Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Εξάμηνο 6ο: Βάσεις Δεδομένων

Εξαμηνιαία Εργασία: “Pulse University Music Festival”

Περίληψη

Η παρούσα εργαστηριακή αναφορά αφορά την εξαμηνιαία εργασία στα πλαίσια του εργαστηρίου του μαθήματος Βάσεις Δεδομένων. Στην άσκηση αυτή σχεδιάσαμε και υλοποιήσαμε ένα σύστημα αποθήκευσης και διαχείρισης πληροφοριών που απαιτούνται για την οργάνωση και την διεξαγωγή του διεθνούς φεστιβάλ μουσικής, Pulse University.

Καβαδάκης Σωτήρης – **03122035**

Βυζηργιαννάκης Γιώργος – **03122108**

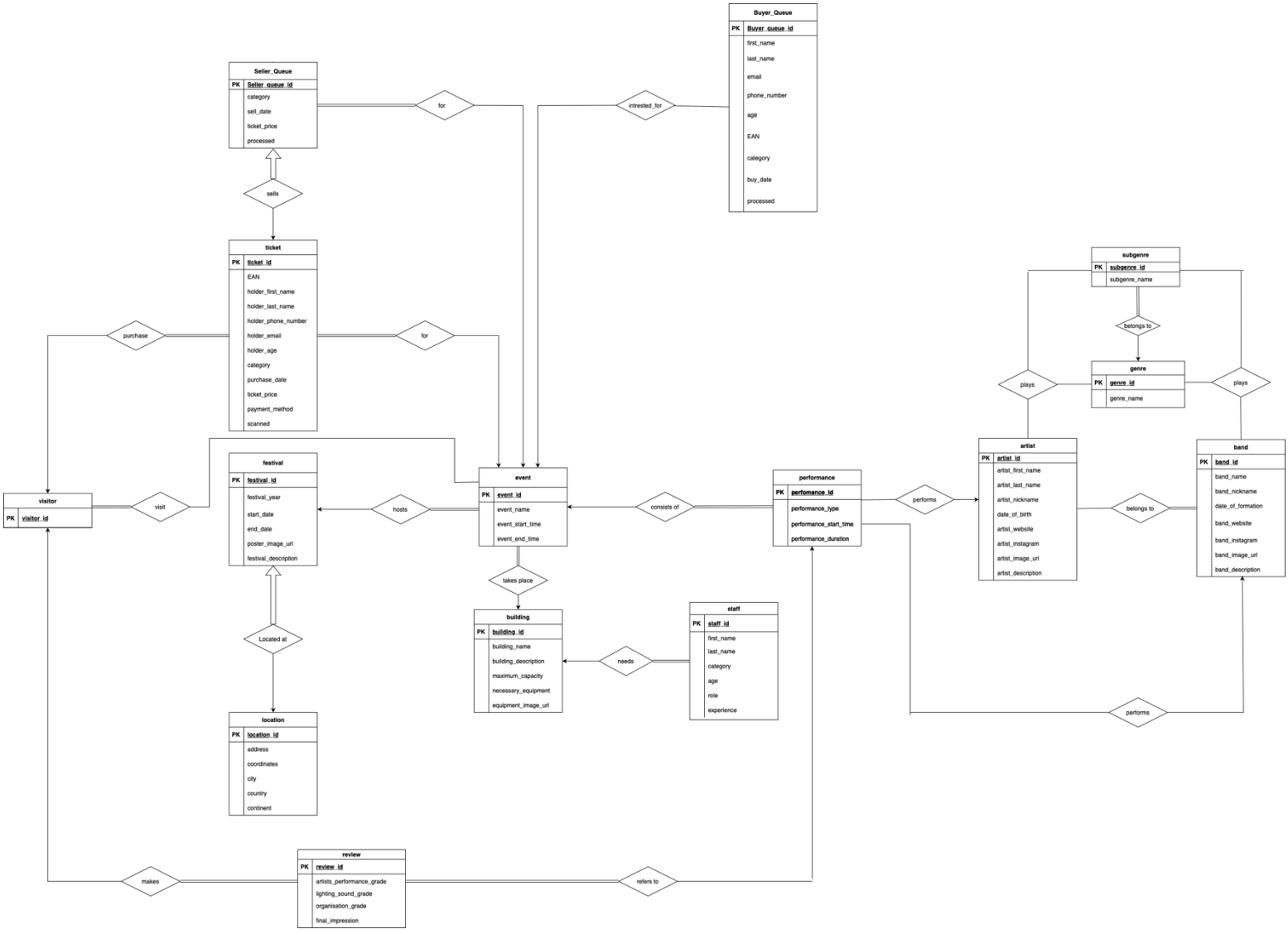
Δαλαμπέκης Ιγνάτιος-Μάρκελλος – **03122017**

Περιεχόμενα

A. Σχεδιασμός και υλοποίηση Βάσης Δεδομένων

A.1. Διάγραμμα Οντοτήτων – Συσχετίσεων (Entity-Relationship Diagram)

Σχεδιάζουμε το ER διάγραμμα που προκύπτει από την εκφώνηση, όπως φαίνεται στην Εικόνα 1:



Το παραπάνω διάγραμμα περιλαμβάνει όλες τις βασικές οντότητες όπως festival, location, event, building, staff, performance, artist, band, genre, subgenre, ticket, visitor, review, Seller και Buyer Queue με τα χαρακτηριστικά τους, όπως αυτά ορίζονται από την εκφώνηση, καθώς και τις μεταξύ τους σχέσεις.

Παρακάτω παρουσιάζονται οι σχέσεις μεταξύ των επιμέρους οντοτήτων:

- **Festival – Location :**

Κάθε φεστιβάλ πραγματοποιείται σε μία συγκεκριμένη τοποθεσία (location), και κάθε τοποθεσία φιλοξενεί ακριβώς ένα φεστιβάλ. Συνεπώς, η σχέση αυτή είναι **1:1** από festival προς location και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά του festival, καθώς κάθε φεστιβάλ πρέπει να έχει καθορισμένη τοποθεσία.

- **Festival – Event :**

Κάθε φεστιβάλ περιλαμβάνει παραστάσεις (events), και κάθε event πραγματοποιείται ακριβώς σε ένα φεστιβάλ. Συνεπώς, η σχέση αυτή είναι **1:N** από festival προς event και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά του event, καθώς κάθε event πρέπει να συνδέεται με ένα συγκεκριμένο φεστιβάλ.

- **Event – Building :**

Κάθε event πραγματοποιείται σε μια συγκεκριμένη μουσική σκήνη/κτήριο (building), και κάθε κτήριο φιλοξενεί πολλά events. Συνεπώς, η σχέση αυτή είναι **1:N** από building προς event και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά του event, καθώς κάθε event πρέπει να έχει καθορισμένο κτίριο.

- **Building – Staff :**

Κάθε μουσική σκήνη/κτίριο (building) χρειάζεται προσωπικό, δηλαδή διάφορους υπαλλήλους (staff id), και κάθε υπάλληλος εργάζεται σε ένα κτίριο (building). Συνεπώς, η σχέση αυτή είναι **1:N** από building προς staff και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά του staff, καθώς κάθε υπάλληλος πρέπει να εργάζεται σε καθορισμένο κτίριο.

- **Event – Performance :**

Κάθε event αποτελείται από εμφανίσεις καλλιτεχνών (performances) και κάθε εμφάνιση πραγματοποιείται σε ένα συγκεκριμένο event. Συνεπώς, η σχέση αυτή είναι **1:N** από event προς performance και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά του performance, καθώς κάθε εμφάνιση πρέπει να σχετίζεται με καθορισμένο event.

- **Performance – Artist / Performance – Band :**

Κάθε εμφάνιση σχετίζεται με έναν συγκεκριμένο καλλιτέχνη (artist) ή συγκρότημα (band), αντίστοιχα, και κάθε καλλιτέχνης ή μπάντα μπορούν να συμμετάσχουν σε πολλές εμφανίσεις. Συνεπώς, οι σχέσεις αυτές είναι **1:N** από artist/band προς performance.

- **Artist – Genre / Artist – Subgenre :**

Κάθε καλλιτέχνης (artist) μπορεί να ερμηνεύσει διάφορα μουσικά είδη (genres) και υποείδη (subgenres) και κάθε μουσικό είδος και υποείδος μπορεί να σχετίζεται με πολλούς καλλιτέχνες. Συνεπώς, οι σχέσεις αυτές είναι **N:N** από artist προς genre και subgenre.
ομοίως Band – Genre / Band– Subgenre

- **Genre – Subgenre :**

Κάθε μουσικό είδος (genre) περιλαμβάνει διάφορα υποείδη (subgenres) και κάθε υποείδος (subgenre) σχετίζεται μόνο με ένα είδος. Συνεπώς, η σχέση αυτή είναι **1:N** από genre προς subgenre και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά του subgenre, καθώς κάθε υποείδος πρέπει να αντιστοιχεί σε καθορισμένο μουσικό είδος.

- **Artist – Band:**

Κάθε καλλιτέχνης (artist) μπορεί να ανήκει σε διάφορα συγκροτήματα (bands) και κάθε συγκρότημα αποτελείται από πολλούς καλλιτέχνες. Συνεπώς, η σχέση αυτή είναι **N:N** από artist προς band και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά της band, καθώς κάθε συγκρότημα πρέπει να αποτελείται από καθορισμένους καλλιτέχνες.

- **Visitor – Event:**

Κάθε επισκέπτης (visitor) μπορεί να παρακολουθήσει διάφορες παραστάσεις (events) και κάθε παράσταση αποτελείται από πολλούς επισκέπτες. Συνεπώς, η σχέση αυτή είναι **N:N** από visitor προς event και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά των visitors, καθώς κάθε επισκέπτης πρέπει να σχετίζεται με καθορισμένο event, ανεξαρτήτως του αν έχει παρακολουθήσει την παράσταση.

- **Visitor – Review :**

Κάθε επισκέπτης (visitor) μπορεί να κάνει διάφορες αξιολογήσεις (reviews) και κάθε αξιολόγηση γίνεται από έναν συγκεκριμένο επισκέπτη. Συνεπώς, η σχέση αυτή είναι **1:N** από visitor προς review και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά των reviews, καθώς κάθε αξιολόγηση πρέπει να σχετίζεται με έναν καθορισμένο επισκέπτη.

- **Review – Performance :**

Κάθε αξιολόγηση (review) αφορά μια συγκεκριμένη εμφάνιση (performance) και κάθε εμφάνιση μπορεί να αξιολογηθεί από διάφορους επισκέπτες. Συνεπώς, η σχέση αυτή είναι **1:N** από performance προς review και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά των reviews, καθώς κάθε αξιολόγηση πρέπει να σχετίζεται με μια καθορισμένη παράσταση.

- **Visitor – Ticket:**

Κάθε επισκέπτης (visitor) μπορεί να αγοράσει διάφορα εισιτήρια (tickets) και κάθε εισιτήριο αφορά μόνο έναν συγκεκριμένο επισκέπτη. Συνεπώς, η σχέση αυτή είναι **1:N** από visitor προς ticket και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά των tickets, καθώς κάθε εισιτήριο πρέπει να σχετίζεται με έναν καθορισμένο επισκέπτη.

- **Ticket – Event :**

Κάθε εισιτήριο (ticket) αφορά μόνο μία συγκεκριμένη παράσταση (event) και κάθε παράσταση μπορεί να σχετίζεται με πολλά εισιτήρια. Συνεπώς, η σχέση αυτή είναι **1:N** από event προς ticket και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά των tickets, καθώς κάθε εισιτήριο πρέπει να σχετίζεται με μια καθορισμένη παράσταση.

- **Ticket – Seller Queue :**

Κάθε εισιτήριο (ticket) μπορεί να καταχωρηθεί σε μια εγγραφή της ουράς μεταπώλησης (Seller Queue), όταν ο κάτοχός του επιθυμεί να το διαθέσει προς πώληση, και κάθε εγγραφή της Seller Queue αντιστοιχεί σε ένα μόνο εισιτήριο. Συνεπώς, η σχέση αυτή είναι **1:1** από ticket προς Seller Queue και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά της Seller Queue, αφού κάθε εγγραφή της πρέπει να σχετίζεται με ένα εισιτήριο.

- **Seller Queue – Event :**

Κάθε εγγραφή της ουράς μεταπώλησης (Seller Queue) αφορά μια συγκεκριμένη παράσταση (event) και κάθε παράσταση μπορεί να σχετίζεται με διάφορες εγγραφές της ουράς μεταπώλησης. Συνεπώς, η σχέση αυτή είναι **1:N** από event προς Seller Queue και παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά της Seller Queue, αφού κάθε εγγραφή τής πρέπει να σχετίζεται με μια παράσταση.

- **Buyer Queue – Event :**

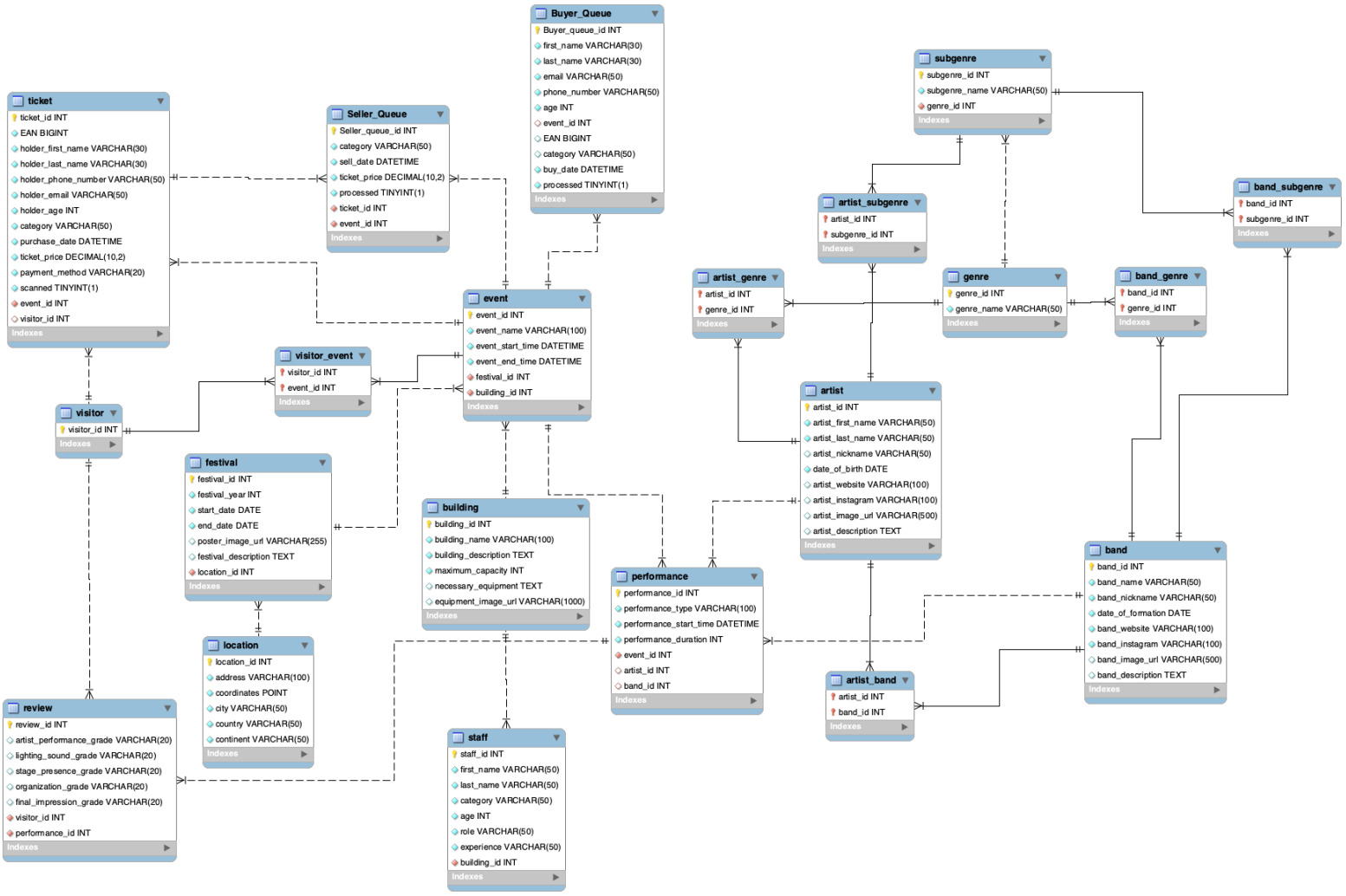
Κάθε εγγραφή της ουράς αναμονής αγοραστών (Buyer Queue) αφορά μία συγκεκριμένη παράσταση (event). Σε αυτήν, κάθε επισκέπτης εκδηλώνει ενδιαφέρον είτε για συγκεκριμένη παράσταση και κατηγορία εισιτηρίου, είτε για ένα συγκεκριμένο εισιτήριο που είναι διαθέσιμο προς πώληση. Κάθε παράσταση μπορεί να σχετίζεται με πολλές εγγραφές της Buyer Queue. Συνεπώς, η σχέση αυτή είναι **1:N** από **event** προς **Buyer Queue** χωρίς να παρουσιάζει **ολική συμμετοχή (total participation)** από την πλευρά της Buyer Queue, καθώς κάθε εγγραφή της μπορεί να σχετίζεται είτε με καθορισμένη παράσταση είτε μόνο με εισιτήριο.

Επίσης, στο Entity Relationship διάγραμμα, κάθε πίνακας διαθέτει ως PRIMARY KEY ένα μοναδικό ID το οποίο έχει οριστεί ως AUTO INCREMENT. Με αυτόν τον τρόπο, εξασφαλίζεται η μοναδικότητα κάθε εγγραφής μέσα στον πίνακα, επιτυγχάνεται βέλτιστη δεικτοδότηση (indexing) και απλοποιείται σημαντικά η υλοποίηση των σχέσεων τύπου JOIN μεταξύ των πινάκων.

Α.2. Σχεσιακό Διάγραμμα και υλοποίηση Βάσης Δεδομένων (Relational Model)

Στη συνέχεια πραγματοποιείται ο μετασχηματισμός του Entity Relationship διαγράμματος σε σχεσιακό μοντέλο (Relational Model) με στόχο τη δημιουργία των αντίστοιχων πινάκων της βάσης δεδομένων.

Κάθε οντότητα και σχέση του ER αποδίδεται με πίνακες, πεδία και συσχετίσεις που αντανακλούν τη δομή και τους κανόνες της βάσης. Το τελικό σχεσιακό διάγραμμα παρουσιάζεται στην Εικόνα 2:



Εικόνα 2

Κατά τη μετατροπή του Entity Relationship διαγράμματος σε σχεσιακό μοντέλο, οι σχέσεις **1:1** αποδίδονται με την προσθήκη του πρωτεύοντος κλειδιού (primary key) του ενός πίνακα, ως ξένο κλειδί (foreign key) στον άλλον πίνακα. Η επιλογή του πίνακα που θα περιλαμβάνει το ξένο κλειδί εξαρτάται από τη συμμετοχή των οντοτήτων στη σχέση, δηλαδή το total participation, ή από σχεδιαστική απλότητα. Ομοίως, οι σχέσεις **1:N** αποδίδονται με την προσθήκη του πρωτεύοντος κλειδιού (primary key) του πίνακα στην πλευρά 1, ως ξένο κλειδί (foreign key) στον πίνακα της πλευράς N. Τέλος, για τις σχέσεις **N:N** δημιουργείται ένας νέος πίνακας, ο οποίος περιλαμβάνει τα πρωτεύοντα κλειδιά (primary keys) των δυο σχετιζόμενων πινάκων ως ξένα κλειδιά (foreign keys). Ο νέος πίνακας ορίζεται με σύνθετο πρωτεύον κλειδί (composite primary key), το οποίο αποτελείται από τα δύο αυτά attributes, εξασφαλίζοντας τη μοναδικότητα κάθε συσχέτισης.

Stored Procedures

match_tickets():

Η αποθηκευμένη διαδικασία match_tickets() έχει σκοπό να πραγματοποιεί αυτόματα το ταίριασμα εισιτηρίων μεταξύ πωλητών και αγοραστών, με βάση δύο διαφορετικά κριτήρια: **τον μοναδικό αριθμό EAN** του εισιτηρίου ή **την κατηγορία εισιτηρίου σε συγκεκριμένο event**. Πρωτού ξεκινήσουμε την υλοποίηση της λογικής της ουράς μεταπώλησης εκτελούμε την εντολή SET SQL_SAFE_UPDATES = 0, η οποία απενεργοποιεί την safe update mode στη MySQL που αποτρέπει την εκτέλεση εντολών UPDATE ή DELETE χωρίς χρήση WHERE ή LIMIT. Επιπρόσθετα, όπως διαπιστώθηκε, δεν κλήθηκε η συνάρτηση αυτή μέσα σε κάποιο από τα αντίστοιχα triggers (π.χ. στο trigger BEFORE INSERT στον πίνακα Buyer_Queue) καθώς η MySQL δεν επιτρέπει σε αυτά να εκτελούν UPDATE ή DELETE στον ίδιο πίνακα, γιατί αυτό μπορεί να προκαλέσει αστάθεια, recursion ή conflicts με την εντολή που εκτελείται εκείνη τη στιγμή. Τέτοιες ενέργειες δημιουργούν ασάφεια, καθώς η γραμμή δεν έχει ακόμη εισαχθεί πλήρως. Έπειτα την ενεργοποιούμε ξανά στο τέλος (SET SQL_SAFE_UPDATES = 1;) για να επαναφέρουμε την ασφαλή συμπεριφορά στο υπόλοιπο σύστημα.
Σε αυτή τη stored procedure, εκτελούνται πολλές εντολές UPDATE που ενημερώνουν γραμμές στους πίνακες Seller_Queue, Buyer_Queue και ticket, βασιζόμενες σε JOIN με προσωρινά tables, δηλαδή CTE's (EAN_Matches, Category_Matches), ένα για κάθε κριτήριο, για να αποθηκεύσει τα αντίστοιχα matches. Στην πρώτη φάση, αναζητούνται αντιστοιχίες EAN – δηλαδή αγοραστές που έχουν δηλώσει συγκεκριμένο EAN, το οποίο ταιριάζει με διαθέσιμα εισιτήρια πωλητών. Αυτές οι αντιστοιχίες θεωρούνται πιο αυστηρές και γίνονται κατά προτεραιότητα, με σειρά βάσει της ημερομηνίας εισαγωγής τους στις ουρές. Στη δεύτερη φάση, εάν κάποιος πωλητής δεν βρέθηκε στην πρώτη φάση, ελέγχεται αν υπάρχουν αγοραστές που ενδιαφέρονται για την ίδια κατηγορία και event, χωρίς να έχουν δηλώσει EAN — και έτσι δημιουργούνται τα Category matches. Αφού βρεθούν τα ζευγάρια, ενημερώνονται οι πίνακες ώστε τόσο ο αγοραστής όσο και ο πωλητής να χαρακτηριστούν ως "processed" (για να μην συμμετάσχουν ξανά σε νέα matching procese), και ενημερώνονται τα στοιχεία του εισιτηρίου με τα προσωπικά στοιχεία του νέου κατόχου, δηλαδή του buyer που βρέθηκε. Τέλος, άξιο αναφοράς είναι το γεγονός πως με μία κλήση της διαδικασίας αυτής, βρίσκονται όλα τα ζευγάρια που ικανοποιούν τα προαναφερθέντα κριτήρια κατά το runtime. Αυτή η διαδικασία αυτοματοποιεί και εξασφαλίζει τη δίκαιη και σωστή ανάθεση εισιτηρίων σε ενδιαφερόμενους αγοραστές, σεβόμενη τη σειρά FIFO.

Α.2.1. Απαραίτητοι Περιορισμοί για ορθότητα

Constraints

Καταρχάς, όσον αφορά τους περιορισμούς ακεραιότητας (Primary Key Constraints) και τους περιορισμούς αναφορικής ακεραιότητας (Foreign Keys) παρουσιάζονται ευκρινώς στο παραπάνω Relational Model. Έπειτα, όσον αφορά την ακεραιότητα πεδίου τιμών(CHECK constraints) και τους περιορισμούς οριζόμενοι από τον χρήστη ορίσαμε τους παρακάτω περιορισμούς:

- CHECK (start_date < end_date):
  Το φεστιβάλ πρέπει να ξεκινά πριν την ημερομηνία λήξης του.
- CHECK (maximum_capacity > 0):
  Η χωρητικότητα του κτιρίου πρέπει να είναι θετικός αριθμός.
- CHECK (category IN ('technical', 'security', 'support')):
  Επιτρεπόμενες κατηγορίες προσωπικού.
- CHECK (age >= 18 AND age <= 65):
  Το προσωπικό πρέπει να έχει ηλικία από 18 έως 65 ετών.
- CHECK (experience IN ('beginner', 'intermediate', 'experienced', 'very experienced', 'professional')):
  Επιτρεπόμενα επίπεδα εμπειρίας προσωπικού.

- CHECK (event_start_time < event_end_time AND TIME(event_start_time) >= '17:00:00' AND TIME(event_end_time) <= '23:00:00'):
  Οι εκδηλώσεις πρέπει να πραγματοποιούνται αυστηρά μεταξύ 17:00 και 23:00 και να τελειώνουν αργότερα από την ώρα έναρξης.
- CHECK (performance_type IN ('Warm up', 'Headline', 'Guest', 'Special Appearance')):
  Επιτρεπόμενοι τύποι εμφάνισης.
- CHECK (performance_duration >= 15 AND performance_duration <= 180):
  Η διάρκεια εμφάνισης πρέπει να είναι από 15 έως 180 λεπτά.
- CHECK ((artist_id IS NULL AND band_id IS NOT NULL) OR (artist_id IS NOT NULL AND band_id IS NULL)):
  Κάθε εμφάνιση πρέπει να έχει **είτε καλλιτέχνη είτε συγκρότημα**, όχι και τα δύο.
- CHECK (artist_performance_grade IN ('Very Unsatisfied', 'Unsatisfied', 'Neutral', 'Satisfied', 'Very Satisfied')):
  Αξιολόγηση απόδοσης καλλιτέχνη από τον επισκέπτη.
- CHECK (lighting_sound_grade IN ('Very Unsatisfied', 'Unsatisfied', 'Neutral', 'Satisfied', 'Very Satisfied')):
  Αξιολόγηση ήχου/φωτισμού.
- CHECK (stage_presence_grade IN ('Very Unsatisfied', 'Unsatisfied', 'Neutral', 'Satisfied', 'Very Satisfied')):
  Αξιολόγηση παρουσίας στη σκηνή.
- CHECK (organization_grade IN ('Very Unsatisfied', 'Unsatisfied', 'Neutral', 'Satisfied', 'Very Satisfied')):
  Αξιολόγηση διοργάνωσης.
- CHECK (final_impression_grade IN ('Very Unsatisfied', 'Unsatisfied', 'Neutral', 'Satisfied', 'Very Satisfied')):
  Συνολική τελική εντύπωση από τον επισκέπτη.
- CHECK (holder_email REGEXP '^[a-zA-Z0-9]+@[a-zA-Z0-9]+\\.[a-z]{2,}$'):
  Το email του κατόχου εισιτηρίου πρέπει να έχει έγκυρη μορφή
- CHECK (holder_age >= 16 AND holder_age <= 85):
  Ηλικιακό όριο για κατόχους εισιτηρίου: 16–85 ετών.
- CHECK (category IN ('VIP', 'General Entrance', 'Backstage')):
  Επιτρεπόμενες κατηγορίες εισιτηρίου.
- CHECK (ticket_price > 0):
  Η τιμή εισιτηρίου πρέπει να είναι θετική.
- CHECK (payment_method IN ('Credit Card', 'Debit Card', 'Bank Transfer')):
  Επιτρεπόμενοι τρόποι πληρωμής.
- CHECK (age >= 16 AND age <= 85):
  Ηλικιακό όριο για επισκέπτες: 16–85 ετών.
- CHECK ((EAN IS NOT NULL AND (category IS NULL AND event_id IS NULL))):
  Ορίζει ότι το εισιτήριο πρέπει να έχει **είτε μοναδικό EAN**, είτε **(event_id + κατηγορία)** αλλά όχι και τα δύο.

Triggers

Τα triggers στην SQL είναι ειδικά είδη stored procedures τα οποία ενεργοποιούνται αυτόματα ως απόκριση σε κάποιες προκαθορισμένες από τον σχεδιαστή μεταβολές. Στη δική μας βάση, στο αρχείο procedures.sql βρίσκονται τα εξής triggers:

- *event_festival_check:*

Το παραπάνω trigger εκτελείται **πριν** από κάθε εισαγωγή νέου record στον πίνακα event και διασφαλίζει δύο βασικούς κανόνες:

1. Η ημερομηνία και ώρα της νέας εκδήλωσης εμπίπτει στο χρονικό διάστημα του αντίστοιχου φεστιβάλ (βάσει start_date – end_date).
2. Δεν υπάρχει άλλη εκδήλωση στην ίδια σκηνή (building) με επικαλυπτόμενο ωράριο. Αν κάποια από αυτές τις συνθήκες δεν ικανοποιείται, διακόπτει την εισαγωγή με κατάλληλο σφάλμα.

- *performance_check:*

Το trigger αυτό ενεργοποιείται **πριν** την εισαγωγή κάθε νέου performance και ελέγχει συνοπτικά τα εξής:

1. Η προγραμματισμένη ώρα και διάρκεια του performance εμπίπτει εντός του χρονικού πλαισίου του αντίστοιχου event στο οποίο ανήκει.
2. Δεν αλληλεπικαλύπτεται με άλλες εμφανίσεις την ίδια ώρα.
3. Έπειτα, εφαρμόζει ξεχωριστούς κανόνες για σόλο καλλιτέχνες και γκρουπ: διασφαλίζει ότι οι καλλιτέχνες είναι στην κατάλληλη ηλικία, δεν έχουν ταυτόχρονη υποχρέωση ως σόλο ή μέλη συγκροτήματος, και δεν έχουν ήδη τρεις συνεχόμενες χρονιές συμμετοχής.
4. Τέλος, επιβάλλει διαλείμματα 5–30 λεπτών πριν και μετά την εμφάνισή τους, αποτρέποντας πολύ μικρά ή μεγάλα κενά ανάμεσα σε διαδοχικές εμφανίσεις.

- *staff_capacity_check:*

Το trigger αυτό εκτελείται πριν τη διαγραφή οποιουδήποτε μέλους του προσωπικού και φροντίζει ώστε, μετά την αφαίρεσή του:

1. Να μην παραβιάζονται οι ελάχιστες αναλογίες ασφαλείας και βοηθητικού προσωπικού που απαιτούνται από τη χωρητικότητα του κτιρίου. Συγκεκριμένα, μετρά πόσα άτομα της ίδιας κατηγορίας («security» ή «support») υπάρχουν στο ίδιο κτίριο και συγκρίνει αυτή την τιμή, μείον το προς διαγραφή άτομο, με το 5% ή 2% της μέγιστης χωρητικότητας αντίστοιχα. Αν η διαγραφή πρόκειται να ρίξει τον αριθμό κάτω από αυτά τα όρια, ακυρώνει τη διαγραφή και επιστρέφει σφάλμα, διασφαλίζοντας έτσι ότι πάντα υπάρχει τουλάχιστον το ελάχιστο ζωτικό προσωπικό σε κάθε κτίριο.

- *ticket_check:*

Αυτό το trigger εκτελείται πριν από κάθε εισαγωγή νέου εισιτηρίου και φροντίζει να τηρούνται οι επιχειρησιακοί κανόνες πώλησης. Δηλαδή:

1. Ελέγχει ότι τα VIP εισιτήρια δεν ξεπερνούν το 10 % της χωρητικότητας της αίθουσας.
2. Διασφαλίζει ότι κάθε επισκέπτης αγοράζει το πολύ ένα εισιτήριο για την ίδια παράσταση την ίδια μέρα.
3. Όταν η συνολική πληρότητα του αντίστοιχου event φτάσει τη μέγιστη χωρητικότητα, εισάγει ένα νέο record στο table των Buyer_queue με τα στοιχεία του ενδιαφερόμενου αγοραστή και έπειτα καλεί τη διαδικασία match_tickets() για να βρει αν υπάρχει εισιτήριο σε διαθεσιμότητα. Έτσι, αποφεύγονται υπερπωλήσεις και διασφαλίζεται η σωστή λειτουργία του συστήματος εισιτηρίων.

- *seller_before_insert:*

Αυτό το trigger «τρέχει» πριν εισαχθεί μια νέα εγγραφή στην ουρά πωλητών και:

1. Φροντίζει πρώτα να εμποδίσει την μεταπώληση ενός εισιτηρίου που έχει ήδη σκαναριστεί στο φεστιβάλ, πετώντας σφάλμα αν το εισιτήριο είναι ήδη χρησιμοποιημένο.
2. Ελέγχει εάν πρόκειται να πουληθεί εισιτήριο για παράσταση που ανήκει στο παρελθόν.

3. Αν το εισιτήριο δεν έχει χρησιμοποιηθεί και αφορά μελλοντική παράσταση, ορίζει αυτόματα την ημερομηνία πώλησης στο τρέχον timestamp και ελέγχει αν ήδη υπάρχει στη σειρά κάποιος αγοραστής για την ίδια παράσταση και κατηγορία ή για συγκεκριμένο EAN. Αν βρεθεί άμεσα διαθέσιμος αγοραστής, σηματοδοτεί την τρέχουσα εγγραφή πωλητή ως «processed», ώστε στην επόμενη φάση —με ένα AFTER INSERT trigger—να ολοκληρωθεί η αντιστοίχιση, να σημειωθεί και ο αγοραστής ως "processed" και να ανανεωθούν τα στοιχεία του εισιτηρίου.

- *seller_after_insert:*

Αυτό το trigger εκτελείται μετά την εισαγωγή ενός νέου πωλητή στην ουρά. Αν η εγγραφή του πωλητή είχε ήδη επισημανθεί ως processed = TRUE, δηλαδή βρέθηκε match, τότε:

1. Αναζητά τον παλαιότερο αγοραστή για τον οποίο ισχύει processed = FALSE με την ίδια παράσταση (event_id) και κατηγορία, δηλαδή τον αγοραστή που προηγουμένως είχε βρεθεί match.
2. Επισημαίνει αυτόν τον αγοραστή ως επεξεργασμένο (processed = TRUE).
3. Ενημερώνει τον πίνακα ticket για το εισιτήριο του πωλητή, αναθέτοντάς του τα στοιχεία του αγοραστή (όνομα, επώνυμο, email, τηλέφωνο, ηλικία), επαναφέρει το scanned = FALSE και βάζει τρέχουσα ημερομηνία αγοράς.

Με άλλα λόγια, μόλις εισαχθεί ένας πωλητής και υπάρχει διαθέσιμος αγοραστής, ολοκληρώνεται αυτόματα η πώληση και μεταβιβάζονται οι πληροφορίες στο αντίστοιχο εισιτήριο.

- *buyer_before_insert:*

Αυτό το trigger «τρέχει» πριν εισαχθεί μια νέα εγγραφή στην ουρά αγοραστών (Buyer_Queue) και φροντίζει αυτόματα:

1. Να ελέγξει ότι ο συγκεκριμένος buyer δεν πάει να εισαχθεί στην ουρά μεταπώλησης για να βρει εισιτήριο που αφορά παράσταση που ανήκει στο παρελθόν
2. Να επιλέξει, αφού εγγυηθεί την ακεραιότητα του, αν υπάρχει, τον παλαιότερο διαθέσιμο πωλητή (Seller_Queue) με την ίδια παράσταση και κατηγορία εισιτηρίου ή με συγκεκριμένο EAN. Ορίζει την τρέχουσα ημερομηνία/ώρα αγοράς στο πεδίο buy_date και, εάν βρεθεί άμεσο ταίρι πωλητή (δηλαδή κάποιο μη επεξεργασμένο αίτημα πώλησης για το ίδιο event και κατηγορία), σηματοδοτεί την εγγραφή του αγοραστή ως "processed", προετοιμάζοντας έτσι την επόμενη φάση για την αντιστοίχιση και την ενημέρωση των στοιχείων του εισιτηρίου.

- *buyer_after_insert:*

Αυτό το trigger εκτελείται αμέσως μετά την εισαγωγή μιας νέας εγγραφής στην ουρά αγοραστών (Buyer_Queue). Αν η εγγραφή έχει ήδη ταιριάξει με κάποιον πωλητή (processed = TRUE), τότε εντοπίζει την αντίστοιχη εγγραφή πωλητή που δεν έχει ακόμη επεξεργαστεί, τη σημειώνει κι αυτή ως "processed" και στη συνέχεια ενημερώνει την εγγραφή του εισιτηρίου (ticket) με τα στοιχεία του νέου κατόχου (όνομα, επώνυμο, email, τηλέφωνο, ηλικία), ορίζει το εισιτήριο ως μη σκαναρισμένο (scanned = FALSE) και καταγράφει την τρέχουσα ημερομηνία αγοράς. Έτσι ολοκληρώνεται αυτόματα η μεταβίβαση του εισιτηρίου από τον πωλητή στον αγοραστή.

Χρησιμοποιούμε **Delimiter** για να αλλάξουμε το σύμβολο που χρησιμοποιείται για τον τερματισμό των εντολών SQL. Κανονικά, το MySQL χρησιμοποιεί το **;** ως τερματιστικό, αλλά όταν δημιουργούμε triggers ή stored procedures με πολλές εντολές, πρέπει να χρησιμοποιήσουμε διαφορετικό delimiter για να αποφεύγουμε τη σύγκρουση. Για παράδειγμα, μπορούμε να ορίσουμε το DELIMITER // πριν από τον κώδικα του trigger και να επαναφέρουμε το delimiter σε **;** μόλις ολοκληρωθεί η δημιουργία του trigger. Έτσι, μπορούμε να συμπεριλάβουμε πολλές εντολές μέσα σε ένα trigger χωρίς να υπάρξει πρόβλημα με την ερμηνεία των εντολών από τον MySQL server.

A.2.2. Ορισμός Ευρετηρίων (Indexing)

Ένα index (ή ευρετήριο) είναι μια ειδική δομή δεδομένων που δημιουργείται σε έναν ή περισσότερους στήλες (πεδία) ενός πίνακα, με σκοπό να επιταχύνει την αναζήτηση και την προσπέλαση των δεδομένων. Μάλιστα, τα indexes μας είναι πολύ χρήσιμα καθώς επιταχύνουν τα SELECT ερωτήματα με WHERE, JOIN, ORDER BY, GROUP BY ειδικά όταν τα εφαρμόζουμε σε attributes τα οποία ζητάμε συχνά. Επίσης, τα indexes είναι πολύ χρήσιμα καθώς χωρίς αυτά η βάση πρέπει να κάνει πλήρη σάρωση για να βρει δεδομένα, κάτι που γίνεται εξαιρετικά αργό σε πίνακες με μεγάλο όγκο εγγραφών. Οπότε για αυτούς τους λόγους θεωρήσαμε σκόπιμο να συμπεριλάβουμε και στη δική μας εργασία για την υλοποιήση μιας Βάσης Δεδομένων για το φεστιβάλ, ορισμένα στοχευμένα indexes για να βελτιώσουμε την απόδοση συγκεκριμένων ερωτημάτων ανάλυσης (queries) που αφορούν συχνά φιλτραρίσματα, ομαδοποιήσεις και ταξινομήσεις. Πιο συγκεκριμένα, να τονιστεί ότι η MySQL η οποία χρησιμοποιήσαμε, έχει ως default engine Inno/DB και φτιάχνει αυτόματα index σε ολά τα PRIMARY KEYs και σε όλα τα FOREIGN KEYs， γεγονός το οποίο είναι πολύ σημαντικό για την υλοποίηση μας , καθώς έτσι δεν χρειάζεται να φτιάξουμε εμείς indexes σε αυτά τα κλειδία τα οποία παίζουν πολύ σημαντικό ρόλο για τη συσχέτιση των πινάκων ειδικά στα JOINs. Επομένως, τα indexes που χρησιμοποιήσαμε και η χρησιμότητα τους για τα queries παρουσιάζονται παρακάτω:

1. **CREATE INDEX idx_contintent ON location (continent);**

Στο ερώτημα 13 (Βρείτε τους καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους.) αυτό το ευρετήριο μας είναι πολύ χρήσιμο καθώς κάνουμε αναζήτηση των καλλιτεχνών βάσει των ξεχωριστών ηπείρων που έχουν επισκεφτεί και του πλήθους αυτών.(COUNT(DISTINCT l.continent) >= 3)

2. **CREATE INDEX idx_festival_year ON festival (festival_year);**

Το συγκεκριμένο ευρετήριο είναι πολύ σημαντικό καθώς σε πολλά ερωτήματα μας ζητείται να αναζητήσουμε , να ομαδοποιήσουμε ή να ξεχωρίσουμε επισκέπτες ή εμφανίσεις ή καλλιτέχνες ή μουσικά είδη βάσει της χρονιάς διεξαγωγής του φεστιβάλ. Οπότε με το να έχουμε ένα index στο festival_year επιταχύνουμε τις πράξεις αναζήτησης βάσει της χρονιάς διεξαγωγής του φεστιβάλ. Ειδικότερα , στο ερώτημα 1 αναζητούμε τα έσοδα ανά έτος οπότε θα χρειαστεί να κάνουμε GROUP BY festival,year. Στο ερώτημα 2, αναζητούμε για συγκεκριμένη χρονιά άρα θέλουμε πρόσβαση στη στήλη festival_year χωρίς να χρειάζεται να διατρέξουμε όλο τον πίνακα (WHERE f.festival_year = 2024). Στο ερώτημα 3, αναζητούμε καλλιτέχνες του ίδιου φεστιβάλ άρα θα χρειαστεί να κάνουμε ομαδοποίηση κατά χρονιά φεστιβάλ αφού το φεστιβάλ είναι ετήσιο. Επίσης, στο ερώτημα 14 χρειάζεται να υπολογίζουμε συνεχόμενα 2 χρόνια οπότε θέλουμε να έχουμε ταχύτατα τις χρονιές κάθε φεστιβάλ (t2.festival_year = t1.festival_year + 1) .

3. **CREATE INDEX idx_staff_category ON staff (category);**

Στο ερώτημα 7 μας ζητείται να βρούμε το φεστιβάλ που είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού οπότε ένα ευρετήριο στη στήλη category του staff επιταχύνει το φίλτρο WHERE category = 'technical', καθώς εντοπίζει γρήγορα μόνο τα άτομα που ανήκουν στην κατηγορία technical, χωρίς να σαρώνει ολόκληρο τον πίνακα staff. Στο ερώτημα 8 πάλι επιταχύνει το φίλτρο WHERE category = 'support', καθώς εντοπίζει γρήγορα μόνο τα άτομα που ανήκουν στην κατηγορία support, χωρίς να σαρώνει ολόκληρο τον πίνακα staff. Στο ερώτημα 12 θέλουμε να παρέχουμε ανάλυση ανά κατηγορία οπότε επιταχύνεται το GROUP BY.

4. **CREATE INDEX idx_staff_experience ON staff (experience);**

Στο ερώτημα 7 μας ζητείται να βρούμε το φεστιβάλ που είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού, οπότε ένα index staff_expereince μας επιταχύνει το aggregate function CONCAT(AVG(CASE s.experience...)) όπου υπολογίζουμε τη μέση εμπειρία του τεχνικού προσωπικού σε ένα φεστιβάλ, καθώς χρειαζόμαστε άμεσα το staff_experience attribute από τον staff πίνακα.

5. **CREATE INDEX idx_genre_name ON genre (genre_name);**

Στο ερώτημα 2 εκτελούμε μια αναζήτηση στους καλλιτέχνες βάσει του φίλτρου WHERE genre_name = 'Jazz', οπότε το index βοηθά να βρεθεί αμέσως το genre_id που αντιστοιχεί στο 'Jazz' χωρίς να σαρώνεται ολόκληρος ο πίνακας genre. Στο ερώτημα 10 μας ζητείται να βρούμε κοινά ζεύγη μουσικών ειδών σε καλλιτέχνες όπως θα φανεί στη συνέχεια ώστε να μας κάνει πιο εύκολη την αναζήτηση. Οπότε το ευρετήριο αυτό μας είναι πολύ χρήσιμο καθώς στο τέλος βάσει αυτού γίνεται η ομαδοποίηση. Τέλος, στο ερώτημα 14 επίσης φαίνεται η χρησιμότητα του συγκεκριμένου ευρετηρίου καθώς πολλά φίλτρα σύγκρισης και ταξινόμησης αναφέρονται στο συγκεκριμένο κλειδί .(ON t1.genre_name = t2.genre_name, ORDER BY t1.genre_name)

6. **CREATE INDEX idx_artist_age ON artist (date_of_birth);**

Αυτό το ευρετήριο το δημιουργήσαμε συγκεκριμένα για το ερώτημα 5 όπου πρέπει να βρούμε καταρχάς καλλιτέχνες υπό τη συνθήκη ότι είναι νέοι δηλαδή έχουν ηλικία κάτω των 30 ετών , οπότε αυτό το ευρετήριο κάνει πιο αποδοτική την αναζήτηση της ημερομηνίας γέννησης του καλλιτέχνη άρα και τη συνάρτηση ηλικίας του καλλιτέχνη:

TIMESTAMPDIFF(YEAR, a.date_of_birth, CURDATE()) AS age.

7. **CREATE INDEX idx_performance_type ON performance (performance_type);**

Αυτό το index είναι κατάλληλο για το ερώτημα 3 όπου αναζητούμε καλλιτέχνες οι οποίοι έχουν συμμετάσχει σε εμφανίσεις συγκεκριμένου τύπου ειδικότερα warm-up, οπότε το index επιταχύνει τη φιλτραρισμένη αναζήτηση στις εγγραφές του πίνακα performance, έτσι ώστε να εξεταστούν μόνο οι εγγραφές με τύπο Warm up.(WHERE p.performance_type = 'Warm up')

8. **CREATE INDEX idx_artist_performance_grade ON review (artist_performance_grade);**

Αυτό το ευρετήριο μας είναι χρήσιμο στο ερώτημα 4 καθώς μας κάνει πιο αποδοτικό το aggregation function AVG(CASE r.artist_performance_grade...) για να υπολογίζουμε αποδοτικά τον μέσο όρο αξιολογήσεων των καλλιτεχνών. Επίσης, αυτό το ευρετήριο μας είναι χρήσιμο και για το ερώτημα 15 καθώς πρέπει να βρούμε τις συνολικές artist_performance_grade για έναν καλλιτέχνη από τους επισκέπτες, οπότε το ευρετήριο κάνει πιο αποδοτικές τις ομαδοποιήσεις που πραγματοποιούνται.

9. **CREATE INDEX idx_payment_method ON ticket (payment_method);**

Αυτό το ευρετήριο μας είναι χρήσιμο για το ερώτημα 1 καθώς πρέπει να παρέχουμε ανάλυση των εσόδων ανά είδος πληρωμής, οπότε το index στο payment_method βοηθά στην ταχύτερη ομαδοποίηση και φιλτράρισμα ανά τρόπο πληρωμής.(GROUP BY f.festival_year, t.payment_method)

10. **CREATE INDEX idx_purchase_date ON ticket (purchase_date);**

Ο δείκτης στην ημερομηνία αγοράς εισιτηρίου επιταχύνει κάθε query που φιλτράρει ή ομαδοποιεί βάσει της ημερομηνίας αγοράς. Πιο συγκεκριμένα στο ερώτημα 1, που σχετίζεται με τα έσόδα ανά έτος φεστιβάλ, ο optimizer μπορεί να χρησιμοποιήσει τον purchase_date index για να σκανάρει μόνο τα tickets που αγοράστηκαν μέσα στην εκάστοτε χρονιά, μειώνοντας το πλήθος των γραμμών που πρέπει να σκαναριστούν.

11. **CREATE INDEX idx_scanned ON ticket (scanned);**

Κάθε φορά που φιλτράρουμε τα “επαληθευμένα” (scanned=true) ή μη-επαληθευμένα εισιτήρια, αυτός ο index μας εξασφαλίζει γρήγορη πρόσβαση. Στον μηχανισμό match_tickets() και στο view scanned_events_per_visitor, ελέγχουμε WHERE t.scanned = TRUE πρωτού δώσουμε το δικαίωμα στο χρήστη να βαθμολογήσει. Ο idx_scanned κάνει αυτό το lookup σχεδόν ακαριαία, αποφεύγοντας full-table scans στον πίνακα ticket.

12. CREATE INDEX idx_sell_date ON Seller_Queue (sell_date);
Στην stored procedure "match_tickets()" που ταιριάζει sellers με buyers, οι sellers παίρνουν σειρά με βάση την ημερομηνία πώλησης (sell_date). Εφόσον στη διαδικασία αυτή κάνουμε ORDER BY sq.sell_date, ο δείκτης idx_sell_date επιτρέπει στη MySQL να διαβάσει τις εγγραφές ήδη ταξινομημένες, χωρίς επιπλέον sort στη μνήμη, κάνοντας πιο γρήγορη όλη τη διαδικασία matching.

13. CREATE INDEX idx_seller_processed ON Seller_Queue (processed);
Πριν ο αλγόριθμος match_tickets επιλέξει υποψήφιους πωλητές, φιλτράρει με WHERE sq.processed = FALSE. Ο δείκτης πάνω στη στήλη processed σημαίνει ότι βρίσκουμε άμεσα όλες τις μη-επεξεργασμένες εγγραφές, χωρίς να κάνουμε full scan, και αυτό μετράει πολύ σε μεγάλους πίνακες ουρών.

14. CREATE INDEX idx_buy_date ON Buyer_Queue (buy_date);
Αντίστοιχα, οι αγοραστές και αυτοί μπαίνουν σε ουρά κατά buy_date. Το idx_buy_date επιτρέπει γρήγορο ταξινόμημα και ανάκτηση των πιο παλαιών ή πρόσφατων αγοραστών, μειώνοντας τη πολυπλοκότητα του ORDER BY bq.buy_date μέσα στη διαδικασία match_tickets.

15. CREATE INDEX idx_buyer_processed ON Buyer_Queue (processed);
Η επιλογή αγοραστών που δεν έχουν πάει ακόμη (bq.processed = FALSE) γίνεται με αυτόν τον index. Διασφαλίζει ότι κάθε αναζήτηση non-processed buyers γίνεται με cost O(log N) αντί για O(N), επιταχύνοντας σημαντικά τη συνολική ροή του matching.

Όπως φαίνεται από τη παρούσα ανάλυση χρησιμοποιήσαμε στοχευμένα indexes για συγκεκριμένα μόνο πεδία που συμμετέχουν συχνά και με "στρατηγικό" τρόπο ώστε να εξυπηρετούν τις ανάγκες των queries μας και αποφύγαμε την αλόγιστη χρήση τους καθώς ,παρότι επιταχύνουν την ανάγνωση, επιβαρύνουν τις εγγραφές και τις ενημερώσεις (INSERT, UPDATE, DELETE), καθώς κάθε αλλαγή απαιτεί και την αντίστοιχη ενημέρωση του index. Επίσης, καταναλώνουν πρόσθετο αποθηκευτικό χώρο.

A.3 Views

Στην εργασία μας, εκτός από τον ορισμό στοχευμένων ευρετηρίων (indexes), υλοποιήσαμε και μια σειρά από views, τα οποία αποτελούν virtual tables που εξυπηρετούν την απλοποίηση των ερωτημάτωνκαι την επεξεργασία των δεδομένων. Τα views μας συγκεντρώνουν δεδομένα από πολλούς πίνακες μέσω JOINs και GROUP BY, και χρησιμοποιούνται για τη διευκόλυνση της εκτέλεσης σύνθετων ερωτημάτων. Μάλιστα , τα ευρετήρια που χρησιμοποιήσαμε είναι άμεσα συνδεδεμένα με με τη λειτουργία αυτών των views καθώς επιταχύνουν τις JOIN, WHERE, GROUP BY και ORDER BY εντολές που περιέχονται στο κάθε view. Τελικά, βελτιώνουν την απόδοση των queries που βασίζονται στα views, καθώς επιτρέπουν στη βάση να προσπελάσει μόνο τα σχετικά δεδομένα, αποφεύγοντας πλήρεις σαρώσεις πινάκων. Οπότε παρακάτω θα παρουσιάσουμε τα views τα οποία υλοποιήσαμε και θα επεξηγήσουμε και τη λειτουργία τους καθώς είναι απαραίτητα για τη κατανόηση των queries μας, όπου θα αναδειχτεί και η χρήση τους.(οι αναλυτικοί κώδικες των views είναι στο αντίστοιχο sql script, εδώ παρέχουμε μόνο συνοπτική επεξήγηση της λειτουργίας τους και των εντολών που χρησιμοποιούν):

1. event_staff_view

Το event_staff_view είναι ένα virtual table που επιστρέφει για κάθε event του φεστιβάλ το event_id, την ημερομηνία διεξαγωγής του (DATE(e.event_start_time) AS event_date), την κατηγορία (category) του προσωπικού που έχει ανατεθεί σε αυτό, το id του υπαλλήλου (staff_id) καθώς και το ονοματεπώνυμό του (CONCAT(s.first_name, ' ', s.last_name) AS staff_name).Το view δημιουργείται

με INNER JOIN μεταξύ των πινάκων event, building και staff. Ειδικότερα, βασιζόμαστε στην παραδοχή που έχουμε κάνει ότι κάθε event γίνεται σε ένα συγκεκριμένο κτίριο (building_id), και κάθε μέλος προσωπικού εργάζεται σε κάποιο από αυτά τα κτίρια (s.building_id = b.building_id). Έτσι, μέσω αυτής της σχέσης, μπορούμε να συσχετίσουμε κάθε event με όλο το προσωπικό που εργάζεται στο κτίριο όπου αυτό πραγματοποιείται. Άρα το view μας επιτρέπει να βλέπουμε εύκολα ποιο προσωπικό έχει προγραμματιστεί να εργάζεται σε κάθε event, σε ποια ημερομηνία και σε ποια κατηγορία (ανάλογα με την κατηγορία του). Το παραπάνω view χρησιμοποιείται στο ερώτημα 8.

2. artist_total_participations_view

Το artist_total_participations_view είναι ένα virtual table το οποίο επιστρέφει τα artist_id, το ονοματεπώνυμο, τη τρέχουσα ηλικία του καλλιτέχνη(TIMESTAMPDIFF(YEAR, a.date_of_birth, CURDATE()) AS age) και τον συνολικό αριθμό των συμμετοχών-εμφανίσεων (performances) που έχει ο καλλιτέχνης στο φεστιβάλ, μόνο για τα φεστιβάλ που έχουν ολοκληρωθεί (δηλαδή f.end_date <= CURDATE()). Ειδικότερα, όπως γνωρίζουμε, ο καλλιτέχνης μπορεί να συμμετέχει, δηλαδή να εμφανιστεί στο φεστιβάλ είτε solo είτε ως μέλος ενός συγκροτήματος. Οπότε στο πρώτο κομμάτι του view βρίσκουμε τις performances του ίδιου του καλλιτέχνη (INNER JOIN performance AS p ON a.artist_id = p.artist_id) και τις συνδέουμε με τα event και festival ώστε να περιοριστούμε μόνο στα φεστιβάλ που έχουν ολοκληρωθεί. Στο δεύτερο μέρος βρίσκουμε τις εμφανίσεις του κάθε καλλιτέχνη ως μέλος συγκροτήματος (INNER JOIN artist_band AS ab ON a.artist_id = ab.artist_id INNER JOIN performance AS p ON ab.band_id = p.band_id) και τις συνδέουμε με τα event και festival για να κρατήσουμε και εδώ μόνο τα φεστιβάλ που έχουν ολοκληρωθεί. Στη συνέχεια, ενώνουμε τις solo και band εμφανίσεις με UNION ALL χωρίς να αφαιρέσουμε διπλές εγγραφές, ώστε να μετρηθούν όλες και να έχουμε τον πλήρη αριθμό εμφανίσεων του καλλιτέχνη, είτε ως solo είτε ως μέλος συγκροτήματος. Τέλος, ομαδοποιούμε τα αποτελέσματα (GROUP BY artist_id, artist_name, age), μετράμε τις συνολικές εμφανίσεις κάθε καλλιτέχνη με COUNT(*) και τις κατατάσσουμε κατά φθίνουσα σειρά πλήθους εμφανίσεων με ORDER BY total_performances DESC. Το παραπάνω ερώτημα χρησιμοποιείται στα ερωτήματα 5 ,11.

3. genre_total_performances_year_view

Το genre_total_performances_year_view είναι ένας εικονικός πίνακας (virtual table) ο οποίος επιστρέφει για κάθε μουσικό είδος (genre_name) και για κάθε έτος φεστιβάλ (festival_year) τον συνολικό αριθμό εμφανίσεων (performances) που σχετίζονται με το συγκεκριμένο είδος, μόνο για φεστιβάλ που έχουν ολοκληρωθεί (δηλαδή f.end_date <= CURDATE()).Ειδικότερα, επειδή ένα μουσικό είδος μπορεί να συσχετίζεται με καλλιτέχνες (solo artists) ή με συγκροτήματα (bands), το view είναι χωρισμένο σε δύο μέρη. Στο πρώτο μέρος βρίσκονται οι εμφανίσεις των solo καλλιτεχνών. Συγκεκριμένα, συνδέουμε τους πίνακες performance,artist, artist_genre και genre, ώστε να βρούμε σε κάθε genre name ποια performances αντιστοιχούν(FROM performance AS p INNER JOIN artist AS a ON p.artist_id = a.artist_id,INNER JOIN artist_genre AS ag ON a.artist_id = ag.artist_id INNER JOIN genre g ON ag.genre_id = g.genre_id) και μέσω της performance συνδέουμε με event και festival, ώστε να καταγράψουμε τις εμφανίσεις του κάθε μουσικού είδους σε φεστιβάλ που έχουν πραγματοποιηθεί(INNER JOIN event e ON p.event_id = e.event_id INNER JOIN festival f ON e.festival_id = f.festival_id,WHERE f.end_date <= CURDATE()). Στο δεύτερο μέρος με ανάλογο τρόπο με το πρώτο μέρος βρίσκουμε τις band εμφανίσεις που αντιστοιχούν στο κάθε μουσικό είδος για κάθε χρόνο για τα χρόνια που έχει πραγματοποιηθεί το φεστιβάλ.Τα δύο σύνολα αποτελεσμάτων ενώνονται με UNION ALL, ώστε να περιλαμβάνονται όλες οι εμφανίσεις (είτε μέσω καλλιτέχνη είτε

μέσω συγκροτήματος) χωρίς να αφαιρούνται διπλές εγγραφές, καθώς ένα μουσικό είδος μπορεί να έχει εμφανιστεί και από solo performance και από band performance και έτσι να μπορούμε να υπολογίσουμε με ακρίβεια το πλήθος εμφανίσεων ανά είδος και έτος. Τέλος, ομαδοποιούμε ανά μουσικό είδος και έτος διεξαγωγής του φεστιβάλ (GROUP BY genre_name, festival_year) και χρησιμοποιούμε COUNT(*) για να μετρήσουμε τις συνολικές εμφανίσεις κάθε μουσικού είδους σε κάθε έτος. Το αποτέλεσμα ταξινομείται βάσει του της χρονιάς και του μουσικού είδους με ORDER BY festival_year, genre_name. Το παραπάνω view χρησιμοποιείται στο ερώτημα 14.

4. visitor_festival_attended_view

Το view visitor_festival_attended_view υπολογίζει για κάθε επισκέπτη τον αριθμό παραστάσεων που παρακολούθησε σε κάθε φεστιβάλ ανά έτος, λαμβάνοντας υπόψη μόνο τις περιπτώσεις όπου έχει παρακολουθήσει περισσότερες από 3 παραστάσεις (δηλαδή όπου scanned = 1, που σημαίνει ότι ο επισκέπτης παραβρέθηκε). Πιο συγκεκριμένα, για κάθε event_id από τον πίνακα event, εντοπίστηκαν τα αντίστοιχα ticket_id μέσω JOIN με τον πίνακα ticket, και στη συνέχεια προσδιορίστηκε το festival_year με επιπλέον JOIN στον πίνακα festival. Το παραπάνω view χρησιμοποιείται στο ερώτημα 9.

A.4 Fake Data

Τα fake data κατασκευάστηκαν στο script "**fake_data.py**" το οποίο είναι γραμμένο σε Python. Στο συγκεκριμένο script παράγουμε τυχαία δεδομένα για κάθε πίνακα του σχεσιακού μοντέλου (Relational Model), τα οποία αποθηκεύονται στη μεταβλητή data με τη μορφή εισαγωγής δεδομένων στη βάση, δηλαδή με τη μορφή:

INSERT INTO table_name (attributes)

Οι εγγραφές αυτές αποθηκεύονται στη συνέχεια στο SQL αρχείο "load.sql", ώστε να μπορούν να φορτωθούν εύκολα στη βάση.

Για την δημιουργία τυχαίων και ψευδών δεδομένων, χρησιμοποιήθηκαν οι βιβλιοθήκες faker και random της Python. Σε περιπτώσεις όπου δεν υπήρχε υποστήριξη από τη faker για συγκεκριμένους τύπους δεδομένων (π.χ. τεχνικός εξοπλισμός κτηρίων), δημιουργήσαμε custom data pools, τα οποία αξιοποιήθηκαν με for loops και βοηθητικές συναρτήσεις για την παραγωγή έγκυρων INSERT statements ανά οντότητα.

Τα δεδομένα που δημιουργήθηκαν, αν και είναι τυχαία, ικανοποιούν όλους του περιορισμούς που υπαγορεύονται από την εκφώνηση, καθώς και τις πρόσθετες παραδοχές και σχεδιαστικούς κανόνες που ορίσαμε κατά την υλοποίηση της βάσης.

Στη συνέχεια, θα αναλύσουμε τα δεδομένα που δημιουργήσαμε στη βάση μας.

Αρχικά, εισάγουμε **12** διαφορετικές τοποθεσίες (**locations**), τις οποίες επιλέγουμε από ένα location pool με 24 location infos, και **12** διαφορετικά φεστιβάλ (**festival**) από το 2016 μέχρι το 2027, δηλαδή προσθέσαμε 2 μελλοντικές χρονιές, και υποθέτουμε ότι κάθε χρονιά του φεστιβάλ διεξάγεται σε διαφορετική τοποθεσία. Έπειτα, εισάγουμε **30** μουσικές σκηνές/κτήρια (**buildings**) και για κάθε μια από αυτές εισάγουμε προσωπικό ασφαλείας, υποστήριξης και τεχνικό (**staff**) βάσει των δοθέντων περιορισμών για το πλήθος τους.

Υποθέτουμε ότι κάθε ημέρα καθενός φεστιβάλ ξεκινάει στις 17:00 και τελειώνει στις 23:00 και διεξάγεται σε ένα συγκεκριμένο building. Για κάθε ημέρα, προσθέτουμε παραστάσεις (**events**) οι οποίες λαμβάνουν χώρα μέσα στο χρονικό εύρος διεξαγωγής του φεστιβάλ και στο αντίστοιχο building που διεξάγεται.

Έπειτα, στους πίνακες **genre**, **subgenres** προσθέτουμε διάφορα μουσικά είδη και υποείδη, αντίστοιχα, από ένα music_genres_subgenres pool που δημιουργήσαμε. Μετά, εισάγουμε **35** artists, **15** bands και μέσω των tables **artist_genre, artist_subgenre, band_genre, band_subgenre**, αντιστοιχίζουμε κάθε καλλιτέχνη και μπάντα με ορισμένα μουσικά είδη και τα αντίστοιχα μουσικά υποείδη. Κάθε καλλιτέχνης και κάθε μπάντα μπορεί να ανήκει σε 1 ή 2 μουσικά είδη και για καθένα να έχει 1 έως 3

υποείδη. Επίσης, για κάθε καλλιτέχνη και μπάντα έχουν προστεθεί ρεαλιστικά image urls και descriptions γεγονός που καθιστά ακόμα πιο ρεαλιστικά τα δεδομένα της βάσης μας.
Ο πίνακας **artist_band** αντιστοιχεί καλλιτέχνες με συγκροτήματα με τυχαίο τρόπο, υποθέτοντας ότι ένα συγκρότημα μπορεί να έχει από 2 έως 6 μέλη και ότι ένας καλλιτέχνης μπορεί να ανήκει σε περισσότερα από ένα συγκροτήματα.
Για να δημιουργήσουμε τις εμφανίσεις (**performances**) κάθε event υλοποιήθηκαν κατάλληλες συναρτήσεις

has_more_than_3_consecutive_years, assign_artist_or_band_with_member_check

οι οποίες ελέγχουν ότι ένας καλλιτέχνης (συγκρότημα) δεν μπορεί να εμφανίζεται σε δύο σκηνές ταυτόχρονα και δεν επιτρέπεται η συμμετοχή του στο φεστιβάλ για περισσότερα από 3 συνεχή έτη.
Με τη βοήθεια των παραπάνω συναρτήσεων, δημιουργήσαμε τα κατάλληλα δεδομένα στον πίνακα **performances**, γεμίζοντας με εμφανίσεις τις παραστάσεις κάθε φεστιβάλ. Θεωρούμε ότι κάθε εμφάνιση έχει διάρκεια από 15 έως 180 λεπτά, ενώ ανάμεσα σε διαδοχικές εμφανίσεις προβλέπεται υποχρεωτικά διάλειμμα ελάχιστης διάρκειας 5 λεπτών και μέγιστης 30 λεπτών, όπως υπαγορεύεται από την εκφώνηση.
Στη συνέχεια, προσθέσαμε **400** επισκέπτες του φεστιβάλ στον πίνακα **visitors** οι οποίοι κατέχουν ένα ή περισσότερα εισιτήρια, ανεξάρτητα από το αν τελικά παρακολουθήσουν τις παραστάσεις.
Για κάθε επισκέπτη δημιουργήσαμε εγγραφές στον πίνακα **tickets**, φροντίζοντας κάθε εισιτήριο να αντιστοιχεί σε συγκεκριμένη παράσταση. Για εισιτήρια που αφορούν παρελθοντικές παραστάσεις, δώσαμε 70% πιθανότητα να είναι "scanned", δηλαδή να έχει γίνει είσοδος του επισκέπτη. Επιπλέον, εξασφαλίσαμε το πλήθος των εισιτηρίων για κάθε παράσταση να μην υπερβαίνει τη χωρητικότητα (max_capacity) του αντίστοιχου κτιρίου.
Παράλληλα, προσθέσαμε δεδομένα στον πίνακα **visitor_event**, ο οποίος αντιστοιχεί κάθε επισκέπτη με τις παραστάσεις για τις οποίες έχει εισιτήριο, υποθέτοντας ότι κάθε επισκέπτης μπορεί να έχει εισιτήρια για 1 έως 25 παραστάσεις.
Ακολούθως, δημιουργήσαμε δεδομένα για τον πίνακα **review**, ο οποίος περιέχει αξιολογήσεις επισκεπτών για εμφανίσεις (performances) που έχουν παρακολουθήσει, δηλαδή για τις οποίες έχουν σκαναρισμένο εισιτήριο για τη σχετική παράσταση.
Στη συνέχεια, συμπληρώσαμε την **ουρά μεταπώλησης (seller_queue)**, με εισιτήρια που δεν έχουν σκαναριστεί και των οποίων η ημέρα αγοράς (purchase date) προηγείται της ημέρας της παράστασης. Η ουρά λειτουργεί με FIFO (First In, First Out) λογική, και τα δεδομένα είναι ταξινομημένα κατά ημερομηνία εισαγωγής.
Αντίστοιχα, δημιουργήσαμε εγγραφές στην **buyer_queue**, που αφορά αγοραστές οι οποίοι ενδιαφέρονται για sold-out παραστάσεις. Κάθε αγοραστής δηλώνει ενδιαφέρον είτε για κάποιο συγκεκριμένο εισιτήριο είτε για κάποια συγκεκριμένη παράσταση και κατηγορία. Η ουρά αγοραστών είναι επίσης FIFO, με τις εγγραφές ταξινομημένες κατά ημερομηνία εκδήλωσης ενδιαφέροντος.
Εν κατακλείδι, δημιουργήσαμε δεδομένα για όλες τις οντότητες και σχέσεις του μοντέλου, τηρώντας όλους τους λειτουργικούς, λογικούς και επιχειρησιακούς περιορισμούς που προκύπτουν είτε από την εκφώνηση, είτε από τις επιλογές σχεδιασμού που ορίσαμε εμείς.

B. Queries

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

Υλοποιούμε το παραπάνω ερώτημα τρέχοντας το παρακάτω query στο MySQLWorkbench:

```
SELECT
  f.festival_year,
  CONCAT(FORMAT(SUM(CASE WHEN t.payment_method = 'Credit Card' THEN t.ticket_price END), 2), '$') AS credit_card,
  CONCAT(FORMAT(SUM(CASE WHEN t.payment_method = 'Debit Card' THEN t.ticket_price END), 2), '$') AS debit_card,
  CONCAT(FORMAT(SUM(CASE WHEN t.payment_method = 'Bank Transfer' THEN t.ticket_price END), 2), '$') AS e_banking,
  CONCAT(FORMAT(SUM(t.ticket_price), 2), '$') AS total_earnings
FROM ticket t
JOIN event e ON t.event_id = e.event_id
JOIN festival f ON e.festival_id = f.festival_id
GROUP BY f.festival_year
ORDER BY f.festival_year DESC;
```

Αρχικά κάνουμε JOIN από ticket → event → festival για να συνδυάσουμε τα κέρδη με τη χρονιά του φεστιβάλ , και ομαδοποιούμε ανά festival_year. Στο SELECT, χρησιμοποιούμε SUM(CASE WHEN … THEN ticket_price END) για να αθροίσουμε μόνο τα ποσά της κάθε μεθόδου πληρωμής. Κάθε άθροισμα περνάει από FORMAT(...,2) για να δείξει δύο δεκαδικά καθώς πρόκειται για κέρδη σε $. Έπειτα έπειτα προσθέτουμε το $ με CONCAT(...,'$') ώστε να είναι εμφανές το νόμισμα. Για τα συνολικά έσοδα, αθροίζουμε όλα τα ticket_price. Τέλος, με ORDER BY … DESC εμφανίζουμε πρώτα τα πιο πρόσφατα φεστιβάλ.

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

Έστω ότι το συγκεκριμένο μουσικό είδος είναι η Jazz και το συγκεκριμένο έτος του φεστιβάλ είναι το 2024.
Υλοποιούμε το παραπάνω ερώτημα τρέχοντας το παρακάτω query στο MySQLWorkbench:

```sql
SELECT
    a.artist_id,
    a.artist_first_name, a.artist_last_name,
    'Jazz' AS genre_name,
    CASE
        WHEN EXISTS (
            SELECT 1
            FROM performance p
            JOIN event e ON p.event_id = e.event_id
            JOIN festival f ON e.festival_id = f.festival_id
            WHERE f.festival_year = 2024
              AND (
                  p.artist_id = a.artist_id
                  OR p.band_id IN (SELECT ab.band_id
                                   FROM artist_band AS ab
                                   WHERE ab.artist_id = a.artist_id)
              )
        )
        THEN 'YES'
        ELSE 'NO'
    END AS participated_in_2024
FROM artist a
JOIN artist_genre ag ON a.artist_id = ag.artist_id
JOIN genre g ON ag.genre_id = g.genre_id
WHERE g.genre_name = 'Jazz';
```

Το παραπάνω query μας επιστρέφει τα ids όλων των καλλιτεχνών που ανήκουν στο συγκεκριμένο μουσικό είδος "Jazz", μαζί με το ονοματεπώνυμό τους, το μουσικό τους είδος ("Jazz"), καθώς και μία ένδειξη ("YES" ή "NO") που δηλώνει αν συμμετείχαν σε φεστιβάλ το έτος 2024. Στο τμήμα FROM ... JOIN ... JOIN ... WHERE γίνεται η επιλογή όλων των καλλιτεχνών που ανήκουν στο μουσικό είδος Jazz. Στη συνέχεια, με χρήση της εντολής CASE και του EXISTS, ελέγχεται αν υπάρχει τουλάχιστον μία εμφάνιση του καλλιτέχνη είτε σόλο είτε ως μέλος ενός συγκροτήματος σε κάποιο event που ανήκει σε φεστιβάλ του 2024. Πιο συγκεκριμένα, το SELECT 1 επιστρέφει απλώς το 1 αν βρει τουλάχιστον μία τέτοια γραμμή, χωρίς να επιστρέφει άλλα δεδομένα. Είναι μια βελτιστοποίηση που κάνει το ερώτημα πιο αποδοτικό, αποφεύγοντας την επιστροφή περιττών στηλών. Με βάση αυτό, υπολογίζεται η κατάλληλη ένδειξη για κάθε καλλιτέχνη.

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

Υλοποιούμε το παραπάνω ερώτημα τρέχοντας το παρακάτω query στο MySQLWorkbench:

```sql
SELECT
  combined.artist_id,
  combined.festival_id,
  COUNT(*) AS warmup_count
FROM (
  -- solo
  SELECT
    p.artist_id,
    e.festival_id
  FROM performance AS p
  JOIN event AS e ON p.event_id = e.event_id
  WHERE p.performance_type = 'Warm Up' AND p.artist_id IS NOT NULL
  UNION ALL
  -- As part of band
  SELECT
    ab.artist_id,
    e.festival_id
  FROM performance AS p
  JOIN artist_band AS ab ON p.band_id = ab.band_id
  JOIN event AS e ON p.event_id = e.event_id
  WHERE p.performance_type = 'Warm Up' AND p.band_id IS NOT NULL
) AS combined
GROUP BY combined.artist_id, combined.festival_id
HAVING COUNT(*) > 2;
```

Με αυτό το query υπολογίζουμε πόσες φορές ένας καλλιτέχνης συμμετείχε σε Warm Up εμφανίσεις στο ίδιο φεστιβάλ, είτε solo είτε ως μέλος μπάντας, και κρατά μόνο όσους έχουν πάνω από 2 συμμετοχές. Πιο συγκεκριμένα, για κάθε artist_id και festival_id βρίσκουμε solo εμφανίσεις warm up από τον πίνακα performance (για το artist_id) JOIN με τον πίνακα event (για το festival_id) ως προς το κοινό τους event_id. Έπειτα, για κάθε artist_id και festival_id βρίσκουμε, ομοίως με προηγουμένως, τις εμφανίσεις του ως μέλος μπάντας, μέσω του table artist_band, από τον πίνακα performance (για το band_id), JOIN με τον πίνακα artist_band (για το artist _id) ως προς το κοινό τους band_id και JOIN με τον πίνακα event (για το festival_id) ως προς το κοινό τους event_id. Τέλος, ενώνουμε με UNION ALL τα δυο tables που προκύπτουν και από τον πίνακα combined που προκύπτει κάνουμε group by combined.artist_id, combined.festival_id και μετράμε τα ζεύγη artist_id, festival_id για να βρούμε αυτά που εμφανίζονται περισσότερες από 2 φορές.

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

Υλοποιούμε το παραπάνω ερώτημα τρέχοντας το παρακάτω query στο MySQLWorkbench:

```
SELECT a.artist_id, a.artist_first_name, a.artist_last_name,
    AVG (
        CASE r.artist_performance_grade
            WHEN 'Very Unsatisfied' THEN 1
            WHEN 'Unsatisfied' THEN 2
            WHEN 'Neutral' THEN 3
            WHEN 'Satisfied' THEN 4
            WHEN 'Very Satisfied' THEN 5
        END
    ) AS 'Average Artist Performance Grade',
    AVG (
        CASE r.final_impression_grade
            WHEN 'Very Unsatisfied' THEN 1
            WHEN 'Unsatisfied' THEN 2
            WHEN 'Neutral' THEN 3
            WHEN 'Satisfied' THEN 4
            WHEN 'Very Satisfied' THEN 5
        END
    ) AS 'Average Final Impression Grade'
FROM review r
JOIN performance p ON r.performance_id = p.performance_id
JOIN artist a ON a.artist_id = p.artist_id
WHERE a.artist_id = 10;
```

Στην έξοδο βλέπουμε για τον καλλιτέχνη με artist_id = 10 το όνομά του, το επώνυμό του, καθώς και το μέσο όρο αξιολογήσεων των ερμηνειών του μαζί με το μέσο όρο αξιολόγησης της τελικής εντύπωσης κάθε επισκέπτη. Στο ερώτημα αυτό παραθέτουμε και εναλλακτικό Query Plan με τη χρήση του FORCE INDEX. Αρχικά τροποποιούμε το παραπάνω μας κώδικα με τη προσθήκη της εντολής EXPLAIN, με την οποία λαμβάνουμε το πλάνο εκτέλεσης που θα ακολουθήσει η MySQL για το συγκεκριμένο SELECT. Αναλυτικότερα μας ενημερώνει όσον αφορά:

1. **id:** Αύξων αριθμός του επιμέρους μέρους του query. Αν έχουμε subqueries ή UNION, θα δούμε πολλαπλά ids.
2. **select_type:** SIMPLE → απλό query χωρίς subqueries
   PRIMARY → το κύριο query
   SUBQUERY → υποερώτημα σε WHERE clause
   DEPENDENT SUBQUERY → υποερώτημα που εξαρτάται από εξωτερικές τιμές
3. **table:** Ποιος πίνακας (στη προκειμένη περίπτωση alias) διαβάζεται.
4. **partitions:** Σε ποιο partition (αν υπάρχει) θα κοιτάξει.
5. **type:** ALL → full table scan, το οποίο προφανώς είναι μη επιθυμητό
   index → scan μόνο του index
   range → χρησιμοποίηση εύρους σε ευρετήριο
   ref → αναζήτηση με index όταν υπάρχουν συγκρίσεις με ισότητα
   eq_ref → ιδανικό για PK/FK joins
   const/system → σταθερός χρόνος
6. **possible_keys:** Ποια indexes θα μπορούσαν να χρησιμοποιηθούν με βάση τα φιλτραρίσματά μας.
7. **key:** Ποιο index επέλεξε τελικά ο optimizer.
8. **key_len:** Μήκος σε bytes του κομματιού του index που χρησιμοποίηθηκε.
9. **ref:** Ποιο πεδίο συγκρίνεται με το index

10. **rows:** Εκτίμηση αριθμού γραμμών που διαβάστηκαν για να βρεθεί το αποτέλεσμα
11. **filtered:** Εκτίμηση ποσοστού γραμμών που περνούν τα φίλτρα ανάγνωσης
12. **Extra:** Πρόσθετες πληροφορίες βελτιστοποίησης:
    Using index → cover query (δε χρειάζεται ανάγνωση του table, μόνο του index)
    Using where → εφαρμογή WHERE μετά από index/range scan
    Using temporary → δημιουργία προσωρινού πίνακα (π.χ. για GROUP BY)
    Using filesort → εξωτερική ταξινόμηση (ORDER BY χωρίς κατάλληλο index)

```sql
EXPLAIN
SELECT a.artist_id, a.artist_first_name, a.artist_last_name,
    AVG (
        CASE r.artist_performance_grade
            WHEN 'Very Unsatisfied' THEN 1
            WHEN 'Unsatisfied' THEN 2
            WHEN 'Neutral' THEN 3
            WHEN 'Satisfied' THEN 4
            WHEN 'Very Satisfied' THEN 5
        END
    ) AS 'Average Artist Performance Grade',
    AVG (
        CASE r.final_impression_grade
            WHEN 'Very Unsatisfied' THEN 1
            WHEN 'Unsatisfied' THEN 2
            WHEN 'Neutral' THEN 3
            WHEN 'Satisfied' THEN 4
            WHEN 'Very Satisfied' THEN 5
        END
    ) AS 'Average Final Impression Grade'
FROM review r
JOIN performance p ON r.performance_id = p.performance_id
JOIN artist a ON a.artist_id = p.artist_id
WHERE a.artist_id = 10;
```

Εικόνα 4.2:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	p	NULL	ref	PRIMARY,artist_id	artist_id	5	const	7	100.00	Using index
1	SIMPLE	r	NULL	ref	performance_id	performance_id	4	festival_db.p.performance_id	1	100.00	NULL

Στο πλάνο εκτέλεσης βλέπουμε ότι η ερώτηση ξεκινάει από τον πίνακα artist κι επειδή φιλτράρει με το πρωτεύον κλειδί (artist_id = σταθερή τιμή), η MySQL τον διαβάζει σαν const, δηλαδή κάνει απευθείας lookup μίας μόνο γραμμής μέσω του PK INDEX. Έπειτα ενώνει τον πίνακα performance χρησιμοποιώντας το πεδίο του artist_id, οπότε το access type είναι ref—η τιμή δεν είναι literal αλλά προέρχεται από τον προηγούμενο πίνακα—και έτσι χρησιμοποιείται ο μη-μοναδικός index για να βρεθούν οι εμφανίσεις του καλλιτέχνη. Τέλος, ενώνει τον πίνακα review (r) με access type ref στον δείκτη performance_id, αφού το performance_id πάλι έρχεται από τον πίνακα performance και όχι ως σταθερή τιμή. Οι εκτιμώμενες rows είναι ελάχιστες σε κάθε βήμα (1 ή 7), δε χρειάζεται filesort ή προσωρινός πίνακας, και βλέπουμε ότι στο δεύτερο join εμφανίζεται “Using index” στο Extra, δηλαδή η MySQL ικανοποιεί το query μόνο από τον δείκτη χωρίς να διαβάσει τα υπόλοιπα πεδία του πίνακα—μια ένδειξη αποτελεσματικής χρήσης των ευρετηρίων.

Πάμε λοιπόν τώρα να εφαρμόσουμε εναλλακτικό Query Plan με χρήση FORCE INDEX:

```
EXPLAIN
SELECT
  a.artist_id,
  a.artist_first_name,
  a.artist_last_name,
  AVG(
    CASE r.artist_performance_grade
      WHEN 'Very Unsatisfied' THEN 1
      WHEN 'Unsatisfied'      THEN 2
      WHEN 'Neutral'          THEN 3
      WHEN 'Satisfied'        THEN 4
      WHEN 'Very Satisfied'   THEN 5
    END
  ) AS `Average Artist Performance Grade`,
  AVG(
    CASE r.final_impression_grade
      WHEN 'Very Unsatisfied' THEN 1
      WHEN 'Unsatisfied'      THEN 2
      WHEN 'Neutral'          THEN 3
      WHEN 'Satisfied'        THEN 4
      WHEN 'Very Satisfied'   THEN 5
    END
  ) AS `Average Final Impression Grade`
FROM review AS r FORCE INDEX (PRIMARY)
JOIN performance AS p FORCE INDEX (PRIMARY)
  ON r.performance_id = p.performance_id
JOIN artist AS a FORCE INDEX (PRIMARY)
  ON a.artist_id = p.artist_id
WHERE a.artist_id = 10;
```

Εικόνα 4.2:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	r	NULL	ALL	NULL	NULL	NULL	NULL	3075	100.00	NULL
1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY	4	festival_db.r.performance_id	1	5.00	Using where

Εικόνα 4.2:

Μετά το FORCE INDEX, η MySQL συνέχισε να χρησιμοποιεί το PRIMARY KEY ως INDEX για τον πίνακα artist με τύπο αναζήτησης const, αλλά για τον πίνακα review αναγκάστηκε να κάνει full table scan αντί να αξιοποιήσει το ευρετήριο στο πεδίο performance_id, καθώς το PRIMARY δεν εξυπηρετεί αποτελεσματικά το JOIN. Αυτό οδήγησε σε χαμηλότερη απόδοση (σκάνναρε 3075 rows έναντι των 7 που είχαμε βρει προηγούμενως), δείχνοντας πόσο προσεκτικά πρέπει να εφαρμόζονται τα index hints: ένα λάθος hint μπορεί να ανατρέψει τη χρήση του βέλτιστου ευρετηρίου και να επιβαρύνει τη διαδικασία.

Πάμε τώρα να αναλύσουμε και αυτό το ερώτημα με τη προσθήκη των εντολών:

SET PROFILING = 1: Ενεργοποιεί τον εσωτερικό profiler της MySQL, ώστε κάθε επόμενη δήλωση να καταγράφεται με τον χρόνο εκτέλεσής της.

SHOW PROFILES: Μετά την εκτέλεση του query, εμφανίζει το χρόνο που χρειάστηκε για να τρέξουν σε δευτερόλεπτα.

Έτσι λοιπόν έχουμε τους εξής χρόνους πριν και μετά τη προσθήκη FORCE INDEX:

- Χωρίς FORCE INDEX:

29	0.00331900	SELECT	a.artist_id, a.…

- Με FORCE INDEX:

32	0.01454400	SELECT	a.artist_id, a.…

Παρατηρούμε πως ο χρόνος αυξάνεται κατά πολύ με τη χρήση FORCE INDEX, το οποίο είναι αναμενόμενο καθώς η MySQL χρησιμοποιεί καλύτερο indexing (με τα default indexes στα PKs και FKs) από ότι το δικό μας.

Ας δοκιμάσουμε να χρησιμοποιήσουμε και άλλες διαφορετικές στρατηγικές JOIN:

- Nested Loop JOIN:

Το Nested Loop JOIN είναι ο default τρόπος υλοποίησης JOIN στη MySQL και η απλούστερη μέθοδος σύνδεσης δύο πινάκων: για κάθε γραμμή του «εξωτερικού» πίνακα επαναλαμβάνει σάρωση στον «εσωτερικό» πίνακα και ελέγχει αν ικανοποιείται η συνθήκη join. Αν βρει ταίρι, παράγει αποτέλεσμα, και συνεχίζει μέχρι να εξαντληθούν όλες οι γραμμές. Είναι εύκολο στον υλοποίηση, αλλά μπορεί να γίνει αργό όταν οι πίνακες είναι μεγάλοι, εκτός αν υποστηρίζεται από κατάλληλους δείκτες (indexes) για γρήγορη εύρεση στο εσωτερικό loop. Παρακάτω βλέπουμε το χρόνο που απαιτήθηκε για να τρέξει το query με στρατηγική Nested Loop JOIN που απαντάει στο ερώτημα 4 (χωρίς FORCE INDEX):

29	0.00331900	SELECT	a.artist_id, a.…

- Hash JOIN:

Το Hash JOIN είναι μια μέθοδος σύνδεσης δύο πινάκων όπου πρώτα «χτίζουμε» έναν γρήγορο πίνακα κατακερματισμού (hash table) με βάση τις τιμές join ενός από τους πίνακες (συνήθως του μικρότερου), και μετά κάνουμε lookup σε αυτόν για κάθε γραμμή του άλλου πίνακα. Έτσι αποφεύγονται οι επαναλαμβανόμενες σάρωσεις, κάνοντας το join πολύ αποδοτικό όταν δεν υπάρχουν κατάλληλοι δείκτες ή όταν τα μεγέθη των πινάκων είναι μεγάλα. Για να την ενεργοποιήσουμε κάνουμε χρήση της εντολής **‘SET SESSION optimizer_switch = 'hash_join = on, block_nested_loop = off, batched_key_access = off’;** Παρακάτω βλέπουμε το χρόνο που απαιτήθηκε για να τρέξει το query με στρατηγική Hash JOIN που απαντάει στο ερώτημα 4 (χωρίς FORCE INDEX):

55	0.01022400	SELECT	a.artist_id, a.…

Παρατηρούμε πως η εκτέλεση του query με την προεπιλεγμένη στρατηγική Nested-Loop JOIN χρειάστηκε περίπου 0,0033 s, ενώ με Hash JOIN (Query_ID 55) ο χρόνος αυξήθηκε σε 0,0102 s. Δηλαδή, στο συγκεκριμένο σενάριο το Nested-Loop ήταν σχεδόν τρεις φορές ταχύτερο, καθώς οι δείκτες επέτρεψαν γρήγορη αναζήτηση στο εσωτερικό loop. Σημειώνουμε επίσης ότι η MySQL δεν υποστηρίζει αλγόριθμο Merge JOIN, επομένως τα παραπάνω είναι οι μοναδικές διαθέσιμες εναλλακτικές στρατηγικές.

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

Υλοποιούμε το παραπάνω ερώτημα τρέχοντας το παρακάτω query στο MySQLWorkbench:

```
SELECT
    artist_id,
    artist_name,
    age,
    total_performances
FROM artist_total_participations_view
WHERE age < 30
ORDER BY total_performances DESC
```

Το ερώτημα αυτό επιστρέφει τα artist_id, το ονοματεπώνυμο, την ηλικία και τον συνολικό αριθμό εμφανίσεων (performances) για νέους καλλιτέχνες ηλικίας κάτω των 30 ετών, οι οποίοι έχουν συμμετάσχει στα περισσότερα φεστιβάλ έως και την τρέχουσα ημερομηνία, είτε ως σόλο καλλιτέχνες είτε ως μέλη συγκροτήματος. Το παραπάνω query χρησιμοποιεί το view artist_total_participations_view, το οποίο επιστρέφει για κάθε καλλιτέχνη το artist_id, το ονοματεπώνυμο, την ηλικία και τον συνολικό αριθμό εμφανίσεων σε φεστιβάλ που έχουν ήδη ολοκληρωθεί (δηλαδή f.end_date <= CURDATE()). Με βάση αυτό το view, το query φιλτράρει μόνο τους καλλιτέχνες που είναι κάτω των 30 ετών (age < 30 όπου age η τρέχουσα ηλικία) και επιστρέφει τα στοιχεία τους μαζί με τον αριθμό συμμετοχών τους, ταξινομημένα κατά φθίνουσα σειρά εμφανίσεων (ORDER BY total_performances DESC), ώστε να εμφανίζονται πρώτοι οι καλλιτέχνες με τις περισσότερες συμμετοχές-εμφανίσεις σε φεστιβάλ.

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

```
SELECT
  r.visitor_id,
  e.event_id,
  e.event_name,
  AVG(
    (
      (CASE r.artist_performance_grade
         WHEN 'Very Unsatisfied' THEN 1
         WHEN 'Unsatisfied'      THEN 2
         WHEN 'Neutral'          THEN 3
         WHEN 'Satisfied'        THEN 4
         WHEN 'Very Satisfied'   THEN 5
       END)
    + (CASE r.lighting_sound_grade
         WHEN 'Very Unsatisfied' THEN 1
         WHEN 'Unsatisfied'      THEN 2
         WHEN 'Neutral'          THEN 3
         WHEN 'Satisfied'        THEN 4
         WHEN 'Very Satisfied'   THEN 5
       END)
```

```
    + (CASE r.stage_presence_grade
          WHEN 'Very Unsatisfied' THEN 1
          WHEN 'Unsatisfied'      THEN 2
          WHEN 'Neutral'          THEN 3
          WHEN 'Satisfied'        THEN 4
          WHEN 'Very Satisfied'   THEN 5
       END)
    + (CASE r.organization_grade
          WHEN 'Very Unsatisfied' THEN 1
          WHEN 'Unsatisfied'      THEN 2
          WHEN 'Neutral'          THEN 3
          WHEN 'Satisfied'        THEN 4
          WHEN 'Very Satisfied'   THEN 5
       END)
    + (CASE r.final_impression_grade
          WHEN 'Very Unsatisfied' THEN 1
          WHEN 'Unsatisfied'      THEN 2
          WHEN 'Neutral'          THEN 3
          WHEN 'Satisfied'        THEN 4
          WHEN 'Very Satisfied'   THEN 5
       END)
    ) / 5.0
  ) AS avg_event_score
FROM review r FORCE INDEX (visitor_id)
JOIN performance AS p FORCE INDEX (PRIMARY)
    ON r.performance_id = p.performance_id
JOIN event AS e FORCE INDEX (PRIMARY)
    ON p.event_id       = e.event_id
WHERE r.visitor_id = 9
GROUP BY
  r.visitor_id,
  e.event_id,
  e.event_name
ORDER BY
  e.event_id;
```

Αυτό το ερώτημα επιστρέφει για κάθε εκδήλωση (event) που έχει παρακολουθήσει ο επισκέπτης με visitor_id = 9 έναν ενιαίο δείκτη ικανοποίησης: πρώτα συνδέει τους πίνακες review → performance → event, μετά με CASE μετατρέπει τις πέντε βαθμολογίες (ερμηνεία καλλιτέχνη, φωτισμός/ήχος, σκηνική παρουσία, οργάνωση, τελική εντύπωση) από μορφή VARCHAR σε ακέραιες τιμές 1–5, τις αθροίζει και διαιρεί με 5.0 ώστε να προκύψει ένα σκορ 1–5 ανά αξιολόγηση, και τέλος παίρνει το AVG(...) όλων αυτών των μεμονωμένων σκορ για κάθε παράσταση. Το GROUP BY r.visitor_id, e.event_id, e.event_name εξασφαλίζει μία γραμμή ανά εκδήλωση και το ORDER BY e.event_id ταξινομεί το τελικό αποτέλεσμα, βάσει του αναγνωριστικού της παράστασης.

Έπειτα χρησιμοποιούμε τα εξής FORCE INDEX:

- Στο review r: FORCE INDEX(visitor_id) το οποίο επιταχύνει το WHERE r.visitor_id = 10, καθώς ο optimizer δεν θα κάνει full-table scan αλλά θα πάει απευθείας στις γραμμές με αυτό το visitor_id.
- Στο performance p: FORCE INDEX(PRIMARY), καθώς απαιτούμε ταχύ lookup στο performance_id.
- Στο event e: FORCE INDEX(PRIMARY), καθώς απαιτούμε ταχύ lookup στο event_id.

Θα αναλύσουμε την απόδοση του query εξετάζοντας το χρόνο που απαιτήθηκε κατά την εκτέλεση του, με την χρήση των profiles:

- Χωρίς FORCE INDEX:

- Με FORCE INDEX:

Παρατηρούμε λοιπόν ότι χωρίς κανένα FORCE INDEX το ερώτημα εκτελείται σε περίπου 0,00283 s, καθώς ο optimizer επιλέγει αυτόματα τους πιο επιλεκτικούς δείκτες (composite ή FK/PK indexes), ενώ όταν εξαναγκάζουμε τη χρήση συγκεκριμένων indexes το κόστος ανεβαίνει σε περίπου 0,00360 s, διότι χάνει το πλεονέκτημα των default multi-column indexes και σαρώνει περισσότερες γραμμές. Συμπερασματικά, η MySQL συχνά διαλέγει καλύτερα τους δείκτες, οπότε τα FORCE INDEX πρέπει να εφαρμόζονται μόνο αν αποδεδειγμένα διορθώνουν suboptimal επιλογές του optimizer.

Έπειτα, δοκιμάζουμε και άλλες διαφορετικές στρατηγικές JOIN:

- Nested Loop JOIN:

Παρακάτω βλέπουμε το χρόνο που απαιτήθηκε για να τρέξει το query με στρατηγική Nested Loop JOIN που απαντάει στο ερώτημα 6 (χωρίς FORCE INDEX):

- Hash JOIN:

Για να την ενεργοποιήσουμε κάνουμε χρήση της εντολής **'SET SESSION optimizer_switch = 'hash_join = on, block_nested_loop = off, batched_key_access = off';** Παρακάτω βλέπουμε το χρόνο που απαιτήθηκε για να τρέξει το query με στρατηγική Hash JOIN που απαντάει στο ερώτημα 6 (χωρίς FORCE INDEX):

Παρατηρούμε πως, όταν τρέχουμε το ερώτημα με την προεπιλεγμένη στρατηγική **Nested-Loop Join**, ο συνολικός χρόνος ήταν **0,002834s**, ενώ με την ενεργοποίηση του **Hash Join** αυξήθηκε σε **0,013202s**. Δηλαδή το Hash Join αποδείχτηκε περίπου **4,7 φορές πιο αργό** στο συγκεκριμένο σενάριο, επιβεβαιώνοντας ότι για μικρό σχετικά πλήθος γραμμών και με κατάλληλους δείκτες το απλό Nested-Loop παραμένει η ταχύτερη επιλογή.

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

```sql
SELECT f.festival_year,
  CONCAT(AVG(CASE s.experience
        WHEN 'beginner'         THEN 1
        WHEN 'intermediate'     THEN 2
        WHEN 'experienced'      THEN 3
        WHEN 'very experienced' THEN 4
        WHEN 'professional'     THEN 5
      END
    ), ' / 5') AS average_experience
FROM staff    AS s
JOIN building AS b ON s.building_id = b.building_id
JOIN event    AS e ON b.building_id = e.building_id
JOIN festival AS f ON e.festival_id  = f.festival_id
WHERE s.category = 'technical'
GROUP BY f.festival_year
ORDER BY average_experience ASC
LIMIT 1;
```

Με τα JOINs συνδέουμε τους πίνακες staff → building → event → festival ώστε να βρούμε σε ποιο φεστιβάλ ανήκει κάθε υπάλληλος, ομαδοποιεί ανά festival_year, και με ORDER BY average_experience ASC LIMIT 1 επιλέγει το έτος που έχει το χαμηλότερο μέσο όρο εμπειρίας. Ο μέσος όρος εμπειρίας του τεχνικού προσωπικού (ισοδυναμώντας τις πέντε βαθμίδες εμπειρίας – "beginner" = 1 έως "professional" = 5 – σε αριθμητικές τιμές), στη συνέχεια μετατρέπει το αποτέλεσμα σε μορφή "x / 5" για ευκολότερη ανάγνωση, φιλτράροντας μόνο τους υπαλλήλους της κατηγορίας "technical"

B.8 Βρείτε το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία;
Έστω ότι αυτή η συγκεκριμένη ημερομηνία είναι 2022-08-09.

```sql
SELECT DISTINCT s.staff_id, s.staff_name, s.category
FROM event_staff_view AS s
WHERE s.category = 'support'
AND s.staff_id NOT IN (
    SELECT staff_id
    FROM event_staff_view
    WHERE event_date = '2022-08-09'
);
```

Το παραπάνω query επιστρέφει τα staff_id και το ονοματεπώνυμο του προσωπικού υποστήριξης που δεν έχει προγραμματισμένη εργασία για την ημερομηνία 2022-08-09. Έχουμε υλοποιήσει το VIEW με όνομα view_event_staff, το οποίο συνδέει τα events με το προσωπικό μέσω του κτιρίου (building), αφού κάθε event είναι συσχετισμένο με ένα συγκεκριμένο κτίριο και κάθε εργαζόμενος ανήκει επίσης σε κάποιο κτίριο. Το view_event_staff μας επιστρέφει για κάθε event: το event_id, την ημερομηνία διεξαγωγής (event_date), την κατηγορία προσωπικού (category), το staff_id και το ονοματεπώνυμο (staff_name). Μέσα από αυτό το view, επιλέγουμε μόνο το προσωπικό που ανήκει στην κατηγορία υποστήριξης (support) και στη συνέχεια ελέγχουμε αν δεν υπάρχει καμία εγγραφή με το staff_id τους για την ημερομηνία 2022-08-09 (χρησιμοποιώντας NOT IN). Τέλος, γράφουμε SELECT DISTINCT ώστε να αποφύγουμε πολλαπλές εγγραφές του ίδιου προσωπικού ασφαλειάς που δεν εργάζεται εκείνη την ημερομηνία. Έτσι, καταλήγουμε σε όσους δεν έχουν εργασία την ημέρα αυτή.

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

```sql
SELECT *
FROM visitor_festival_attended_view AS v
WHERE (festival_year, event_visited) IN (
  SELECT festival_year, event_visited
  FROM visitor_festival_attended_view
  GROUP BY festival_year, event_visited
  HAVING COUNT(*) > 1
)
ORDER BY festival_year, event_visited;
```

Σε αυτό το query χρησιμοποιήσαμε το view visitor_festival_attended_view, το οποίο υπολογίζει για κάθε επισκέπτη τον αριθμό παραστάσεων που παρακολούθησε σε κάθε φεστιβάλ ανά έτος, λαμβάνοντας υπόψη μόνο τις περιπτώσεις όπου έχει παρακολουθήσει περισσότερες από 3 παραστάσεις, για να εντοπίσουμε ζεύγη ή ομάδες επισκεπτών που, στο ίδιο έτος, έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων, εντός του ίδιου φεστιβάλ.

B.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.

```sql
SELECT
    v1.genre_name AS first_genre,
    v2.genre_name AS second_genre,
    COUNT(*) AS pair_count
FROM
    (SELECT ag.artist_id, ag.genre_id, g.genre_name
     FROM artist_genre ag
     JOIN genre g ON ag.genre_id = g.genre_id
     UNION ALL
     SELECT bg.band_id AS artist_id, bg.genre_id, g.genre_name
     FROM band_genre bg
     JOIN genre g ON bg.genre_id = g.genre_id) AS v1
JOIN
    (SELECT ag.artist_id, ag.genre_id, g.genre_name
     FROM artist_genre ag
     JOIN genre g ON ag.genre_id = g.genre_id
     UNION ALL
     SELECT bg.band_id AS artist_id, bg.genre_id, g.genre_name
     FROM band_genre bg
     JOIN genre g ON bg.genre_id = g.genre_id) AS v2
ON v1.artist_id = v2.artist_id AND v1.genre_id < v2.genre_id
GROUP BY v1.genre_name, v2.genre_name
ORDER BY pair_count DESC
LIMIT 3;
```

Αυτό το ερώτημα βρίσκει τα τρία πιο “δημοφιλή” ζεύγη μουσικών ειδών που συναντώνται σε καλλιτέχνες και μπάντες για το ίδιο καλλιτεχνικό σύνολο. Χρησιμοποιεί τα tables artist_genre και band_genre δύο φορές μέσω UNION ALL, συνδυάζοντας τα μουσικά είδη που ανήκουν σε καλλιτέχνες και μπάντες. Στη συνέχεια, συνδέει τα δεδομένα μέσω ενός self JOIN με βάση το ίδιο artist_id και το επιπλέον κριτήριο ότι το genre_id του πρώτου μουσικού είδους είναι μικρότερο από του δεύτερου, αποφεύγοντας την καταμέτρηση αντίστροφων ζευγών (π.χ., (Rock, Pop) και (Pop, Rock)). Η ομαδοποίηση γίνεται κατά το όνομα του πρώτου και δεύτερου μουσικού είδους, ενώ η

συχνότητα εμφάνισης των ζευγών υπολογίζεται με την συνάρτηση COUNT(*). Τα αποτελέσματα ταξινομούνται κατά φθίνουσα συχνότητα και περιορίζονται στα τρία πρώτα ζεύγη με τις περισσότερες εμφανίσεις (LIMIT 3).

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

```sql
SELECT
    artist_id,
    artist_name,
    age,
    total_performances
FROM artist_total_participations_view
WHERE total_performances <= (
    SELECT MAX(total_performances) - 5
    FROM artist_total_participations_view
)
ORDER BY total_performances DESC;
```

Το παραπάνω query χρησιμοποιεί το view_artist_total_participations, το οποίο επιστρέφει για κάθε καλλιτέχνη το artist_id, το ονοματεπώνυμο, την ηλικία και τον συνολικό αριθμό συμμετοχών του σε φεστιβάλ που έχουν ολοκληρωθεί (δηλαδή f.end_date <= CURDATE()). Με βάση αυτό το view, το query εντοπίζει τους καλλιτέχνες που έχουν τουλάχιστον **5 λιγότερες συμμετοχές** από τον καλλιτέχνη με τις περισσότερες WHERE total_performances <= SELECT MAX(total_performances) – 5 FROM view_artist_total_participations), και επιστρέφει τα στοιχεία τους ταξινομημένα κατά φθίνουσα σειρά εμφανίσεων (ORDER BY total_performances DESC), ώστε να φαίνονται πρώτοι όσοι έχουν τις περισσότερες από τις «χαμηλές» συμμετοχές.

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

```sql
SELECT
    DATE(e.event_start_time) AS event_date,
    COUNT(DISTINCT CASE WHEN s.category = 'Technical' THEN s.staff_id END) AS technical_staff,
    COUNT(DISTINCT CASE WHEN s.category = 'Security' THEN s.staff_id END) AS security_staff,
    COUNT(DISTINCT CASE WHEN s.category = 'Support' THEN s.staff_id END) AS support_staff
FROM event e
JOIN festival f ON e.festival_id = f.festival_id
JOIN staff s ON s.building_id = e.building_id
GROUP BY event_date
ORDER BY event_date;
```

Αυτό το query υπολογίζει το απαιτούμενο προσωπικό ανά κατηγορία (τεχνικό προσωπικό, προσωπικό ασφαλείας, βοηθητικό προσωπικό) για κάθε ημέρα του φεστιβάλ. Χρησιμοποιώντας τη συνάρτηση COUNT(DISTINCT ...), εξασφαλίζεται ότι κάθε υπάλληλος μετριέται μόνο μία φορά ανά κατηγορία και ημερομηνία, ανεξαρτήτως του πόσες παραστάσεις έχει συμμετάσχει την ίδια ημέρα. Τα αποτελέσματα ομαδοποιούνται κατά ημερομηνία εκδήλωσης, και το προσωπικό μετράται ξανά σε διαφορετικές ημέρες, αν είναι απαιτούμενο για άλλες παραστάσεις. Το αποτέλεσμα ταξινομείται κατά ημερομηνία για να παρέχει μια σαφή εικόνα του προσωπικού που απαιτείται σε κάθε ημέρα του φεστιβάλ.

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

```sql
SELECT
    a.artist_first_name AS first_name,
    a.artist_last_name AS last_name,
    COUNT(DISTINCT l.continent) AS performances_across_continents
FROM
    artist a
JOIN performance p ON p.artist_id = a.artist_id
JOIN event e ON e.event_id = p.event_id
JOIN festival f ON f.festival_id = e.festival_id
JOIN location l ON f.location_id = l.location_id
GROUP BY
    a.artist_id, a.artist_first_name, a.artist_last_name
HAVING
    COUNT(DISTINCT l.continent) >= 3
ORDER BY
    performances_across_continents DESC;
```

Αυτό το ερώτημα εντοπίζει όλους τους καλλιτέχνες που έχουν εμφανιστεί σε φεστιβάλ σε τουλάχιστον τρεις διαφορετικές ηπείρους. Αρχικά συνδέει τους πίνακες artist → performance → event → festival → location βάσει τα αντίστοιχα id τους, ώστε για κάθε καλλιτέχνη να συγκεντρωθούν οι γεωγραφικές πληροφορίες των φεστιβάλ που συμμετείχε. Έπειτα, το COUNT(DISTINCT l.continent) μετράει πόσες μοναδικές ηπείρους έχει καλύψει ο κάθε καλλιτέχνης, το GROUP BY a.artist_id, a.artist_first_name, a.artist_last_name εξασφαλίζει μία γραμμή ανά καλλιτέχνη, και με το HAVING COUNT(DISTINCT l.continent) >= 3 διατηρούμε μόνο όσους έχουν τουλάχιστον τρεις ηπείρους. Τέλος, το ORDER BY performances_across_continents DESC φέρνει πρώτους τους πιο «παγκόσμιους» καλλιτέχνες, δηλαδή αυτούς με τις περισσότερες ηπείρους συμμετοχών.

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

```sql
SELECT
    t1.genre_name,
    t1.festival_year AS year1,
    t2.festival_year AS year2,
    t1.total_genre_performances
FROM genre_total_performances_year_view AS t1
INNER JOIN genre_total_performances_year_view AS t2
  ON t1.genre_name = t2.genre_name
 AND t2.festival_year = t1.festival_year + 1
 AND t1.total_genre_performances = t2.total_genre_performances
WHERE t1.total_genre_performances >= 3 AND t2.total_genre_performances>= 3
ORDER BY t1.genre_name, year1;
```

Το παραπάνω query αξιοποιεί το view genre_total_performances_year_view, το οποίο μας επιστρέφει για κάθε μουσικό είδος και κάθε έτος τον συνολικό αριθμό εμφανίσεων(solo performances+band performances of this genre) που είχαν στο φεστιβάλ, λαμβάνοντας υπόψη μόνο τα φεστιβάλ που έχουν ολοκληρωθεί (δηλαδή f.end_date <= CURDATE()).Πιο συγκεκριμένα, κάνουμε join δύο αντικειμένων αυτού του view ώστε να συγκρίνουμε **κάθε είδος** με τον εαυτό του(ON t1.genre_name = t2.genre_name) σε **δύο συνεχόμενες χρονιές** (t2.festival_year = t1.festival_year + 1), κρατάμε μόνο τα είδη που είχαν **ακριβώς τον ίδιο αριθμό εμφανίσεων** στις δύο συνεχόμενες χρονιές

(t1.total_genre_performances = t2.total_genre_performances), και φιλτράρουμε ώστε να εμφανίζονται μόνο όσες χρονιές έχουν **τουλάχιστον 3 εμφανίσεις** (t1.total_genre_performances >= 3 AND t2.total_genre_performances >= 3). Τέλος, επιστρέφουμε το είδος, τις δύο συνεχόμενες χρονιές (year1, year2) και τον αριθμό των εμφανίσεων, ταξινομημένα με βάση το είδος και το πρώτο έτος.

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

Έστω ότι αυτός ο καλλιτέχνης αντιστοιχεί στον artist_id = 10.

```sql
SELECT
    CONCAT(t.holder_first_name, ' ', t.holder_last_name) AS visitor_full_name,
    CONCAT(a.artist_first_name, ' ', a.artist_last_name) AS artist_full_name,
    SUM(
        CASE r.artist_performance_grade
            WHEN 'Very Unsatisfied' THEN 1
            WHEN 'Unsatisfied' THEN 2
            WHEN 'Neutral' THEN 3
            WHEN 'Satisfied' THEN 4
            WHEN 'Very Satisfied' THEN 5
        END
     ) AS total_artist_score
FROM review as r
JOIN ticket as t ON r.visitor_id = t.visitor_id
JOIN performance as p ON r.performance_id = p.performance_id
JOIN artist as a ON p.artist_id = a.artist_id
WHERE a.artist_id = 10
GROUP BY visitor_full_name, artist_full_name
ORDER BY total_artist_score DESC
LIMIT 5;
```

Σε αυτό το query, για κάθε αξιολόγηση από τον πίνακα review βρίσκουμε το εισιτήριο του αντίστοιχου επισκέπτη μέσω JOIN με τον πίνακα ticket και παίρνουμε το ονοματεπώνυμο τού. Έπειτα, βρίσκουμε το artist_id του καλλιτέχνη, ο οποίος συμμετείχε στην αντίστοιχη εμφάνιση για την οποία έκανε την αξιολόγηση, μέσω JOIN με τον πίνακα performance. Επίσης, βρίσκουμε το ονοματεπώνυμο του καλλιτέχνη με JOIN με τον πίνακα artist. Τέλος, υπολογίζουμε τη συνολική βαθμολογία του επισκέπτη για αυτόν τον καλλιτέχνη με μετατροπή των λεκτικών τιμών σε αριθμητική κλίμακα 1–5 και ομαδοποιώντας βάσει των ονοματεπωνύμων τους. Στη συνέχεια, ταξινομούμε σε φθίνουσα σε σειρά (ORDER BY total_artist_score DESC) με βάση τη συνολική βαθμολογία κάθε επισκέπτη και επιλέγουμε 5 από αυτούς που έχουν δώσει τη συνολικά υψηλότερη βαθμολογία με LIMIT 5. Το αποτέλεσμα παρουσιάζει το ονοματεπώνυμο του επισκέπτη, το ονοματεπώνυμο του καλλιτέχνη και το συνολικό σκορ αξιολόγησης, ταξινομημένο κατά φθίνουσα σειρά.

Γ. Υλοποίηση εφαρμογής

Γ. 1. Επισκόπηση Εφαρμογής

Η εφαρμογή είναι χτισμένη χρησιμοποιώντας το **Flask**, ένα micro-framework για την ανάπτυξη εφαρμογών στον Python, με στόχο να παρέχει ένα δυναμικό περιβάλλον για την εξερεύνηση διαφορετικών πτυχών της βάσης δεδομένων του μουσικού φεστιβάλ. Οι χρήστες μπορούν να δουν και να αλληλεπιδράσουν με πληροφορίες που αφορούν τα φεστιβάλ, τους καλλιτέχνες, τις μπάντες και τις παραστάσεις, καθώς και να εκτελούν custom SQL queries στη βάση δεδομένων. Η εφαρμογή είναι δομημένη έτσι ώστε να προσφέρει εύκολη και φιλική προς το χρήστη εμπειρία με λειτουργίες όπως:

- Εμφάνιση των φεστιβάλ και των εκδηλώσεων
- Προβολή καλλιτεχνών και μπαντών που συμμετέχουν σε αυτά τα φεστιβάλ
- Δυνατότητα εκτέλεσης SQL queries για ανάλυση δεδομένων

Γ. 2. Τεχνολογίες και Εργαλεία

Για την ανάπτυξη της εφαρμογής, χρησιμοποιήθηκαν οι εξής τεχνολογίες:

- **Python 3.8+**: Η γλώσσα προγραμματισμού για τη δημιουργία της εφαρμογής.
- **Flask**: Η web framework για την κατασκευή του server-side μέρους της εφαρμογής.
- **MySQL**: Χρησιμοποιείται για την αποθήκευση και διαχείριση της βάσης δεδομένων.
- **Jinja2**: Για την δυναμική δημιουργία HTML σελίδων από το Flask.
- **HTML, CSS**: Χρησιμοποιήθηκαν για την ανάπτυξη της διεπαφής χρήστη (UI).

Γ. 3. Δομή Εφαρμογής

Η εφαρμογή περιλαμβάνει τις παρακάτω βασικές ενότητες:

- **Αρχική Σελίδα (Home Page)**: Παρουσιάζει την κεντρική πληροφορία για την εφαρμογή και παρέχει συνδέσμους για τα φεστιβάλ, τους καλλιτέχνες και τις μπάντες.
- **Φεστιβάλ (Festivals)**: Εμφανίζει μια λίστα φεστιβάλ με εικόνες και πληροφορίες, δίνοντας τη δυνατότητα στον χρήστη να δει περισσότερες λεπτομέρειες για κάθε φεστιβάλ.
- **Καλλιτέχνες (Artists)**: Προβάλλει καλλιτέχνες με εικόνες και περιγραφές, καθώς και τα φεστιβάλ στα οποία συμμετέχουν.
- **Μπάντες (Bands)**: Παρουσιάζει μπάντες και τις πληροφορίες τους με ανάλογο τρόπο.
- **SQL Query Interface**: Παρέχει στον χρήστη τη δυνατότητα να εισάγει και να εκτελέσει SQL ερωτήματα για να ανακτήσει δεδομένα απευθείας από τη βάση δεδομένων.

Γ. 4. Οδηγίες Εγκατάστασης και Χρήσης

Για την εκτέλεση της εφαρμογής, πρέπει να ακολουθηθούν τα εξής βήματα:

1. **Εγκατάσταση Python**: Αρχικά, βεβαιωθείτε ότι έχετε εγκατεστημένο το Python 3.8 ή νεότερη έκδοση.
2. **Εγκατάσταση Απαραίτητων Βιβλιοθηκών**: Εκτελέστε την εντολή pip install -r requirements.txt για να εγκαταστήσετε τις απαραίτητες βιβλιοθήκες (Flask, MySQL Connector, κ.λπ.).

3. **Ρύθμιση Βάσης Δεδομένων**: Εισάγετε τα δεδομένα στη βάση MySQL, δημιουργώντας τη βάση και τα αντίστοιχα tables. Αυτό μπορεί να γίνει με το script που παρέχεται.
4. **Εκκίνηση της Εφαρμογής**: Εκκινήστε τον server εκτελώντας την εντολή app.py. Στη συνέχεια, ανοίξτε τον περιηγητή σας στη διεύθυνση http://127.0.0.1:5000 για να αποκτήσετε πρόσβαση στην εφαρμογή.