίτερα, όμως, το **function check_event_staff_coverage(p_event_id INT)** δεν εκτελείται από κάποιο trigger, αλλά οποιαδήποτε στιγμή από τον χρήστη. Ο περιορισμός για τον αριθμό προσωπικού ασφαλείας και βοηθητικού προσωπικού δεν μπορεί να ελεγχτεί πριν την εισαγωγή όλων των δεδομένων. Γι' αυτό, ο χρήστης μπορεί οποιαδήποτε στιγμή να εκτελέσει την συνάρτηση αυτή δίνοντας της ως είσοδο κάποιο συγκεκριμένο event και αυτή επιστρέφει κατάλληλο μήνυμα δίνοντας πληροφορίες για την επίτευξη (ή μη) των περιορισμών.

5.5 Προσθήκη πίνακα για αποθήκευση εικόνων για όλες τις οντότητες

Για την μελλοντική κατασκευή ιστοσελίδας, και για να αρχίσουμε την συλλογή εικόνων για κάθε οντότητα του συστήματος, προσθέσαμε έναν ανεξάρτητο πίνακα MEDIA. Ο πίνακας έχει τα παρακάτω attributes:

- **media_id** (Primary Key)
- **image_url** με την τοποθεσία (URL) που είναι αποθηκευμένο το αρχείο της εικόνας
- **description** κείμενο που περιγράφει την αντίστοιχη εικόνα.

Στη συνέχεια, περάσαμε σε όλες τις βασικές οντότητες του μοντέλου (φεστιβάλ, καλλιτέχνης, συγκρότημα, εξοπλισμός κτλ.) ένα προαιρετικό foreign key *media_id*. Η προσθήκη έγινε με την χρήση ALTER TABLE. Παράδειγμα:

ALTER TABLE visitor ADD COLUMN media_id INT REFERENCES media(media_id);

Με αυτή την επέκταση διατηρούμε το σχήμα όπως πριν σε περίπτωση που θέλουμε να το επαναφέρουμε εύκολα, και αποθηκεύουμε συγκεντρωτικά όλες τις εικόνες σε έναν μόνο πίνακα. Έτσι η βάση καλύπτει πλήρως την προοπτική δημιουργίας ιστοσελίδας όπου κάθε εγγραφή θα εμφανίζεται μαζί με την αφίσα, τη φωτογραφία ή το εικονίδιο που της αντιστοιχεί.

Ενότητα 6: install.sql, load.sql

Ενδεικτικά επισυνάπτουμε μέρη του *install.sql* script για την δημιουργία δύο από τους πιο κύριους πίνακες **event** και **performance.**

Πίνακας Event:

```
 CREATE TABLE event (
    event_id SERIAL PRIMARY KEY,
    festival_id INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    start_timestamp TIMESTAMP NOT NULL,
    end_timestamp TIMESTAMP NOT NULL,
    stage_id INT NOT NULL,
    sold_out BOOLEAN,
    FOREIGN KEY (festival_id) REFERENCES festival(festival_id),
    FOREIGN KEY (stage_id) REFERENCES stage(stage_id)
);
```

Πίνακας Performance

```
CREATE TABLE performance (
    performance_id integer DEFAULT nextval('performance_id_seq'::regclass) PRIMARY KEY,
    name varchar(255) NOT NULL,
    event_id integer NOT NULL,
    artist_id integer,
    band_id integer,
    performance_type varchar(255) NOT NULL,
    start_time timestamp NOT NULL,
    end_time timestamp NOT NULL,
    duration integer GENERATED ALWAYS AS ((EXTRACT(EPOCH FROM end_time - start_time) /
60)::integer) STORED, --in minutes
    FOREIGN KEY (artist_id) references artist (artist_id),
    FOREIGN KEY (band_id) references band (band_id),
    CHECK (start_time < end_time),
    CHECK (end_time - start_time <= INTERVAL '3 hours'),
    CHECK (
        (artist_id IS NOT NULL AND band_id IS NULL) OR
        (band_id IS NOT NULL AND artist_id IS NULL)
    )
);
```

Παραδείγματα από insert statements που χρησιμοποιήθηκαν:

```
INSERT INTO "event" (festival_id, "name", start_timestamp, end_timestamp, stage_id, sold_out,
media_id) VALUES
(1, 'Tomorrowland Day1 Session1', '2014-07-18 12:00:00.000', '2014-07-18 18:00:00.000', 1,
false, 621),
(1, 'Tomorrowland Day1 Session2', '2014-07-18 17:00:00.000', '2014-07-18 23:00:00.000', 2,
false, 622),
```

```
(1, 'Tomorrowland Day2 Session1', '2014-07-19 12:00:00.000', '2014-07-19 21:00:00.000', 1,
false, 623),
(1, 'Tomorrowland Day2 Session2', '2014-07-19 17:00:00.000', '2014-07-19 23:30:00.000', 2,
false, 624),
(1, 'Tomorrowland Day3 Session1', '2014-07-20 09:00:00.000', '2014-07-20 16:00:00.000', 1,
false, 625),
(1, 'Tomorrowland Day3 Session2', '2014-07-20 14:00:00.000', '2014-07-20 21:00:00.000', 2,
false, 626),
(2, 'New Orleans Jazz & Heritage Festival Day1 Session1', '2015-04-24 10:00:00.000', '2015-04-
24 16:00:00.000', 3, false, 627),
(2, 'New Orleans Jazz & Heritage Festival Day1 Session2', '2015-04-24 15:00:00.000', '2015-04-
24 19:00:00.000', 4, false, 628),
(2, 'New Orleans Jazz & Heritage Festival Day1 Session3', '2015-04-24 20:00:00.000', '2015-04-
24 23:00:00.000', 5, false, 629),
(2, 'New Orleans Jazz & Heritage Festival Day2 Session1', '2015-04-25 10:00:00.000', '2015-04-
25 16:00:00.000', 3, false, 630),
(2, 'New Orleans Jazz & Heritage Festival Day2 Session2', '2015-04-25 15:00:00.000', '2015-04-
25 19:00:00.000', 4, false, 631),
(2, 'New Orleans Jazz & Heritage Festival Day2 Session3', '2015-04-25 20:00:00.000', '2015-04-
25 23:00:00.000', 5, false, 632),
(2, 'New Orleans Jazz & Heritage Festival Day3 Session1', '2015-04-26 10:00:00.000', '2015-04-
26 16:00:00.000', 3, false, 633),
(2, 'New Orleans Jazz & Heritage Festival Day3 Session2', '2015-04-26 15:00:00.000', '2015-04-
26 19:00:00.000', 4, false, 634),
(2, 'New Orleans Jazz & Heritage Festival Day3 Session3', '2015-04-26 20:00:00.000', '2015-04-
26 23:00:00.000', 5, false, 635),
(3, 'Osheaga Day1 Session1', '2016-07-29 12:00:00.000', '2016-07-29 18:00:00.000', 6, false,
636)
```

και για τα performances:

```
INSERT INTO performance ("name", event_id, artist_id, band_id, performance_type, start_time,
end_time, media_id) VALUES
('Acoustic Spectrum', 1, 40, NULL, 'Headline', '2014-07-18 12:00:00', '2014-07-18 12:33:00',
954),
('Electric Groove', 1, NULL, 55, 'Warm Up', '2014-07-18 12:58:00', '2014-07-18 14:59:00', 955),
('Electric Spectrum', 1, 31, NULL, 'Warm Up', '2014-07-18 15:07:00', '2014-07-18 16:00:00',
956),
('AZ DC Electric', 1, NULL, 28, 'Warm Up', '2014-07-18 16:20:00', '2014-07-18 17:30:00', 957),
('Infinite Reverie', 2, NULL, 29, 'Headline', '2014-07-18 17:00:00', '2014-07-18 18:53:00',
958),
('Urban Fusion', 2, NULL, 1, 'Support', '2014-07-18 19:04:00', '2014-07-18 19:40:00', 959),
('Acoustic Groove', 2, NULL, 28, 'Warm Up', '2014-07-18 20:07:00', '2014-07-18 21:00:00', 960),
('AZ DC Electric', 2, 60, NULL, 'Warm Up', '2014-07-18 21:30:00', '2014-07-18 22:30:00', 961),
('Cosmic Spectrum', 3, NULL, 34, 'Warm Up', '2014-07-19 12:00:00', '2014-07-19 13:15:00', 962),
('Wild Beat', 3, 49, NULL, 'Headline', '2014-07-19 13:32:00', '2014-07-19 15:05:00', 963),
('Infinite Voyage', 3, NULL, 50, 'Main', '2014-07-19 15:13:00', '2014-07-19 16:00:00', 964),
('AZ DC Electric', 3, 28, NULL, 'Warm Up', '2014-07-19 16:30:00', '2014-07-19 18:30:00', 965)
```

Ενότητα 7: Queries

Question 1:

```
select f.name as festival_name, pm.name as payment_method, f.year,  SUM(tic.cost)
as total_revenue from ticket tic
join event eve on (eve.event_id = tic.event_id)
join payment_method pm on (pm.payment_method_id = tic.payment_method_id)
join festival f on (f.festival_id=eve.festival_id)
group by  f.name, pm.name, f.year
order by f.year
```

Question 2:

```
select distinct a.stage_name,
case
    when g."name" = 'Pop'
    then 'appearred'
    else 'not appearred'
end as appeared
from performance p
join artist a on (a.artist_id = p.artist_id)
join artist_genre ag ON (a.artist_id = ag.artist_id)
join genre g on (g.genre_id= ag.genre_id)
join "event" e on (e.event_id=p.event_id)
join festival f ON (f.festival_id = e.festival_id)
where f.year = 2014
```

Question 3:

```
select f.name as festival_name, f.year, a.stage_name
from performance p
join event     e on p.event_id    = e.event_id
join festival  f on e.festival_id = f.festival_id
join artist    a on p.artist_id   = a.artist_id
where p.performance_type = 'Warm Up'
group by a.stage_name, f.name, f.year
having count(*) > 2
```

Question 4: Έγινε ανάλυση με την χρήση του explain()

Τρόπος 1:

```
select a.artist_id, a.stage_name, round(avg(pr.artist_performance_rating),2),
round(avg(pr.overall_impression_rating),2) from artist a
join performance p ON p.artist_id = a.artist_id
join performance_rating pr on pr.performance_id = p.performance_id
group by a.artist_id, a.stage_name
order by artist_id asc
```

	QUERY PLAN
1	GroupAggregate (cost=17.43..19.23 rows=60 width=79)
2	Group Key: a.artist_id
3	-> Sort (cost=17.43..17.58 rows=60 width=23)
4	Sort Key: a.artist_id
5	-> Hash Join (cost=6.40..15.66 rows=60 width=23)
6	Hash Cond: (p.artist_id = a.artist_id)
7	-> Hash Join (cost=2.35..11.45 rows=60 width=12)
8	Hash Cond: (p.performance_id = pr.performance_id)
9	-> Seq Scan on performance p (cost=0.00..6.77 rows=277 width=8)
10	-> Hash (cost=1.60..1.60 rows=60 width=12)
11	-> Seq Scan on performance_rating pr (cost=0.00..1.60 rows=60 width=12)
12	-> Hash (cost=2.91..2.91 rows=91 width=15)
13	-> Seq Scan on artist a (cost=0.00..2.91 rows=91 width=15)

Τρόπος 2:

```
with avg_performance_rating as (
    select pr.performance_id,  pr.artist_performance_rating ,
pr.overall_impression_rating
    from performance_rating pr
),
artist_performances as (
    select p.performance_id, p.artist_id, a.stage_name from performance p
    join artist a ON a.artist_id = p.artist_id
)
select a.artist_id, a.stage_name, round(avg(apr.artist_performance_rating),2),
round(avg(apr.overall_impression_rating),2) from artist a
join artist_performances ap on ap.artist_id = a.artist_id
```

```
join avg_performance_rating apr on apr.performance_id = ap.performance_id
group by a.artist_id, a.stage_name
order by artist_id
```

QUERY PLAN
GroupAggregate (cost=21.63..23.43 rows=60 width=79)
Group Key: a.artist_id
-> Sort (cost=21.63..21.78 rows=60 width=23)
Sort Key: a.artist_id
-> Hash Join (cost=10.45..19.86 rows=60 width=23)
Hash Cond: (p.artist_id = a_1.artist_id)
-> Hash Join (cost=6.40..15.66 rows=60 width=27)
Hash Cond: (p.artist_id = a.artist_id)
-> Hash Join (cost=2.35..11.45 rows=60 width=12)
Hash Cond: (p.performance_id = pr.performance_id)
-> Seq Scan on performance p (cost=0.00..6.77 rows=277 width=8)
-> Hash (cost=1.60..1.60 rows=60 width=12)
-> Seq Scan on performance_rating pr (cost=0.00..1.60 rows=60 width=
-> Hash (cost=2.91..2.91 rows=91 width=15)
-> Seq Scan on artist a (cost=0.00..2.91 rows=91 width=15)
-> Hash (cost=2.91..2.91 rows=91 width=4)
-> Seq Scan on artist a_1 (cost=0.00..2.91 rows=91 width=4)

Συμπεράσματα:

Βλέπουμε οτι με την χρήση του πρώτου τρόπου, το query πρώτα διαβάζει τις εμφανίσεις, μετά τις βαθμολογίες και μετά τους καλλιτέχνες, και κάνει τα hash joins με μια φορά και συνολικό κόστος περίπου 17 με 19.

Με την χρήση του 2ου τρόπου, η διαδικασία είναι πιο πολύπλοκη και χρονοβόρα καθώς ο optimizer αναγκάζεται να χειριστεί την κατασκευή του κάθε προσωρινού πινάκων ξεχωριστά, και να κάνει 3 ξεχωριστά hash joins στο τέλος για να επιστρέψει τα αποτελέσματα. Το συνολικό κόστος αυξήθηκε στα 21.6 με 23.4.

Question 5:

```
with under_30 as (
    select a.artist_id, a.stage_name, count(distinct e.festival_id) as
number_of_appearences from artist a
    join performance p on (p.artist_id = a.artist_id)
    join event e on (e.event_id = p.event_id)
    where a.date_of_birth >= now() - '30 years'::interval
    group by a.artist_id, a.stage_name
    order by 3 desc
    ),
max_cnt as (
    select MAX(number_of_appearences) AS top_cnt
    from   under_30
    )
select u30.stage_name, mx.top_cnt from under_30 u30
join max_cnt mx on mx.top_cnt = u30.number_of_appearences
```

Question 6 Έγινε ανάλυση με την χρήση του explain():

Τρόπος 1:

```
select p.performance_id, p.name,
  ROUND( (pr.artist_performance_rating + pr.sound_lighting_rating +
pr.stage_presence_rating + pr.organization_rating + pr.overall_impression_rating)
/ 5.0 , 2)
  from performance p
  join performance_rating pr on pr.performance_id = p.performance_id
  where pr.visitor_id = 2
```

QUERY PLAN
Hash Join (cost=1.77..9.32 rows=2 width=54)
Hash Cond: (p.performance_id = pr.performance_id)
-> Seq Scan on performance p (cost=0.00..6.77 rows=277 width=22)
-> Hash (cost=1.75..1.75 rows=2 width=24)
-> Seq Scan on performance_rating pr (cost=0.00..1.75 rows=2 width=24)
Filter: (visitor_id = 2)

Τρόπος 2:

```
WITH
  average_ratings AS (
    SELECT pr.performance_id,
```

```
      ROUND( (pr.artist_performance_rating + pr.sound_lighting_rating +
pr.stage_presence_rating + pr.organization_rating + pr.overall_impression_rating)
/ 5.0 , 2) as average_rating
    FROM performance_rating pr
    WHERE pr.visitor_id = 2
    ORDER BY performance_id
  ),
  performance_sorted AS (
    select performance_id, name FROM performance
    ORDER BY performance_id
  )
select ps.name AS performance_name,
  ar.average_rating
FROM average_ratings ar
JOIN performance_sorted ps ON ar.performance_id = ps.performance_id
```

QUERY PLAN
Merge Join (cost=19.80..23.79 rows=2 width=50)
Merge Cond: (performance.performance_id = pr.performance_id)
-> Sort (cost=18.01..18.70 rows=277 width=22)
Sort Key: performance.performance_id
-> Seq Scan on performance (cost=0.00..6.77 rows=277 width=22)
-> Materialize (cost=1.79..1.82 rows=2 width=36)
-> Sort (cost=1.79..1.80 rows=2 width=36)
Sort Key: pr.performance_id
-> Seq Scan on performance_rating pr (cost=0.00..1.78 rows=2 width=36)
Filter: (visitor_id = 2)

Συμπεράσματα:

Εδώ βλέπουμε ακόμα πιο εύκολα την διαφορά των δύο queries το πρώτο με χρήση Hash Join μεταξύ των πινάκων performance και performance_rating είναι πολύ αποτελεσματικό και γρήγορο με κόστος περίπου 1.77 με 9.32.

Σε αντίθεση το 2ο query κάνει χρήση Merge Join, και πραγματοποιεί sort διαδικασίες πρώτα για κάθε προσωρινό πίνακα, το κόστος είναι σχεδόν τριπλάσιο 19.8 με 23.87.

Question 7:

```
select f.name as festival_name, round(avg(s.experience_level_id),2) as
avg_experience_level
from event e
join event_staff es on e.event_id = es.event_id
join festival f on e.festival_id = f.festival_id
join staff s on s.staff_id = es.staff_id
join staff_category sc on sc.staff_category_id = s.staff_category_id
where sc.name = 'Technical'
--where s.staff_category_id = 1
group by e.festival_id, f.name
order by avg_experience_level
limit 1
```

Question 8:

```
select s.staff_id, s.first_name, s.last_name from staff s
join staff_category sc ON s.staff_category_id = sc.staff_category_id
where sc.name = 'Support'
and s.staff_id not in (
    select es.staff_id
    from event_staff es
    join event e on es.event_id = e.event_id
    where DATE(e.start_timestamp) = '2015-04-24')
order by s.last_name, s.first_name
```

Question 9:

```
with visitors_events_per_year as (
    select f.year, t.visitor_id, count(*) as freq from event e
    join festival f on f.festival_id = e.festival_id
    join ticket t ON t.event_id = e.event_id
    where t.used = true
    group by f.year, t.visitor_id
    having count(*)>3
    order by f.year, t.visitor_id asc
)
select concat(v1.first_name, ' ', v1.last_name) visitor_1, concat(v2.first_name,
' ', v2.last_name) visitor_2, vpy1.year, vpy1.freq
from visitors_events_per_year vpy1
join visitors_events_per_year vpy2 on vpy1.year = vpy2.year and
vpy1.freq=vpy2.freq and vpy1.visitor_id<vpy2.visitor_id
```

```
join visitor v1 on vpy1.visitor_id = v1.visitor_id
join visitor v2 on vpy2.visitor_id = v2.visitor_id
order by v1.visitor_id, year
```

Question 10:

```
 with artist_festivals as (

    select distinct p.artist_id, f.festival_id
    from performance p
    join event e on e.event_id = p.event_id
    join festival f on f.festival_id = e.festival_id
),
artist_genres as (                      -- είδη κάθε καλλιτέχνη
    select ag.artist_id, g.genre_id, g.name
    from artist_genre ag
    join genre g on g.genre_id = ag.genre_id
),
genre_pairs_per_fest as (
    select af.festival_id, g1.name  as genre1,  g2.name  as genre2
    from artist_festivals af
    join artist_genres g1 on g1.artist_id = af.artist_id
    join artist_genres g2 on g2.artist_id = af.artist_id
    where g1.genre_id < g2.genre_id           -- μοναδικοποίηση ζευγών
)
select genre1, genre2, count(distinct festival_id) as fest_cnt
from genre_pairs_per_fest
group by genre1, genre2
order by count(distinct festival_id) desc
limit 3
```

Question 11:

```
with all_artist_participations as (
    select a.artist_id, a.stage_name, count(e.festival_id) as
count_participations
    from performance p
    join artist a on a.artist_id = p.artist_id
    join event e on e.event_id = p.event_id
    group by a.artist_id
    order by 2 desc
)
```

```
select a.artist_id, a.stage_name, arp.count_participations from
all_artist_participations arp
join artist a on a.artist_id = arp.artist_id
where arp.count_participations <= (select max(arp2.count_participations) from
all_artist_participations arp2) - 4
order by 3 desc
```

Question 12:

```
with festival_per_day as(
    select e.festival_id, date(start_timestamp) as running_date, e.event_id from
event e
    where festival_id = 1
    union
    select e.festival_id, date(end_timestamp) as running_date, e.event_id from
event e
    where festival_id = 1
)
select fpd.festival_id, fpd.running_date, sc.name
, count(sc.staff_category_id) as number_of_staff_employeed
from festival_per_day fpd
join event_staff es on es.event_id = fpd.event_id
join staff s on s.staff_id = es.staff_id
join staff_category sc on sc.staff_category_id = s.staff_category_id
group by fpd.festival_id, fpd.running_date, sc.name
order by fpd.running_date asc
```

Question 13:

```
with festival_per_continent as ( --festival per continent
    select f.festival_id, c.name as continent
    from festival f
    join location l on l.location_id = f.location_id
    join continent c on c.continent_id =l.continent_id),
artist_per_festival as (
    --artist per festival
    select a.artist_id, e.festival_id from artist a
    join performance p on p.artist_id = a.artist_id
    join event e on e.event_id = p.event_id
)
select a.artist_id, a.real_name
from festival_per_continent fpc
join artist_per_festival apf on  apf.festival_id = fpc.festival_id
```

```
join artist a on a.artist_id = apf.artist_id
group by a.artist_id
having count(distinct fpc.continent)>= 3
```

Question 14:

```
with perf_genre as (
    select p.performance_id, f.year as festival_year, ag.genre_id
    from performance p
    join event e  on p.event_id = e.event_id
    join festival f  on e.festival_id=f.festival_id
    join artist_genre ag on p.artist_id=ag.artist_id
    union all
    select p.performance_id, f.year as festival_year, bg.genre_id
    from performance p
    join event e  on p.event_id = e.event_id
    join festival f  on e.festival_id = f.festival_id
    join band_genre bg on p.band_id = bg.band_id
),
genre_year_counts as (
    select genre_id, festival_year, count(*) as appearances
    from perf_genre
    group by genre_id, festival_year
)
select g.name as genre_name, g1.festival_year  as year1, g2.festival_year  as
year2, g1.appearances as appearances_per_year
from genre_year_counts g1
join genre_year_counts g2 on g1.genre_id= g2.genre_id
join genre g on g.genre_id=g1.genre_id
where g1.appearances >= 3
and g2.appearances= g1.appearances
and g2.festival_year= g1.festival_year + 1
order by g.name, g1.festival_year
```

Question 15:

```
with visitor_artist_scores as (
  select pr.visitor_id, p.artist_id,
    sum(coalesce(pr.artist_performance_rating,
0)+coalesce(pr.sound_lighting_rating, 0) + coalesce(pr.stage_presence_rating,0)+
coalesce(pr.organization_rating,0) + coalesce(pr.overall_impression_rating, 0))
as total_score
  from performance_rating pr
```

```
  join performance p on pr.performance_id = p.performance_id
  where p.artist_id is not null
  group by pr.visitor_id, p.artist_id
)
select
  concat(v.first_name,' ', v.last_name) as visitor_name,
  a.stage_name as stage_name,
  vas.total_score
from visitor_artist_scores vas
join visitor v on vas.visitor_id = v.visitor_id
join artist  a on vas.artist_id  = a.artist_id
order by vas.total_score desc
limit 5
```

Ενότητα 9: Installation Guide

Για την εγκατάσταση ακολουθήστε τα παρακάτω βήματα:

1. Εγκατάσταση Postgres. Ακολουθήστε τον σύνδεσμο παρακάτω για να εγκαταστήσετε την Postgres: https://www.postgresql.org/
2. Εγκατάσταση DBMS. Εγκαταστήστε κάποιο Data Management System, όπως πχ: https://dbeaver.io/
3. Χρησιμοποιώντας το psql terminal, δημιουργήστε μια νέα βάση με την εντολή:
   CREATE DATABASE festival_db;

   ```
   postgres=# CREATE DATABASE festival_db;
   CREATE DATABASE
   postgres=#
   ```
4. Συνδεθείτε στην βάση με την εντολή **\c music_festival**
5. Για να δημιουργήσετε το schema, τρέξτε το αρχείο **install.sql** βρίσκετε στον φάκελο sql είτε μέσα από το DBMS, είτε μέσα από τον psql terminal με την εντολή:
   \i path_to_file\install.sql
   Example: \i C:/Users/[Username]/Documents/install.sql
6. Για να προσθέσετε δεδομένα στην βάση, τρέξτε το αρχείο **load.sql** βρίσκετε στον φάκελο sql είτε μέσα από το DBMS, είτε μέσα από τον psql terminal με την εντολή:
   \i path_to_file\load.sql
   Example: \i C:/Users/[Username]/Documents/load.sql

```
festival_db=# \i C:/Users/stavros_work/Documents/load.sql
SET
SET
SET
SET
INSERT 0 1413
INSERT 0 7
INSERT 0 30
INSERT 0 15
INSERT 0 37
INSERT 0 10
INSERT 0 150
```

7. Αφού φορτώσετε τα δεδομένα, μπορείτε να τρέξετε τα queries που βρίσκονται στον φάκελο sql σημειωμένα ως **Qx.sql** ανοίγοντας τα αρχεία στο DBMS ή στο psql terminal ανοίγοντάς τα αρχεία όπως περιγράψαμε παραπάνω