ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

«Υλοποίηση βάσης δεδομένων για το διεθνές φεστιβάλ μουσικής, Pulse University»

Εξαμηνιαία Εργασία

στο μάθημα **«Βάσεις δεδομένων»**

των φοιτητών

Νικόλαου Σμυρνάκη, Α.Μ.: 03122428

Μαρίνας Φραγκούλη, Α.Μ.: 03122429

Διδάσκοντες: Δ. Τσουμάκος, Μ. Κόνιαρης

Αθήνα, Μάιος 2025

Πίνακας περιεχομένων

1 *Διευκρινίσεις*

Εισάγει τον αναγνώστη στο θέμα και το αντικείμενο της εργασίας. Περιλαμβάνει ορισμούς και ερμηνείες όρων, παραδοχές, συναφή θεωρητικά και ερευνητικά δεδομένα όπως παρουσιάζονται στη βασική βιβλιογραφία του υπό μελέτη θέματος. Τα παραπάνω στοιχεία τεκμηριώνονται από βιβλιογραφικές αναφορές που εμφανίζονται στο σώμα του κειμένου. Γίνεται αναφορά στη σημασία της εργασίας. Από τα παραπάνω στοιχεία προκύπτουν τα ερευνητικά ερωτήματα που καλείται να απαντήσει η εργασία, τα οποία διατυπώνονται στο τέλος αυτής της πρώτης ενότητας. Μια συνηθισμένη πρακτική για την εισαγωγή είναι να απαντώνται τα παρακάτω ερωτήματα:

• Ποιο είναι το επιστημονικό πεδίο με το οποίο ασχολείται η εργασία;

• Ποιο είναι το θέμα της εργασίας;

• Ποιο είναι το ενδιαφέρον του θέματος από επιστημονικής πλευράς;

• Ποιο είναι το ενδιαφέρον του θέματος από πρακτικής πλευράς (ποιους ενδιαφέρει, πρακτικές εφαρμογές ή χρήσεις);

• Πως γίνεται η ανάλυση του θέματος;

Το μέγεθος της εισαγωγής συνήθως δεν είναι μεγαλύτερο από το ένα τέταρτο της συνολικής έκτασης της εργασίας.

1.1 Διαγράμματα

Σχεδιασμός και Οπτικοποίηση της Βάσης Δεδομένων

- Το **ER διάγραμμα** δημιουργήθηκε με χρήση του εργαλείου **draw.io**.

- Χρησιμοποιήθηκε η λειτουργία **Export Schema** από τον **PHPMyAdmin Designer** για την εξαγωγή του αρχικού relational diagram.

- Για καλύτερη αισθητική και καθαρή παρουσίαση, δημιουργήθηκαν δύο εκδόσεις:

    - Μία **relational έκδοση** με βάση την εξαγωγή του PHPMyAdmin.

    - Μία **βελτιωμένη (prettier)** έκδοση στο **DB diagram.io**, με καλύτερη στοίχιση και οργάνωση των οντοτήτων.

Σημαντική παρατήρηση: Η τελευταία εφαρμογή (DB diagram.io) απαιτεί τροποποίηση του SQL κώδικα, καθώς δεν υποστηρίζει πλήρως τη σύνταξη της MySQL. Πραγματοποιήθηκαν αλλαγές για συμβατότητα.

1.2 Κώδικας

1.2.1 Δομή

Το αρχείο festival_marina.sql έχει την εξής δομή:

- Tables
- Indexes
- View
- Triggers
- Constrains
- Cascades

· Events

Τα παραπάνω αναλύονται παρακάτω.

Ως Event έχουμε μία συνάρτηση την delete_matched_resale_entries η οποία καλείται κάθε μία ώρα και έχει την παρακάτω λογική -- Delete the matched resale entry from the resale_queue after some time

1.2.2 Μεθοδολογία ανάπτυξης

Τα βήματα που ακολουθήθηκαν στην εργασία ήταν τα εξής

1. ER Diagram development
2. Table development (sql) in festival_marina.sql
3. Python dummy data creator (python) data_creator.py
4. Triggers incorporated (sql) in festival_marina.sql
5. Update python accordingly (python) data_creator.py
6. Backup creation backup
7. Queries (sql)
8. Views
9. Indexes to facilitate queries
10. Cascades
11. Export ER Diagram
12. Export relational diagram
13. Renaming according to the instructions
14. Authentication
15. Readme και Αναφορά

1.2.3 Περιβάλλον ανάπτυξης

- **DBMS (Data Base Management System):** Χρησιμοποιήσαμε MySQL / MariaDB.

- **Περιβάλλον Ανάπτυξης & Web Stack:** Εργαστήκαμε με το XAMPP, το οποίο περιλαμβάνει Apache, MySQL / MariaDB και PHP.

- **Συνεργατική ανάπτυξη:** Για την απομακρυσμένη συνεργασία και τον συγχρονισμό του κώδικα χρησιμοποιήσαμε το GitHub.

- Η ανάπτυξη της βάσης πραγματοποιήθηκε στο Visual Studio Code.

- Τα dummy δεδομένα δημιουργήθηκαν αυτόματα με το script data_creator2304.py, αξιοποιώντας τη βιβλιοθήκη faker.

1.3 Παραδοχές

Παραδοχές βάσει ER Διαγράμματος και Triggers

1. Εισιτήρια & Επισκέπτες

- Κάθε επισκέπτης μπορεί να αγοράσει μόνο ένα εισιτήριο για μία συγκεκριμένη παράσταση και ημέρα του φεστιβάλ.

- Ένας επισκέπτης μπορεί να έχει συνολικά πολλά εισιτήρια, αρκεί κάθε εισιτήριο να αφορά διαφορετική παράσταση ή/και ημέρα.

2. Κανόνες Μεταπώλησης (Resale)

- Η μεταπώληση εισιτηρίων ενεργοποιείται όχι όταν εξαντληθούν όλα τα εισιτήρια ενός event, αλλά όταν εξαντληθεί ένας συγκεκριμένος τύπος εισιτηρίου (π.χ. μόνο τα VIP).

- Η εισαγωγή στην ουρά μεταπώλησης (resale_queue) επιτρέπεται μόνο όταν ισχύει μία από τις παρακάτω περιπτώσεις:

Πωλητής (Seller):

  - ticket_ID IS NOT NULL
  - event_name IS NOT NULL
  - ticket_type IS NOT NULL

- Αγοραστής (Buyer):
  - Περίπτωση 1: Θέλει εισιτήριο τύπου χωρίς να ξέρει ποιο ακριβώς:
    - ticket_ID IS NULL

        - event_name IS NOT NULL

        - ticket_type IS NOT NULL

    - Περίπτωση 2: Θέλει να αγοράσει συγκεκριμένο εισιτήριο:

        - ticket_ID IS NOT NULL

        - event_name IS NULL

        - ticket_type IS NULL

- Η μεταπώληση επιτρέπεται μόνο για μη ενεργοποιημένα εισιτήρια.
- Η μεταπώληση επιτρέπεται μόνο πριν από το DATETIME έναρξη του event.
- Οι αγοραστές μπορούν να εκδηλώσουν ενδιαφέρον είτε για συγκεκριμένο ticket_ID, είτε για συνδυασμό event_name και ticket_type.

- Η αντιστοίχιση αγοραστών και πωλητών γίνεται αυτόματα μέσω trigger (match_resale_after_insert), και το αποτέλεσμα καταχωρείται στον πίνακα temp_resale_matches. Δηλαδή στον τελευταίο πίνακα

- Η ουρά λειτουργεί με σειρά προτεραιότητας FIFO.

3. Χρηματικές Συναλλαγές

- Οι χρεώσεις και πιστώσεις δεν υλοποιούνται σε επίπεδο βάσης δεδομένων (δεν υπάρχουν σχετικά πεδία ή πίνακες).

- Η λειτουργία αυτή θεωρείται ότι καλύπτεται εξωτερικά, σε επίπεδο εφαρμογής (backend), και δεν υλοποιείται με triggers ή SQL logic.

4. Αξιολόγηση Εμφανίσεων (Reviews)

- Μόνο κάτοχοι ενεργοποιημένων εισιτηρίων μπορούν να υποβάλουν αξιολόγηση, σύμφωνα με trigger (check_ticket_activation).

- Επιτρέπεται η αξιολόγηση μόνο αν το performance ανήκει στο ίδιο event με το εισιτήριο (check_review_validity).

- Η αξιολόγηση γίνεται με βάση πέντε κριτήρια (ερμηνεία, ήχος και φωτισμός, σκηνική παρουσία, οργάνωση, συνολική εντύπωση), με χρήση κλίμακας Likert (ENUM: '1', '2', '3', '4', '5') — όχι με αριθμητικούς τύπους δεδομένων.

5. Εξοπλισμός Κτιρίων

- Το πεδίο technical_equipment του πίνακα building περιγράφει τον αναμενόμενο εξοπλισμό για τις εκδηλώσεις, όχι τον πραγματικά διαθέσιμο — αποτελεί στατική απαίτηση του εκάστοτε χώρου.

6. Εικόνες και Περιγραφές

- Στη βάση δεδομένων δεν υπάρχουν πεδία για εικόνες ή λεκτικές περιγραφές για οντότητες όπως festival, artist, building, equipment.

- Η δυνατότητα προσθήκης εικόνων και πολυμέσων αποτελεί παραδοχή σχεδιαστικής επέκτασης, η οποία μπορεί να καλυφθεί με την προσθήκη πεδίων όπως image_url, image_description.

Παρατήρηση για την Python Η Python τα εισιτήρια μας τα δίνει ήδη κάποια ενεργοποιημένα και κάποια όχι ενεργοποιημένα.

2 *DDL script(install.sql)*

Σε αυτή την εργασία η SQL χρησιμοποιήθηκε ως DDL(Data Definition Language).

festival

Ο πίνακας festival καταγράφει τα βασικά στοιχεία κάθε φεστιβάλ, με μοναδικό αναγνωριστικό (festival_ID), ημερομηνία έναρξης (starting_date) και διάρκεια σε ημέρες (duration).

festival_location

Ο πίνακας festival_location συνδέει κάθε φεστιβάλ με τις διάφορες τοποθεσίες του, αποθηκεύοντας διεύθυνση, πόλη, χώρα, ήπειρο και γεωγραφικές συντεταγμένες, ενώ αναφέρεται στο αντίστοιχο φεστιβάλ μέσω ξένου κλειδιού.

personel

Στον πίνακα personel αποθηκεύονται τα προσωπικά και επαγγελματικά στοιχεία των υπαλλήλων ή εθελοντών (π.χ. όνομα, επώνυμο, ηλικία, email, τηλέφωνο) και η βαθμίδα εμπειρίας τους (expertise_status).

building

Ο πίνακας building περιγράφει τα κτήρια όπου διοργανώνονται εκδηλώσεις, με πεδία για όνομα, λεπτομερή περιγραφή και μέγιστη χωρητικότητα.

technical_equipment

Στον πίνακα technical_equipment φυλάσσονται πληροφορίες για τον τεχνικό εξοπλισμό που αντιστοιχεί σε κάθε κτήριο (όνομα και περιγραφή), συνδεδεμένες με το κτήριο μέσω ξένου κλειδιού.

artist

Ο πίνακας artist καταγράφει ατομικούς καλλιτέχνες, αποθηκεύοντας πραγματικό και σκηνικό όνομα, ημερομηνίες γέννησης και ντεμπούτου, προαιρετικά links (website, Instagram) και πλήθος διαδοχικών ετών συμμετοχής.

group

Στον πίνακα group καταχωρείται το αναλυτικό προφίλ μουσικών ή άλλων ομάδων, με στοιχεία όπως όνομα, ημερομηνία ίδρυσης, ντεμπούτου, social media, λίστα με ονόματα μελών και διαδοχικά έτη συμμετοχής.

genre

Ο πίνακας genre ταξινομεί καλλιτέχνες ή ομάδες με βάση κύριο είδος και υποείδος, συνδέοντας κάθε εγγραφή είτε με έναν artist είτε με έναν group μέσω ξένων κλειδιών.

group_members

Η group_members είναι ένας συνδετικός πίνακας πολλών-προς-πολλά που συσχετίζει ομάδες με καλλιτέχνες, χρησιμοποιώντας ως σύνθετο πρωτεύον κλειδί τα group_ID και artist_ID.

events

Στον πίνακα events αποτυπώνεται κάθε επιμέρους εκδήλωση ενός φεστιβάλ, με πληροφορίες όπως όνομα, ημέρα στο πλαίσιο του φεστιβάλ, ώρες έναρξης/λήξης, κτήριο διεξαγωγής, υπολογιζόμενη διάρκεια και διαθέσιμα εισιτήρια (VIP, backstage, general).

performances

Ο πίνακας performances καταγράφει κάθε εμφάνιση (warm up, headline, special guest, finale) που ανήκει σε ένα event, με ώρες, υπολογιζόμενη διάρκεια, κτήριο, και — εναλλακτικά — συνδεδεμένο artist ή group, ελέγχοντας και το ανώτατο όριο διάρκειας.

role_of_personel_on_event

Η role_of_personel_on_event ορίζει τους ρόλους (τεχνικός, ασφάλεια, υποστήριξη) που κάθε μέλος του προσωπικού έχει σε κάθε εκδήλωση, σε μια σχέση πολλών-προς-πολλά με σύνθετο πρωτεύον κλειδί.

visitor

Στον πίνακα visitor καταχωρούνται τα προσωπικά στοιχεία των επισκεπτών (όνομα, επώνυμο, τηλέφωνο, email, ηλικία), τα οποία χρησιμοποιούνται για την αγορά και ενεργοποίηση εισιτηρίων.

ticket

Ο πίνακας ticket συνδέει κάθε εισιτήριο με έναν επισκέπτη και ένα event, αποθηκεύοντας τύπο εισιτηρίου, ημερομηνία/τιμή αγοράς, μέθοδο πληρωμής, κατάσταση ενεργοποίησης και αποσπασματικά στοιχεία του visitor για ιστορικούς λόγους.

buyer

Η buyer είναι βοηθητικός πίνακας που δηλώνει ποιοι επισκέπτες ενδιαφέρονται να αγοράσουν εισιτήρια, επαναχρησιμοποιώντας το ίδιο αναγνωριστικό με τον αντίστοιχο visitor.

seller

Αντίστοιχα, η seller καταχωρεί όσους επισκέπτες προσφέρουν εισιτήρια προς μεταπώληση, συνδέοντας το seller_ID με τον visitor.

resale_queue

Ο πίνακας resale_queue λειτουργεί ως FIFO λίστα για εισητήρια σε μεταπώληση, κρατώντας buyer, seller, ticket, τύπο εισιτηρίου, event και χρονοσήμανση καταχώρισης.

review

Στην review αποθηκεύονται οι αξιολογήσεις των επισκεπτών για κάθε performance (καλλιτεχνική εμφάνιση, ήχος/φωτισμός, παρουσία στη σκηνή, οργάνωση, συνολική εντύπωση) αφού έχει ενεργοποιηθεί το εισιτήριό τους.

temp_resale_matches

Η temp_resale_matches είναι προσωρινός πίνακας για να αποθηκεύονται «ζευγάρια» buyer–seller–ticket κατά την αναζήτηση πιθανών αντιστοιχιών για μεταπώληση.

photo

Τέλος, ο πίνακας photo αποθηκεύει μεταδεδομένα φωτογραφιών ή άλλων μέσων (όνομα, περιγραφή) που μπορεί να συνδέονται με καλλιτέχνες, ομάδες, performances, events, φεστιβάλ ή τεχνικό εξοπλισμό, παρέχοντας ευέλικτη συσχέτιση με πολλαπλές οντότητες.

Εκτός από τα Tables έχουμε φτιάξει και ένα view το artist_participations όπου συγκεντρώνουμε σε έναν ενιαίο πίνακα όλες τις συμμετοχές των καλλιτεχνών –είτε ως solo είτε μέσα από γκρουπ– σε παραστάσεις φεστιβάλ.

```sql
-- == VIEW == --
CREATE VIEW artist_participations AS
SELECT
    a.artist_ID,
    a.artist_date_of_birth,
    a.artist_name,
    f.festival_ID,
    e.event_ID,
    p.performance_ID,
    p.performance_type,
    p.performance_start_time,
    p.performance_end_time,
    p.performance_duration,
    'solo' AS participation_type
FROM
    artist a
    JOIN performances p ON a.artist_ID = p.artist_ID
    JOIN events e ON p.event_ID = e.event_ID
    JOIN festival f ON e.festival_ID = f.festival_ID

UNION ALL

SELECT
    gm.artist_ID,
    a.artist_date_of_birth,
    a.artist_name,
    f.festival_ID,
    e.event_ID,
    p.performance_ID,
    p.performance_type,
    p.performance_start_time,
    p.performance_end_time,
    p.performance_duration,
    'group' AS participation_type
FROM
    group_members gm
    JOIN `group` g ON gm.group_ID = g.group_ID
    JOIN performances p ON g.group_ID = p.group_ID
    JOIN events e ON p.event_ID = e.event_ID
    JOIN festival f ON e.festival_ID = f.festival_ID
    JOIN artist a ON gm.artist_ID = a.artist_ID;
```

- Στην αρχή βρίσκουμε του solo artists, στην συνέχεια τα groups και αυτά τα ενώνουμε με UNION ALL. Ενώνουμε δηλαδή τους πίνακες artist → performances → events → festival και προσθέτει σταθερά τη στήλη participation_type = 'solo’.
- Για τα group ενώνει τους πίνακες group_members → group → performances → events → festival κι έπειτα ξανασυνδέει στο artist ώστε να εμφανιστούν τα στοιχεία των ατομικών μελών. Η στήλη participation_type = 'group’ διακρίνει αυτές τις εγγραφές.

Κάθε γραμμή του view περιλαμβάνει:

1. artist_ID, artist_name, artist_date_of_birth
2. festival_ID, event_ID, performance_ID
3. performance_type, performance_start_time, performance_end_time, performance_duration
4. participation_type (solo ή group)

Με τον τρόπο αυτό πετυχαίνουμε πιο απλά και σαφή queries. Δημιουργούμε με τον τρόπο αυτό ένα μόνο query σε όλους τους καλλιτέχνες ανεξαρτήτως τρόπου συμμετοχής. Επιπλέον αντί πολλαπλών JOINS μέσα στις αναφορές, αρκεί ένα SELECT * FROM artist_participations ενώ η στήλη participation_type επιτρέπει εύκολη φιλτράρισμα ή ανάλυση solo vs group.

Στην ενότητα αυτή παρουσιάζεται το θεωρητικό υπόβαθρο της μελέτης. Γίνεται αναλυτική περιγραφή του τρόπου διεξαγωγής των πειραμάτων και της διαδικασίας συλλογής των δεδομένων, θεμελιώνονται οι αλγόριθμοι επεξεργασίας των δεδομένων και οι μέθοδοι αξιολόγησης των αποτελεσμάτων.

3 *Σχεδιασμός και υλοποίηση*

3.1 E-R διάγραμμα

Με υπογράμμιση φαίνονται τα Primary Keys και με (FK) τα Foreign Keys (σύμφωνα με το βιβλίο τα FK παραλείπονται λόγω της ύπαρξης των ρόμβων όμως με αυτόν τον τρόπο ήταν πιο εύκολη η μετάβαση από το ER στην SQL). Σύμφωνα με την εκφώνηση κάθε μια από τις υπογραμμισμένες λέξεις είναι ένας πίνακας. Οι πίνακες της εκφώνησης είναι οι μπλε πίνακες ενώ πορτοκαλί είναι οι πίνακες που προσθέσαμε εμείς κατά παραδοχή. Το ER μας έχει ως weak entities τα role_of_personel_on _event, τα reviews, τα group_members, τους buyers και τους sellers.

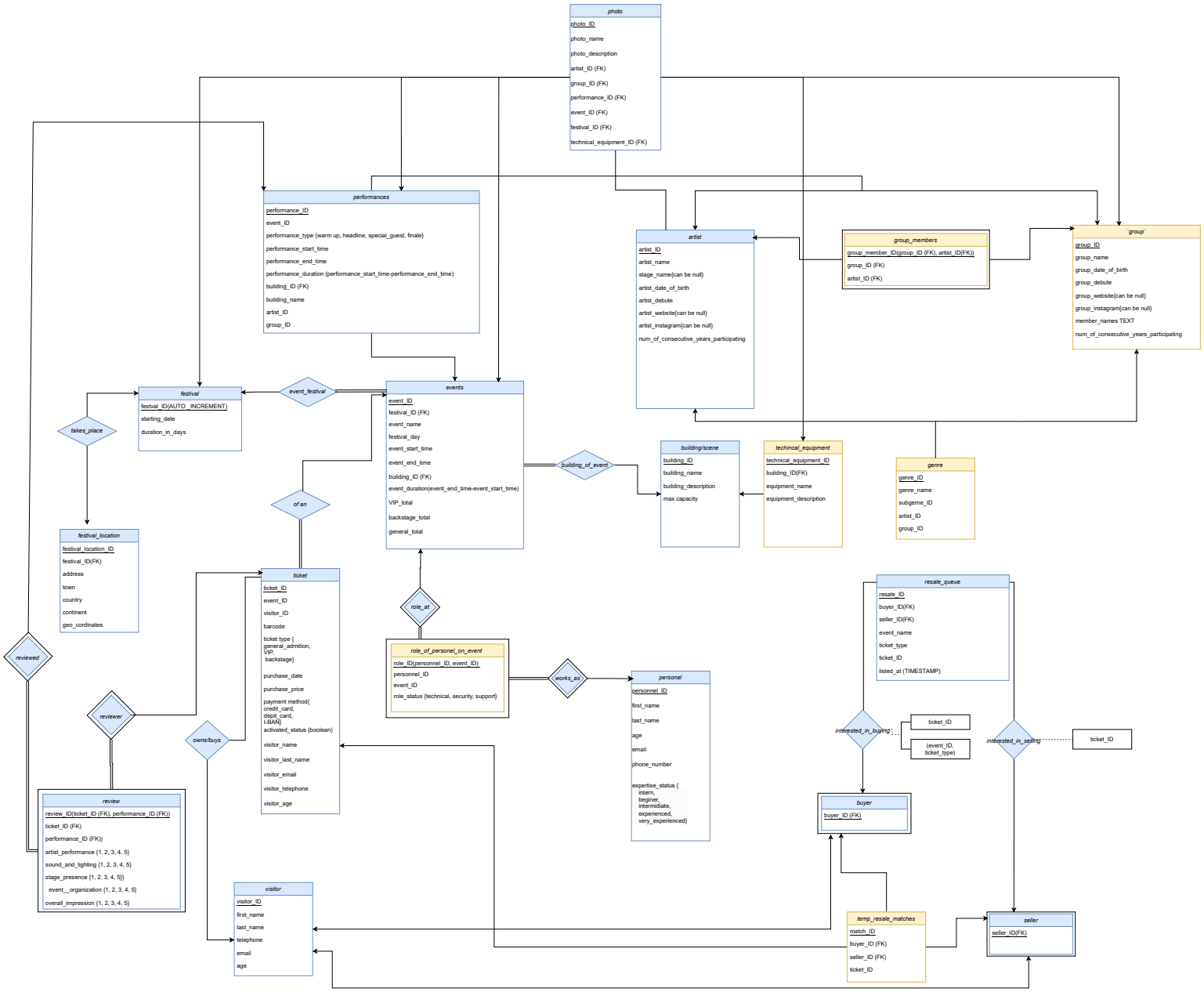
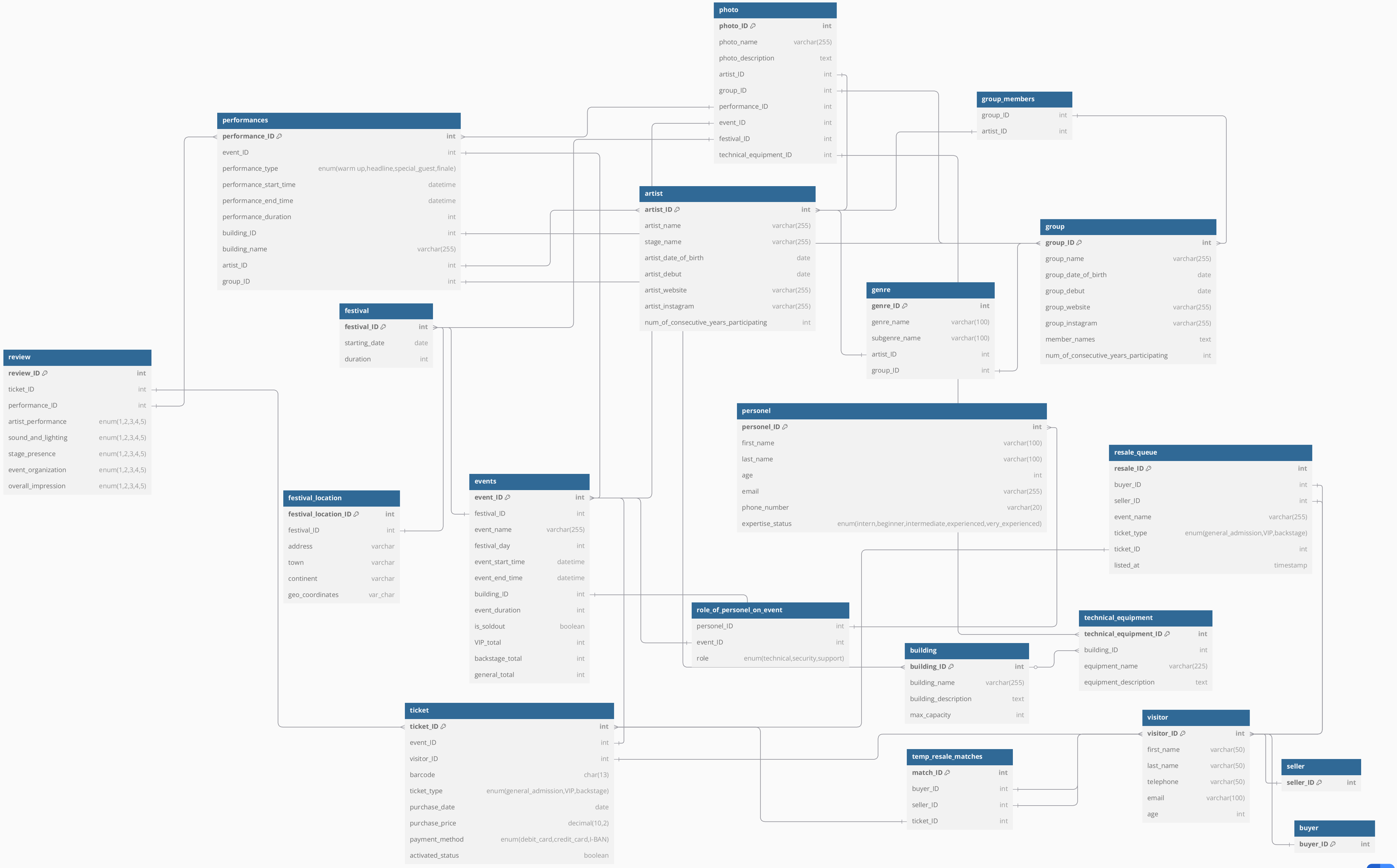
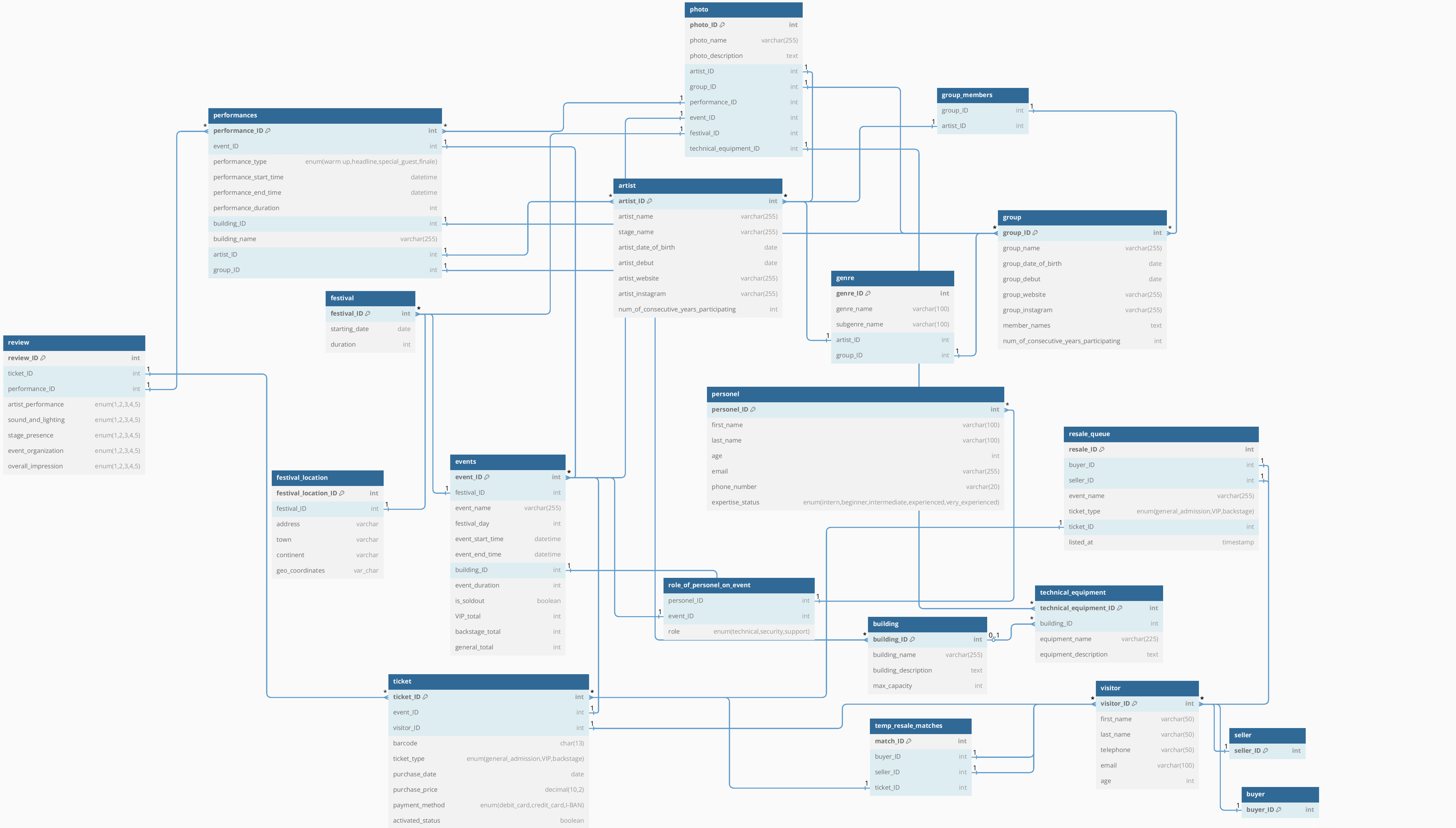


photo
photo_ID
photo_name
photo_description
artist_ID (FK)
group_ID (FK)
performance_ID (FK)
event_ID (FK)
festival_ID (FK)
technical_equipment_ID (FK)
performances
performance_ID
event_ID
performance_type (warm up, headline, special_guest, finale)
performance_start_time
performance_end_time
performance_duration (performance_start_time-performance_end_time)
building_ID (FK)
building_name
artist_ID
group_ID
artist
artist_ID
artist_name
stage_name(can be null)
artist_date_of_birth
artist_debute
artist_website(can be null)
artist_instagram(can be null)
num_of_consecutive_years_participating
group_members
group_member_ID(group_ID (FK), artist_ID(FK))
group_ID (FK)
artist_ID (FK)
'group'
group_ID
group_name
group_date_of_birth
group_debute
group_website(can be null)
group_instagram(can be null)
member_names TEXT
num_of_consecutive_years_participating
festival
festival_ID(AUTO_INCREMENT)
starting_date
duration_in_days
event_festival
events
event_ID
festival_ID (FK)
event_name
festival_day
event_start_time
event_end_time
building_ID (FK)
event_duration(event_end_time-event_start_time)
VIP_total
backstage_total
general_total
takes_place
building_of_event
building/scene
building_ID
building_name
building_description
max capacity
techincal_equipment
technical_equipment_ID
building_ID(FK)
equipment_name
equipment_description
genre
genre_ID
genre_name
subgerne_ID
artist_ID
group_ID
of an
festival_location
festival_location_ID
festival_ID(FK)
address
town
country
continent
geo_cordinates
ticket
ticket_ID
event_ID
visitor_ID
barcode
ticket type (general_admition, VIP, backstage)
purchase_date
purchase_price
payment method(credit_card, depit_card, I-BAN)
activated_status (boolean)
visitor_name
visitor_last_name
visitor_email
visitor_telephone
visitor_age
role_at
role_of_personel_on_event
role_ID(personnel_ID, event_ID),
personnel_ID
event_ID
role_status (technical, security, support)
works_as
personel
personnel_ID
first_name
last_name
age
email
phone_number
expertise_status (intern, beginer, intermidiate, experienced, very_experienced)
resale_queue
resale_ID
buyer_ID(FK)
seller_ID(FK)
event_name
ticket_type
ticket_ID
listed_at (TIMESTAMP)
interested_in_buying
ticket_ID
(event_ID, ticket_type)
interested_in_selling
ticket_ID
buyer
buyer_ID (FK)
seller
seller_ID(FK)
temp_resale_matches
match_ID
buyer_ID (FK)
seller_ID (FK)
ticket_ID
reviewed
reviewer
owns/buys
review
review_ID(ticket_ID (FK), performance_ID (FK))
ticket_ID (FK)
performance_ID (FK))
artist_performance {1, 2, 3, 4, 5)
sound_and_lighting {1, 2, 3, 4, 5)
stage_presence {1, 2, 3, 4, 5))
event__organization (1, 2, 3, 4, 5)
overall_impression {1, 2, 3, 4, 5)
visitor
visitor_ID
first_name
last_name
telephone
email
age

3.2 Σχεσιακό διάγραμμα

3.2.1 Το διάγραμμα





3.2.2 Ορισμός περιορισμών

Εκφώνηση

Να ορίσετε όλους τους απαραίτητους περιορισμούς που θα εξασφαλίζουν την ορθότητα της ΒΔ. Αυτοί είναι περιορισμοί ακεραιότητας, κλειδιά, αναφορική ακεραιότητα, ακεραιότητα πεδίου τιμών και περιορισμοί οριζόμενοι από τον χρήστη.

Απάντηση

Γενικότερα για να εξασφαλιστεί η ορθότητα της βάσης δεδομένων (ΒΔ), πρέπει να οριστούν οι παρακάτω περιορισμοί (constraints):

1. Περιορισμοί ακεραιότητας (PK, FK, UNIQUE, NOT NULL)
2. Αναφορική ακεραιότητα(Referential Integrity) (FK, ON DELETE CASCADE / ON UPDATE CASCADE)
3. Ακεραιότητα Πεδίου Τιμών (Domain Integrity) (CHECK, ENUM)
4. Περιορισμοί από τον Χρήστη.

Εμείς χρησιμοποιήσαμε Triggers, Constrains και Cascades.

TRIGGERS

Deletion Triggers

- prevent_festival_deletion -- Prevent Festival Deletion Trigger (2)
- prevent_performance_deletion -- Prevent Performance Deletion Trigger (2)

Ticket Triggers

- check_ticket_availability -- Check if the ticket can be sold based on the event's capacity and ticket type limits (3)
- fill_ticket_visitor_data -- When a new ticket is created, fill in visitor data from the visitor table

Visitor Triggers

- trg_ticket_update_visitor -- Ensure visitor's info is updated in the ticket table when visitor info is updated (instead of a cascade)

Resale Triggers

- check_ticket_activation_before_resale -- Trigger to check that the ticket is not activated before resale (3)
- trg_check_soldout_before_resale -- Check if the ticket is sold out before allowing resale (4)
- create_buyer_or_seller_after_visitor -- When a new resale entry of a buyer is created, add the buyer to the buyer table. When a new resale entry of a seller is created, add the seller to the seller table (4)
- match_resale_after_insert -- Matching Seller and Buyer and Updating Ticket Info

Event Triggers

- check_festival_day -- Ensure that the festival_day is within the festival duration (4)

Genre Triggers

- check_genre_entity_exclusivity -- Ensure that each genre is linked to either one artist or one group, but not both or neither (3)

Performance Triggers

- check_performance_overlap -- Ensure a minimum 5-minute break between performances of the same event in the same building
- trg_check_consecutive_years_artists -- Check for consecutive years of participation for artists (4)
- trg_check_consecutive_years_groups -- Check for consecutive years of participation for groups (4)

- prevent_artist_group_overlap (on Insert) -- Check for overlapping performances for the same artist or group (4)

- prevent_artist_group_overlap_update (on Update) -- Check for overlapping performances for the same artist or group on update (4)

Ticket Triggers

- check_vip_limit VIP ticket limit check -- Ensure that the number of VIP tickets does not exceed 10% of total tickets for the event. This is done using a trigger before inserting a new ticket (4)

- prevent_duplicate_ticket -- Prevent duplicate tickets for the same visitor and event

Group Triggers

- group_member_names -- When a new group member is added, update the member_names field in the group table

Review Triggers

- check_ticket_activation -- Ensure that a review can only be created if the ticket is activated

- check_review_validity -- Ensure that the performance belongs to the same event as the ticket and ticket is activated

Role of personnel on event Triggers

- check_personel_availability -- Ensure that the same personel cannot have multiple roles in the same event

CONSTRAINS

Resale Constrains

chk_seller_or_buyer -- Ensure that either ticket_ID is NULL or event_name and ticket_type are NULL (4)

chk_one_side_only -- Ensure that either buyer_ID or seller_ID is NULL, but not both (4)

CASCADES

On deletion, on update on personel delete (and update) of role_of_personel_on event as well

On deletion, on update on ticket delete (and update) of review as well

On deletion, on update on group delete (and update) of group_members as well

Για να δημιουργήσουμε τα cascades αυτά έπρεπε πρώτα να δούμε πως έχει ονομάσει η sql τα FK στα οποία θα αναφερόμασταν. Έτσι:

Για το attribute role_of_personel_on_event.personel_ID θα αναφερθούμε σε αυτό ως role_of_personel_on_event. role_of_personel_on_event_ibfk_1 ενώ για το attribute role_of_personel_on_event.event_ID θα αναφερθούμε σε αυτό ως role_of_personel_on_event. role_of_personel_on_event_ibfk_2

```
MariaDB [db1]> SHOW CREATE TABLE role_of_personel_on_event\G
*************************** 1. row ***************************
       Table: role_of_personel_on_event
Create Table: CREATE TABLE `role_of_personel_on_event` (
  `personel_ID` int(11) NOT NULL,
  `event_ID` int(11) NOT NULL,
  `role` enum('technical','security','support') NOT NULL,
  PRIMARY KEY (`personel_ID`,`event_ID`),
  KEY `idx_role_event_role` (`event_ID`,`role`),
  KEY `idx_role_event_role_personel` (`role`,`personel_ID`,`event_ID`),
  CONSTRAINT `role_of_personel_on_event_ibfk_1` FOREIGN KEY (`personel_ID`) REFERENCES `personel` (`personel_ID`),
  CONSTRAINT `role_of_personel_on_event_ibfk_2` FOREIGN KEY (`event_ID`) REFERENCES `events` (`event_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
1 row in set (0.004 sec)
```

Για το attribute review.ticket_ID θα αναφερθούμε σε αυτό ως review.review_ibfk_1 ενώ για το attribute review.performace_ID θα αναφερθούμε σε αυτό ως review.review_ibfk_2

```
MariaDB [db1]> SHOW CREATE TABLE review\G
*************************** 1. row ***************************
       Table: review
Create Table: CREATE TABLE `review` (
  `ticket_ID` int(11) NOT NULL,
  `performance_ID` int(11) NOT NULL,
  `artist_performance` enum('1','2','3','4','5') DEFAULT NULL,
  `sound_and_lighting` enum('1','2','3','4','5') DEFAULT NULL,
  `stage_presence` enum('1','2','3','4','5') DEFAULT NULL,
  `event_organization` enum('1','2','3','4','5') DEFAULT NULL,
  `overall_impression` enum('1','2','3','4','5') DEFAULT NULL,
  PRIMARY KEY (`ticket_ID`,`performance_ID`),
  KEY `performance_ID` (`performance_ID`),
  KEY `idx_review_ticket` (`ticket_ID`),
  CONSTRAINT `review_ibfk_1` FOREIGN KEY (`ticket_ID`) REFERENCES `ticket` (`ticket_ID`),
  CONSTRAINT `review_ibfk_2` FOREIGN KEY (`performance_ID`) REFERENCES `performances` (`performance_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
1 row in set (0.001 sec)
```

Για το attribute group_members.group_ID θα αναφερθούμε σε αυτό ως group_members.group_members_ibfk_1 ενώ για το attribute group_members.artist_ID θα αναφερθούμε σε αυτό ως group_members.group_members_ibfk_2

```
MariaDB [db1]> SHOW CREATE TABLE group_members\G
*************************** 1. row ***************************
       Table: group_members
Create Table: CREATE TABLE `group_members` (
  `group_ID` int(11) NOT NULL,
  `artist_ID` int(11) NOT NULL,
  PRIMARY KEY (`group_ID`,`artist_ID`),
  KEY `artist_ID` (`artist_ID`),
  CONSTRAINT `group_members_ibfk_1` FOREIGN KEY (`group_ID`) REFERENCES `group` (`group_ID`),
  CONSTRAINT `group_members_ibfk_2` FOREIGN KEY (`artist_ID`) REFERENCES `artist` (`artist_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
1 row in set (0.001 sec)
```

3.2.3 Ευρετήρια

Εκφώνηση

Να ορίσετε κατάλληλα ευρετήρια (indexes) για τους πίνακες της ΒΔ και να δικαιολογήσετε την επιλογή σας με βάση την χρησιμότητα τους για τα ερωτήματα στα οποία χρησιμοποιούνται.

Απάντηση

```
-- == INDEXES == --


CREATE INDEX idx_perf_event_artist ON performances(event_ID,
artist_ID); Q04, Q05, Q09, Q10, Q11, Q13, Q14, Q15

CREATE INDEX idx_artist_name ON artist(artist_name); Q04

CREATE INDEX idx_perf_artist_event ON performances(artist_ID,
event_ID); Q02, Q04, Q05, Q10,Q11, Q13, Q15

CREATE INDEX idx_perf_group_event ON performances(group_ID,
event_ID); Q02, Q04, Q05, Q10,Q11, Q13, Q15


CREATE INDEX idx_ticket_visitor_event ON ticket(visitor_ID,
event_ID); Q06, Q09, Q15

CREATE INDEX idx_role_event_role ON
role_of_personel_on_event(event_ID, role); Q12

CREATE INDEX idx_visitor_full_name ON visitor(last_name,
first_name); Q06

CREATE INDEX idx_ticket_purchase_year_price ON
ticket(purchase_date, purchase_price); Q01

CREATE INDEX idx_perf_type_artist_event ON
performances(performance_type, artist_ID, event_ID); Q03

CREATE INDEX idx_role_event_role_personel ON
role_of_personel_on_event(role, personel_ID, event_ID); Q07, Q08
```

```
CREATE INDEX idx_personel_expertise ON personel(expertise_status); Q08

CREATE INDEX idx_festival_location_continent ON festival_location(festival_ID, continent); Q13
```

Τα παραπάνω προέκυψαν για να διευκολύνουμε τα queries και δεδομένου ότι κάποια indexes δημιουργούνται αυτόματα από τα αντίστοιχα foreign key constrains.

4 *Queries*

Εκφώνηση

Να σχεδιάσετε και εκτελέσετε τα ακόλουθα ερωτήματα:

(τα ερωτήματα είναι ισόβαθμα). Κάθε ερώτημα θα πρέπει να υλοποιείται με ένα query και να επιστρέφει ένα σύνολο αποτελεσμάτων. Όλα τα ερωτήματα πρέπει να επιστρέφουν έγκυρα αποτελέσματα, διαφορετικά δεν θα βαθμολογούνται.

** Για τα ερωτήματα 4 και 6, η απάντηση σας θα πρέπει να περιέχει εκτός από το query, εναλλακτικό Query Plan (πχ με force index), τα αντίστοιχα traces και τα συμπεράσματα σας από την μελέτη αυτών. 4 Να δοκιμάσετε διαφορετικές στρατηγικές join (π.χ. Nested Loop Join, Hash Join, Merge Join) για να αναλύσετε την επίδραση στη συνολική απόδοση.*

Q01

Στο ερώτημα αυτό θα βρούμε τα έσοδα του φεστιβάλ ανά έτος από την πώληση εισητηριών. Εκτελούμε SELECT YEAR(purchase_date) AS festival_year

- Η συνάρτηση YEAR() εξάγει το έτος από την ημερομηνία αγοράς (purchase_date). Το αποτέλεσμα ονομάζεται festival_year, δηλαδή το έτος στο οποίο έγινε η αγορά.

Στην συνέχεια κάνουμε SUM(purchase_price) AS total_revenue

- Η συνάρτηση συνόλου SUM() αθροίζει όλες τις τιμές του purchase_price για κάθε

έτος. Το άθροισμα ονομάζεται total_revenue, δηλαδή το συνολικό έσοδο από εισιτήρια.

Χρησιμοποιούμε το FROM ticket αφού τα δεδομένα προέρχονται από τον πίνακα ticket, όπου καταχωρούνται όλες οι πωλήσεις εισιτηρίων. Χρησιμοποιούμε επίσης WHERE purchase_price IS NOT NULL
για να φιλτράρουμε μόνο τις εγγραφές που έχουν καταγεγραμμένη τιμή αγοράς (αγνοούμε δηλαδή εισιτήρια χωρίς τιμή, για να μην επηρεάσουν τα στατιστικά).

Τέλος εκτελούμε GROUP BY festival_year για να ομαδοποιήσουμε τα αποτελέσματα κατά festival_year, ώστε να υπολογιστεί ξεχωριστό άθροισμα εσόδων για κάθε έτος και ORDER BY festival_year για να ταξινομίσουμε τα τελικά αποτελέσματα με αύξουσα σειρά έτους, ώστε να εμφανίζονται πρώτα οι παλαιότεροι χρόνοι και στο τέλος οι πιο πρόσφατοι.

Q02

Το ερώτημα επιστρέφει όλους τους φορείς (“entity”) που σχετίζονται με το είδος «rock» (είτε καλλιτέχνες είτε γκρουπ) μαζί με πληροφορία για το αν έχουν συμμετάσχει σε φεστιβάλ του 2024.

Με το πρώτο κομμάτι του κώδικα (μέχρι δηλαδή την γραμμή END AS has_participated) ελέγχουμε αν υπάρχει artist_ID - τότε είναι καλλιτέχνης, αλλιώς είναι γκρουπ την πληροφορία αυτή αποθηκεύουμε ως entity_type. Την πληροφοφία για το entity_ID και το entity_name είτε του μεμονωμένου καλλιτέχνη είτε του group ανάλογα με το ποιό δεν είναι NULL με την εντολή coalesce (αξιολογεί μια λίστα από εκφράσεις με τη σειρά και επιστρέφει την πρώτη τιμή που δεν είναι NULL.) Στο genre_name αποθηκεύουμε το όνομα του είδους που ήδη φιλτράρουμε στο τέλος (στην περίπτωσή μας rock). Στο has_participated έχουμε άλλο ένα CASE που βάζει ‘yes’ αν έχει βρεθεί στον υπο-ερώτημα συμμετοχής του 2024 είτε ως καλλιτέχνης είτε ως γκρουπ, αλλιώς ‘no’.

Στο κόμμάτι:

```
FROM genre g

-- Συνδέουμε με artist αν υπάρχει artist_ID
```

```
LEFT JOIN artist a
  ON g.artist_ID = a.artist_ID
-- Συνδέουμε με group αν υπάρχει group_ID
LEFT JOIN `group` gr
  ON g.group_ID = gr.group_ID
```

Ξεκινάμε από τον πίνακα genre ο οποίος περιέχει ξένο κλειδί είτε σε artist_ID είτε σε group_ID ενώ το άλλο πεδίο είναι NULL. Με τη χρήση ενός LEFT JOIN στους πίνακες artist και group, διατηρούμε όλες τις εγγραφές από τον πρώτο πίνακα και, όπου υπάρχει αντιστοίχιση, φέρνουμε επιπλέον τα αντίστοιχα δεδομένα από τον δεύτερο, χωρίς να χάνουμε γραμμές που δεν έχουν αντιστοιχία.

Στο κομμάτι:

```
-- Υπο-ερώτημα για artists που έπαιξαν το 2024
LEFT JOIN (
  SELECT DISTINCT p.artist_ID
  FROM performances p
  JOIN events e ON p.event_ID = e.event_ID
  JOIN festival f ON e.festival_ID = f.festival_ID
  WHERE YEAR(f.starting_date) = 2024
) AS art2024
  ON a.artist_ID = art2024.artist_ID
-- Υπο-ερώτημα για groups που έπαιξαν το 2024
LEFT JOIN (
  SELECT DISTINCT p.group_ID
  FROM performances p
  JOIN events e ON p.event_ID = e.event_ID
```

```
  JOIN festival f ON e.festival_ID = f.festival_ID
  WHERE YEAR(f.starting_date) = 2024
) AS grp2024
  ON gr.group_ID = grp2024.group_ID
```

Κάθε υπο-ερώτημα βρίσκει μοναδικά (DISTINCT) artist_ID ή group_ID που εμφανίζονται σε παραστάσεις (performances) συνδεδεμένες με φεστιβάλ του 2024 (YEAR(f.starting_date) = 2024). Έπειτα τα κάνουμε LEFT JOIN πίσω ώστε να "σημειώσουμε" ποιοι φορείς έπαιξαν φέτος.

Ενώ με το `WHERE g.genre_name = 'rock';` απλώς φιλτράρουμε το είδος μουσικής.

Q03

Σε αυτό το ερώτημα χρησιμοποιούμε το view artist_participation. Αυτό μας βοηθά να πάρουμε όλες τις συμμετοχές ενός καλλιτέχνη ανεξαρτήτως από το αν η συμμετοχή αυτή ήταν solo ή σε group. Το view αυτό απλοποιεί τον κώδικα και θα χρησιμοποιηθεί και σε επόμενα queries.

```sql
SELECT
    ap.artist_ID,
    a.artist_name,
    ap.festival_ID,
    COUNT(*) AS warmup_count
FROM
    artist_participations ap
    JOIN artist a ON ap.artist_ID = a.artist_ID
WHERE
    ap.performance_type = 'warm up'
GROUP BY
    ap.artist_ID,
    ap.festival_ID
HAVING
    COUNT(*) > 2
ORDER BY
    warmup_count DESC;
```

Επιλέγουμε τα attributes(στήλες) που μας ενδιαφέρουν, αναφερόμαστε στον πίνακα που μας ενδιαφέρει ή στην περίπτωσή μας view. Φιλτράρουμε με το WHERE, ομαδοποιούμε με το GROUP BY, φιλτράρουμε περεταίρω με το HAVING και τέλος ταξινομούμε με το ORDER BY.

Q04

Το ερώτημα υπολογίζει τους μέσους όρους βαθμολογίας ενός συγκεκριμένου καλλιτέχνη (Ernest Long), στρογγυλοποιώντας τα αποτελέσματα σε δύο δεκαδικά ψηφία, ενώ χρησιμοποιεί συνδέσεις (JOIN) για να φέρει δεδομένα από τους πίνακες review, ticket, events, performances και artist, φιλτράρει μη-NULL τιμές, περιορίζει το ερώτημα μέσω υπο-ερωτήματος στην artist και ομαδοποιεί το τελικό αποτέλεσμα ανά όνομα καλλιτέχνη.

Η AVG() υπολογίζει τον μέσο όρο της στήλης, παραβλέποντας τιμές NULL. Η ROUND(expr, 2) στρογγυλοποιεί το αποτέλεσμα του AVG σε 2 δεκαδικά ψηφία. Τα αποτελέσματα ονομάζονται avg_artist_performance και avg_overall_impression για ευκολότερη ανάγνωση.

FROM

review r

JOIN ticket t ON r.ticket_ID = t.ticket_ID

```
JOIN events e ON t.event_ID = e.event_ID
JOIN performances p ON e.event_ID = p.event_ID
AND t.event_ID = p.event_ID
JOIN artist a ON p.artist_ID = a.artist_ID
```

JOIN review → ticket συνδέει κάθε κριτική (review) με το αντίστοιχο εισιτήριο (ticket).

JOIN ticket → events → performances → artist δημιουργεί την αλυσίδα ώστε να δούμε σε ποια παράσταση (performances) του καλλιτέχνη αντιστοιχεί η κριτική.

Η διπλή συνθήκη ON e.event_ID = p.event_ID AND t.event_ID = p.event_ID διασφαλίζει ότι η παράσταση και το εισιτήριο αναφέρονται στο ίδιο event.

Στο παρακάτω κομμάτι φιλτράρουμε:

```
WHERE
    a.artist_ID = (
    SELECT
    artist_ID
    FROM
    artist
    WHERE
    artist_name = 'Ernest Long'
    )
    AND r.artist_performance IS NOT NULL
    AND r.overall_impression IS NOT NULL
```

Συγκεκριμένα η συνθήκη a.artist_ID = (…) χρησιμοποιεί υπο-ερώτημα (nested subquery) για να βρει το ID του καλλιτέχνη με όνομα ‘Ernest Long’, επιστρέφοντας μία μόνο τιμή για σύγκριση.

Επιπλέον φιλτράρονται οι εγγραφές ώστε οι βαθμολογίες artist_performance και overall_impression να μην είναι NULL. GROUP BY a.artist_name;

Στα ερωτήματα 4 και 6 φτιάξαμε δύο υλοποιήσεις και με την χρήση της εντολής EXPLAIN συγκρίναμε τα αποτελέσματα.

Για το ερώτημα 4 με τον β τρόπο:

```
-- With the indexes we saw it used Block Nested Loop Join
EXPLAIN FORMAT = JSON
SELECT
        a.artist_name,
        ROUND(AVG(r.artist_performance), 2) AS avg_artist_performance,
        ROUND(AVG(r.overall_impression), 2) AS avg_overall_impression
FROM
        review r FORCE INDEX (idx_review_ticket)
        JOIN ticket t FORCE INDEX (idx_ticket_event) ON r.ticket_ID = t.ticket_ID
        JOIN events e ON t.event_ID = e.event_ID
        JOIN performances p FORCE INDEX (idx_perf_event_artist) ON e.event_ID = p.event_ID
        AND t.event_ID = p.event_ID
        JOIN artist a FORCE INDEX (idx_artist_name) ON p.artist_ID = a.artist_ID
WHERE
        a.artist_ID = (
        SELECT
        artist_ID
        FROM
```

```
        artist FORCE INDEX (idx_artist_name)
        WHERE
        artist_name = ' Albert Carr'
        )
        AND r.artist_performance IS NOT NULL
        AND r.overall_impression IS NOT NULL
GROUP BY
        a.artist_name;


-- Without indexes Block Nested Loop Join
SET
        join_cache_level = 8;


EXPLAIN FORMAT = JSO
SELECT
        a.artist_name,
        ROUND(AVG(r.artist_performance), 2) AS avg_artist_performance,
        ROUND(AVG(r.overall_impression), 2) AS avg_overall_impression
FROM
        review r IGNORE INDEX (idx_review_ticket)
        JOIN ticket t IGNORE INDEX (idx_ticket_event) ON r.ticket_ID = t.ticket_ID
        JOIN events e ON t.event_ID = e.event_ID
        JOIN performances p IGNORE INDEX (idx_perf_event_artist) ON e.event_ID =
p.event_ID
        AND t.event_ID = p.event_ID
        JOIN artist a IGNORE INDEX (idx_artist_name) ON p.artist_ID = a.artist_ID
WHERE
```

```
        a.artist_ID = (
        SELECT
        artist_ID
        FROM
        artist IGNORE INDEX (idx_artist_name)
        WHERE
        artist_name = 'Albert Carr'
        )
GROUP BY
        a.artist_name;
```

Στην πρώτη υλοποίηση κάνουμε εξαναγκασμένη χρήση των index και παρατηρήσαμε από το EXPLAIN ότι έγινε χρήση του Block-Nested-Loop Join όπως φαίνεται και παρακάτω:

```
{
  "query_block": {
    "select_id": 1,
    "filesort": {
      "sort_key": "a.artist_name",
      "temporary_table": {
        "table": {
          "table_name": "a",
          "access_type": "index",
          "key": "idx_artist_name",
          "key_length": "1022",
          "used_key_parts": ["artist_name"],
          "rows": 50,
          "filtered": 100,
          "attached_condition": "a.artist_ID = (subquery#2)",
          "using_index": true
        },
        "block-nl-join": {
          "table": {
            "table_name": "t",
            "access_type": "index",
            "possible_keys": ["idx_ticket_event"],
            "key": "idx_ticket_event",
            "key_length": "5",
            "used_key_parts": ["event_ID"],
            "rows": 218,
            "filtered": 100,
            "using_index": true
          },
          "buffer_type": "flat",
          "buffer_size": "51Kb",
          "join_type": "BNL",
          "attached_condition": "t.event_ID is not null and t.event_ID is no
t null"
```


Στην δεύτερη υλοποίηση κάνουμε εξαναγκασμένη χρήση χωρίς index και παρατηρήσαμε από το EXPLAIN ότι έγινε χρήση του Nested-Loop Join.

Q05

Το ερώτημα αυτό γίνεται ομοίως με το ερώτημα 3 χρησιμοποιώντας δηλαδή το view και κατάλληλα φίλτρα. (ορίζουμε δύο CTE και κάνουμε την τελική επιλογή με SELECT)

Q06

Αυτό το ερώτημα επιλέγει το πλήρες όνομα του επισκέπτη, το όνομα της εκδήλωσης και τον μέσο όρο της συνολικής εντύπωσης που έδωσε στις κριτικές του, χρησιμοποιώντας συναρτήσεις συναρμολόγησης συμβολοσειρών, συγκεντρωτικές συναρτήσεις και συναρτήσεις στρογγυλοποίησης. Στη συνέχεια φιλτράρει μόνο τις κριτικές του επισκέπτη «Lindsey David», ομαδοποιεί τα αποτελέσματα ανά εκδήλωση και υπολογίζει τον μέσο όρο με δύο δεκαδικά ψηφία, εμφανίζοντας για κάθε εκδήλωση την τιμή avg_overall_impression.

Η εντολή SELECT καθορίζει ποιες στήλες θα εμφανιστούν στα αποτελέσματα.

Η συνάρτηση CONCAT(v.first_name, ' ', v.last_name) ενώνει το first_name με το last_name σε μία συμβολοσειρά για το visitor_name.

Με το AS δίνονται ψευδώνυμα (visitor_name, avg_overall_impression) για ευκολότερη ανάγνωση των στηλών.

JOIN visitor → ticket: συνδέει κάθε επισκέπτη (visitor) με τα εισιτήριά του (ticket) βάσει του visitor_ID

JOIN ticket → events: φέρνει το όνομα της εκδήλωσης (event_name) μέσω του event_ID

JOIN events → review: παίρνει τις κριτικές (review) που αντιστοιχούν στο κάθε εισιτήριο.

Η WHERE περιορίζει τις γραμμές μόνο στον επισκέπτη με first_name = 'Lindsey' και last_name = 'David', εξασφαλίζοντας ότι θα υπολογιστούν μόνο οι κριτικές αυτού του επισκέπτη.

Η AVG(r.overall_impression) υπολογίζει τον μέσο όρο των τιμών overall_impression, παραβλέποντας τυχόν NULL τιμές.

Η ROUND(..., 2) στρογγυλοποιεί το αποτέλεσμα του AVG σε δύο δεκαδικά ψηφία, προσφέροντας πιο καθαρή παρουσίαση των τιμών.

Η GROUP BY e.event_name συγκεντρώνει τις γραμμές ανά event_name, επιτρέποντας στον AVG να υπολογίσει ξεχωριστά τον μέσο όρο για κάθε εκδήλωση.

Για το ερώτημα 6 με τον β τρόπο:

```
-- using forced index it used Nested Loop Join
EXPLAIN FORMAT = JSON
SELECT
        CONCAT (v.first_name, ' ', v.last_name) AS visitor_name,
        e.event_name,
        ROUND(AVG(r.overall_impression), 2) AS avg_overall_impression
FROM
        visitor v FORCE INDEX (idx_visitor_full_name)
        JOIN ticket t ON v.visitor_ID = t.visitor_ID
        JOIN events e ON t.event_ID = e.event_ID
        JOIN review r ON t.ticket_ID = r.ticket_ID
WHERE
        v.first_name = 'Jason'
```

```
    AND v.last_name = 'Perez'
GROUP BY
    e.event_name;


-- Wihtout indexes still used Nested Loop Join
EXPLAIN FORMAT = JSON
SELECT
    CONCAT (v.first_name, ' ', v.last_name) AS visitor_name,
    e.event_name,
    ROUND(AVG(r.overall_impression), 2) AS avg_overall_impression
FROM
    visitor v IGNORE INDEX (idx_visitor_full_name)
    JOIN ticket t ON v.visitor_ID = t.visitor_ID
    JOIN events e ON t.event_ID = e.event_ID
    JOIN review r ON t.ticket_ID = r.ticket_ID
WHERE
    v.first_name = 'Jason'
    AND v.last_name = 'Perez'
GROUP BY
    e.event_name;
```

Στο ερώτημα 6 και στις δύο υλοποιήσεις που κάνουμε παρατηρήσαμε απο το EXPLAIN ότι έγινε χρήση του Nested-Loop Join.

Επειδή η βάση δεδομένων μας είναι μικρή δεν παρατηρήσουμε κάποια βελτίωση στην ταχύτητα εκτέλεσης στα ερωτήματα 4 και 6, ενώ θεωρητικά τα indexes επιταχύνουν την διαδικασία.

Q07

Το ερώτημα υπολογίζει για κάθε φεστιβάλ το μέσο όρο “expertise score” του τεχνικού προσωπικού, όπου κάθε επίπεδο εμπειρογνωμοσύνης («intern», «beginer», «intermidiate», «experienced», «very_experienced») μετατρέπεται σε αντίστοιχη αριθμητική τιμή από 1 έως 5. Στη συνέχεια, ομαδοποιεί τα αποτελέσματα ανά festival_ID, ταξινομεί κατά αύξουσα σειρά του μέσου όρου και επιστρέφει μόνο το φεστιβάλ με το χαμηλότερο σκορ.

Η εσωτερική σύνδεση (JOIN) εξασφαλίζει ότι λαμβάνονται μόνο τα τεχνικά ρόστερ (r.role = 'technical') που υπάρχουν και στους τέσσερις πίνακες.

Το φιλτράρισμα γίνεται με το WHERE το οποίο επιλέγει αποκλειστικά τις εγγραφές όπου το πεδίο role του πίνακα role_of_personel_on_event είναι ακριβώς technical. Τέλος, η ομαδοποιήση γίνεται με το GROUP BY το οποίο ομαδοποιεί τις γραμμές για κάθε festival_ID, επιτρέποντας στον AVG να υπολογίσει τον μέσο όρο ανά φεστιβάλ, ταξινόμηση με το ORDER BY και περιορισμός στο να επιστρέφει μόνο ένα δηλαδή αυτόν το χαμηλότερο score.

Q08

Αυτό το ερώτημα επιστρέφει το personel_ID και το πλήρες όνομα κάθε μέλους προσωπικού που έχει ορισμένο επίπεδο εμπειρογνωμοσύνης (expertise_status IS NOT NULL) και δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία (2024-08-10). Επιλέγει το μοναδικό αναγνωριστικό personel_ID από τον πίνακα personel. Με τη συνάρτηση CONCAT συνενώνει το first_name και το last_name σε μία στήλη full_name. Φιλτράρει με WHERE μόνο τα μέλη προσωπικού που έχουν καθορισμένο επίπεδο εμπειρογνωμοσύνης, απορρίπτοντας όσα είναι χωρίς τιμή (NULL). Το NOT EXISTS ελέγχει αν δεν υπάρχει κάποια γραμμή στο υπο-ερώτημα που να αντιστοιχεί στο ίδιο personel_ID. Το υπο-ερώτημα ενώνει τον πίνακα

role_of_personel_on_event με τον πίνακα events για να εξετάσει κάθε ανάθεση προσωπικού σε συγκεκριμένο event. Η συνάρτηση DATE() εξάγει μόνο την ημερομηνία από την event_start_time, αγνοώντας την ώρα και συγκρίνει αυτή την ημερομηνία με την σταθερή τιμή '2024-08-10'.

Q09

Η λέξη-κλειδί WITH (πχ visitor_performance_counts AS (...)) ορίζει ένα ή περισσότερα CTEs Common Table Expressions , δηλαδή προσωρινά ονόματα για υπο-ερωτήματα τα οποία μπορούν να χρησιμοποιηθούν πολλαπλές φορές στο κύριο ερώτημα. Οι CTEs βελτιώνουν την αναγνωσιμότητα και επιτρέπουν τον επαναχρησιμοποιήσιμο κώδικα μέσα σε ένα μόνο statement. Στο πρώτο Πρώτη CTE visitor_performance_counts χρησιμοποιούμε JOINs για να ενώσουμε τους πίνακες visitor → ticket → events → performances ώστε να υπολογίσουμε πόσες παραστάσεις έχει παρακολουθήσει κάθε επισκέπτης σε κάθε έτος. Με το YEAR() εξάγουμε το έτος από την ημερομηνία event_start_time. Με το GROUP BY … HAVING COUNT(...) > 3 ομαδοποιούμε ανά επισκέπτη/έτος και κρατάμε μόνο όσους έχουν πάνω από τρεις παραστάσεις. Στο δεύτερο CTE matching_counts κάνουμε αναφορά στο πρώτο CTE, ομαδοποιούμε με GROUP BY performance_count και φιλτράρουμε με HAVING COUNT(*) > 1.

Με τον παρακάτω κώδικα:

```
SELECT
        vpc.visitor_ID,
        vpc.event_year,
    vpc.performance_count
FROM
    visitor_performance_counts vpc
        JOIN matching_counts mc ON vpc.performance_count = mc.performance_count
ORDER BY
    vpc.performance_count DESC,
        vpc.visitor_ID;
```

Συνδέουμε τις δύο CTEs με JOIN στο κοινό πεδίο performance_count, ώστε να εμφανιστούν μόνο οι επισκέπτες των οποίων ο αριθμός εμφανίσεων συναντάται σε τουλάχιστον δύο επισκέπτες.

ORDER BY … DESC: Ταξινομεί κατά φθίνουσα σειρά του performance_count, δείχνοντας πρώτα τους πιο "ενεργητικούς" επισκέπτες, και χρησιμοποιεί το visitor_ID ως tie-breaker.

Q10

Σε αυτό το ερώτημα εντοπίζουμε τα τρία πιο συχνά ζεύγη ειδών ("genre-pairs") για καλλιτέχνες και γκρουπ, μετρώντας σε πόσα διακριτά φεστιβάλ εμφανίστηκαν μαζί.

```
-- Top 3 genre-pairs για artists
SELECT
        'artist'                    AS entity_type,
        LEAST(g1.genre_name, g2.genre_name)  AS genre1,
        GREATEST(g1.genre_name, g2.genre_name)   AS genre2,
        COUNT(DISTINCT f.festival_ID)          AS appearances
FROM genre g1
JOIN genre g2
  ON g1.artist_ID   = g2.artist_ID
 AND g1.genre_name < g2.genre_name
JOIN artist a
  ON a.artist_ID   = g1.artist_ID
JOIN performances p
  ON p.artist_ID   = a.artist_ID
JOIN events e
```

```
  ON p.event_ID = e.event_ID
JOIN festival f
  ON e.festival_ID = f.festival_ID
GROUP BY genre1, genre2

UNION ALL

-- Top 3 genre-pairs για groups
SELECT
        'group'                              AS entity_type,
        LEAST(g1.genre_name, g2.genre_name)  AS genre1,
        GREATEST(g1.genre_name, g2.genre_name)   AS genre2,
        COUNT(DISTINCT f.festival_ID)        AS appearances
FROM genre g1
JOIN genre g2
  ON g1.group_ID    = g2.group_ID
 AND g1.genre_name < g2.genre_name
JOIN `group` gr
  ON gr.group_ID    = g1.group_ID
JOIN performances p
  ON p.group_ID= gr.group_ID
JOIN events e
  ON p.event_ID = e.event_ID
JOIN festival f
  ON e.festival_ID = f.festival_ID
GROUP BY genre1, genre2
```

```
ORDER BY appearances DESC
LIMIT 3;
```

Χρησιμοποιεί self-joins για να παράγει όλα τα μοναδικά ζεύγη ειδών ανά οντότητα, τις συναρτήσεις LEAST και GREATEST για την κανονικοποίηση της σειράς τους, COUNT(DISTINCT) για την αποφυγή διπλών καταμετρήσεων φεστιβάλ, UNION ALL για να ενώσει τα αποτελέσματα των δύο ερωτημάτων, και στο τέλος ORDER BY … DESC LIMIT 3 για να φέρει τα κορυφαία τρία ζεύγη.

Ο πίνακας genre γίνεται self-join (g1 JOIN genre g2 ON g1.artist_ID = g2.artist_ID AND g1.genre_name < g2.genre_name) ώστε να δημιουργηθούν όλα τα δυνατά μοναδικά ζεύγη ειδών ανά καλλιτέχνη, αποφεύγοντας διπλοεγγραφές με τη συνθήκη g1.genre_name < g2.genre_name.

Με το LEAST() επιστρέφουμε την ελάχιστη τιμή μεταξύ των δύο ειδών ώστε να έχει πάντα πρώτο το "μικρότερο" λεξικογραφικά όνομα. Με το GREATEST() επιστρέφουμε τη μέγιστη τιμή, κανονικοποιώντας τη σειρά των ονομάτων έτσι ώστε genre1 ≤ genre2. Με το COUNT(DISTINCT f.festival_ID) μετράμε μόνο τον αριθμό διακριτών φεστιβάλ όπου εμφανίστηκε το κάθε ζεύγος, αποφεύγοντας διπλές καταμετρήσεις. Τα αποτελέσματα self-join συνδέονται με τους πίνακες artist → performances → events → festival ώστε να μετρηθούν οι εμφανίσεις των ειδών σε κάθε φεστιβάλ και ομαδοποιούνται ανά κανονικοποιημένο ζεύγος (genre1, genre2), επιτρέποντας στη συνάρτηση COUNT(DISTINCT) να λειτουργήσει σωστά με την εντολή GROUP BY genre1, genre 2. Τέλος, χρησιμοποιούμε UNION ALL για να συνενώσουμε τα δύο σύνολα αποτελεσμάτων, διατηρώντας όλα τα rows (χωρίς αφαίρεση διπλοτύπων), ορίζουμε φθίνουσα ORDER BY .. DESC σειρά και περιορίζουμε την επιστροφή μόνο 3 αποτελεσμάτων (LIMIT 3).

Q11

```sql
WITH artist_festival_counts AS (
    SELECT
        artist_ID,
        artist_name,
        COUNT(DISTINCT festival_ID) AS total_festival_participations
    FROM
        artist_participations
    GROUP BY
        artist_ID, artist_name
),
max_participation AS (
    SELECT MAX(total_festival_participations) AS max_count
    FROM artist_festival_counts
)
SELECT
    afc.artist_ID,
    afc.artist_name,
    afc.total_festival_participations
FROM
    artist_festival_counts afc,
    max_participation mp
WHERE
    afc.total_festival_participations <= mp.max_count - 5
ORDER BY
    total_festival_participations DESC;
```

Πάλι όμοια με τα Q03, Q05 και χρησιμοποιώντας το view artist_participation.

Q12

```sql
SELECT
    f.festival_ID,
    e.festival_day,
    r.role,
    COUNT(DISTINCT r.personel_ID) AS required_personnel
FROM
    role_of_personel_on_event r
    JOIN events e ON r.event_ID = e.event_ID
    JOIN festival f ON e.festival_ID = f.festival_ID
GROUP BY
    f.festival_ID,
    e.festival_day,
    r.role
ORDER BY
    f.festival_ID,
    e.festival_day,
    r.role;
```

Το ερώτημα συγκεντρώνει για κάθε φεστιβάλ και ημέρα του φεστιβάλ πόσους μοναδικούς υπαλλήλους απαιτεί κάθε role, χρησιμοποιώντας συνδέσεις μεταξύ των πινάκων role_of_personel_on_event, events και festival, ομαδοποίηση (GROUP BY)

και μέτρηση μοναδικών τιμών με COUNT(DISTINCT…). Το festival_ID είναι το μοναδικό αναγνωριστικό κάθε φεστιβάλ, η festival_day είναι η συγκεκριμένη ημέρα του φεστιβάλ , role: ο ρόλος του προσωπικού (π.χ. “technical”). Χρησιμοποιούμε COUNT(DISTINCT r.personel_ID) για να υπολογίσουμε τον αριθμό μοναδικών personel_ID για κάθε συνδυασμό φεστιβάλ-ημέρας-ρόλου, αποφεύγοντας διπλές καταμετρήσεις. Με τα FROM … JOIN συνδέουμε τον πίνακα role_of_personel_on_event με τον events μέσω event_ID ώστε να βρούμε σε ποια ημέρα κάθε ρόλος εφαρμόζεται και ενώνουμε τα events με τον πίνακα festival μέσω festival_ID προκειμένου να ταξινομήσουμε ανά φεστιβάλ. Τέλος ομαδοποιούμε ανά φεστιβάλ–ημέρα–ρόλο, επιτρέποντας στη συνάρτηση COUNT(DISTINCT…) να υπολογίσει σωστά το πλήθος παρ’ όλο που υπάρχουν πολλαπλές εγγραφές για κάθε συνδυασμό και ταξινομούμε τα αποτελέσματά μας.

```
ORDER BY
    f.festival_ID,
    e.festival_day,
    r.role;
```

Q13

```sql
WITH artist_continents AS (
    SELECT
        ap.artist_ID,
        ap.artist_name,
        fl.continent
    FROM
        artist_participations ap
        JOIN festival_location fl ON ap.festival_ID = fl.festival_ID
    GROUP BY
        ap.artist_ID, ap.artist_name, fl.continent
),
continent_counts AS (
    SELECT
        artist_ID,
        artist_name,
        COUNT(DISTINCT continent) AS num_of_continents
    FROM artist_continents
    GROUP BY artist_ID, artist_name
)
SELECT
    artist_ID,
    artist_name,
    num_of_continents
FROM
    continent_counts
WHERE
    num_of_continents >= 3
ORDER BY
    num_of_continents DESC;
```

Όμοια με Q03, Q05, Q11 χρησιμοποιούμε το view για να επιστρέψουμε τους καλλιτέχνες που έχουν εμφανιστεί σε φεστιβάλ σε **τουλάχιστον τρεις διαφορετικές ηπείρους. Ως artist_continents** για κάθε καλλιτέχνη συλλέγουμε σε ποιες ηπείρους έχει παίξει (βάσει festival_location.continent). Ως **continent_counts** μετράμε πόσες **μοναδικές** ηπείρους έχει ο καθένας (COUNT(DISTINCT continent)), ενώ στο τελικό SELECT παίρνουμε μόνο όσους έχουν num_of_continents ≥ 3 και τους ταξινομούμε με φθίνουσα ως προς τον αριθμό ηπείρων σειρά.

Q14

Το ερώτημα εντοπίζει ποια μουσικά είδη είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές με τουλάχιστον 3 εμφανίσεις ανά έτος. Οι εμφανίσεις αυτές μπορούν να είναι είτε από καλλιτέχνες είτε από group. Το τελικό αποτέλεσμα

επιστρέφει, για κάθε τέτοιο ζεύγος, το όνομα του είδους, το πρώτο και δεύτερο έτος, καθώς και τον κοινό αριθμό εμφανίσεων.

Πάλι χρησιμοποιούμε CTE (1. artist_genre_year_counts και 2. group_genre_year_counts).

```
WITH
    -- Genre-year counts for artists
    artist_genre_year_counts AS (
        SELECT
            g.genre_name,
            YEAR(f.starting_date) AS year,
            COUNT(*)              AS appearances
        FROM genre      g
        JOIN artist     a  ON g.artist_ID = a.artist_ID
        JOIN performances p ON p.artist_ID = a.artist_ID
        JOIN events     e  ON p.event_ID   = e.event_ID
        JOIN festival   f  ON e.festival_ID = f.festival_ID
        GROUP BY
            g.genre_name,
            YEAR(f.starting_date)
        HAVING
            COUNT(*) >= 3
    ),
```

Στο πρώτο ξεκινάμε από τον πίνακα genre, βρίσκοντας σε ποιόν καλλιτέχνη ανήκει κάθε genre (JOIN artist). Συνδέουμε τις παραστάσεις του καλλιτέχνη (JOIN performances) και μέσω των events με τα festival. Ομαδοποιούμε ανά genre_name και έτος (YEAR(f.starting_date)). Με το HAVING COUNT(*) >= 3 κρατάμε μόνο τα είδη που στο συγκεκριμένο έτος εμφανίστηκαν **τουλάχιστον 3 φορές**.

```sql
-- Genre-year counts for groups
group_genre_year_counts AS (
    SELECT
        g.genre_name,
        YEAR(f.starting_date) AS year,
        COUNT(*)             AS appearances
    FROM genre      g
    JOIN `group`    gr ON g.group_ID = gr.group_ID
    JOIN performances p  ON p.group_ID = gr.group_ID
    JOIN events     e  ON p.event_ID   = e.event_ID
    JOIN festival   f  ON e.festival_ID = f.festival_ID
    GROUP BY
        g.genre_name,
        YEAR(f.starting_date)
    HAVING
        COUNT(*) >= 3
),
```

Στο δεύτερο CTE επαναλαμβάνουμε την αλλά για **γκρουπ** αντί για μεμονωμένους καλλιτέχνες.

```sql
-- Union both sets into one CTE
genre_year_counts AS (
    SELECT * FROM artist_genre_year_counts
    UNION ALL
    SELECT * FROM group_genre_year_counts
)
```

Με το παραπάνω κομμάτι ενώνουμε τα δύο σύνολα με UNION ALL, ώστε το νέο CTE να περιέχει όλες τις εγγραφές και από καλλιτέχνες και από γκρουπ.

Σημειώνουμε ότι χρησιμοποιούμε UNION ALL (και όχι UNION) για να διατηρήσουμε πιθανά διπλότυπα εμφάνισης, αν ένα genre σε ένα έτος έχει τόσο καλλιτέχνες όσο και γκρουπ με ≥3 εμφανίσεις.

```sql
SELECT
    g1.genre_name,
    g1.year   AS year1,
    g2.year   AS year2,
    g1.appearances
FROM
    genre_year_counts g1
    JOIN genre_year_counts g2
     ON g1.genre_name = g2.genre_name
    AND g2.year       = g1.year + 1
    AND g2.appearances = g1.appearances
ORDER BY
    g1.genre_name,
    g1.year;
```

Επιλέγουμε τα attributes του πίνακα που επιστρέφει το quert genre_name, year1 (το πρώτο έτος), year2 (το αμέσως επόμενο), appearances (ο κοινός αριθμός εμφανίσεων).

Χρησιμοποιούμε Self-Join για να συνδέσουμε το genre_year_counts με τον εαυτό του, έτσι ώστε για κάθε εγγραφή g1 να βρούμε την εγγραφή g2 του επόμενου έτους (g2.year = g1.year + 1) ίδιου genre. Η Ταξινόμηση γίνεται κατά genre_name και year1, ώστε τα ζεύγη να εμφανίζονται σε ευανάγνωστη σειρά.

Q15

Το ερώτημα αυτό βρίσκει τους top-5 επισκέπτες που έχουν δώσει συνολικά την υψηλότερη βαθμολόγηση σε ένα καλλιτέχνη και επιστρέφει για τους επισκέπτες αυτούς όνομα επισκέπτη, όνομα καλλιτέχνη και συνολικό σκορ βαθμολόγησης.

Με το SELECT επιλέγουμε τις στήλες v.visitor_ID, a.artist_ID, a.artist_name. Χρησιμοποιούμε το CONCAT για να συγχωνεύσουμε το όνομα και επώνυμο του επισκέπτη σε μια στήλη. Στην συνέχεια μετατρέπυμε κάθε κριτική (η οποία μπορεί να αποθηκεύεται ως κείμενο ή null) σε μη-αρνητικό ακέραιο αριθμό και αθροίζουμε τις

πέντε μετατρεμμένες τιμές για κάθε συνδυασμό επισκέπτη–καλλιτέχνη, παραβλέποντας αυτόματα τιμές NULL. Οι εσωτερικές συνδέσεις(JOINS) εξασφαλίζουν ότι λαμβάνουμε μόνο κριτικές που συνδέονται με έγκυρα εισιτήρια, επισκέπτες, εκδηλώσεις, παραστάσεις και καλλιτέχνες. Φιλτράρουμε ομαδοποιούμε ταξινομούμε και περιορίζουμε τα αποτελέσματα σε πλήθος 5.

s

5 *Authorization*

Με το authentication.sql δημιουργούμε ή επαναδημιουργούμε συγκεκριμένους χρήστες. (Κάθε DROP USER IF EXISTS διαγράφει τον τυχόν υπάρχοντα λογαριασμό πριν από τη δημιουργία, αποφεύγοντας τα σφάλματα 1396. Κάθε CREATE USER φτιάχνει τον χρήστη με το αντίστοιχο password).

Με το GRANT αναθέτουμε συγκεκριμένα δικαιώματα στους χρήστες που δημιουργήσαμε. Συγκεκριμένα:

– Ο **admin** έχει πλήρη δικαιώματα (ALL PRIVILEGES) σε όλη τη βάση db1.

– Ο **manager** μπορεί να διαχειριστεί (SELECT/INSERT/UPDATE/DELETE) μόνο τα δεδομένα των πινάκων events, performances, ticket, review.

– Ο **personel_ops** διαχειρίζεται μόνο τον πίνακα personel και τη σχέση role_of_personel_on_event.

– Ο **visitor_portal** έχει δικαίωμα ανάγνωσης και προσθήκης σε visitor και ticket, και μόνο εισαγωγής σε review.

– Το **resale_bot** διαχειρίζεται την ουρά μεταπώλησης (resale_queue) και προσωρινά ταίρια (temp_resale_matches).

– Ο **analytics** είναι μόνο για ανάγνωση (SELECT) σε όλη τη βάση.

Τέλος, το FLUSH PRIVILEGES; ανανεώνει την cache δικαιωμάτων, ώστε οι αλλαγές να ισχύουν οι αλλαγές που κάναμε.