

2025

Αναφορά Εξαμηνιαίας Εργασίας Βάσεων Δεδομένων

ΜΑΤΘΑΙΟΣ ΑΓΓΕΛΗΣ (el22027)

ΝΙΚΟΛΑΟΣΑΘΑΝΑΣΟΠΟΥΛΟΣ(el22101)

ΝΙΚΗ ΠΑΠΑΔΗΜΗΤΡΙΟΥ (el22005)

6ο Εξάμηνο ΗΜΜΥ

Ακαδημαϊκό Έτος 2024-25

Περιεχόμενα

[1. Βάση Δεδομένων 2](#_Toc198062993)

[1.1. Παραδοχές και Σχεδιαστικές Αποφάσεις 2](#_Toc198062994)

[1.2. ΔιάγραμμαΟντοτήτων-Συσχετίσεων (Entity-Reletionship Diagram) 3](#_Toc198062995)

[1.3. Σχεσιακό σχήμα (Reletional Schema) 3](#_Toc198062996)

[1.4. Ευρετήρια (Indexes) 4](#_Toc198062997)

[1.5. DDL Script 4](#_Toc198062998)

[Tables 4](#_Toc198062999)

[Triggers 6](#_Toc198063000)

[Procedures 9](#_Toc198063001)

[1.6. DML Scripts 10](#_Toc198063002)

[2. Ανάλυση Queries 10](#_Toc198063003)

# Βάση Δεδομένων

## Παραδοχές και Σχεδιαστικές Αποφάσεις

Κατά τη μοντελοποίηση και υλοποίηση του συστήματος λήφθηκαν υπόψη ορισμένες βασικές παραδοχές και σχεδιαστικές αποφάσεις που εξασφαλίζουν τη ρεαλιστικότητα και τη λειτουργική συνέπεια της βάσης δεδομένων.

* **Επικοινωνία επισκεπτών**: Από τα στοιχεία επικοινωνίας του κάθε Visitor αποθηκεύουμε μόνο το email, θεωρώντας ότι αυτό επαρκεί για τις ανάγκες επικοινωνίας, ειδοποιήσεων και ταυτοποίησης στο σύστημα.
* **Εισιτήρια και μεταπώληση**: Κάθε εισιτήριο μπορεί να μεταπωληθεί μόνο μία φορά. Αυτό επιβάλλεται τόσο λογικά μέσω triggers και procedures, όσο και πρακτικά, ώστε να περιοριστεί η κυκλοφορία εισιτηρίων και να διατηρείται η ακεραιότητα των μεταβιβάσεων.
* **Δημιουργία φεστιβάλ**: Η βάση περιλαμβάνει 10 φεστιβάλ, εκ των οποίων 8 είναι παρελθοντικά και 2 μελλοντικά, ώστε να καλύπτονται και σενάρια ιστορικής ανάλυσης και σενάρια κράτησης.
* **Εισιτήρια και εκδηλώσεις**: Τα εισιτήρια (Ticket) είναι συνδεδεμένα μόνο με συγκεκριμένα events συγκεκριμένων φεστιβάλ. Δεν υποστηρίζονται “γενικά” εισιτήρια που ισχύουν για πολλαπλές εκδηλώσεις.
* **Ενεργοποίηση εισιτηρίων**: Ενεργοποιημένα εισιτήρια (Is\_Activated = TRUE) αντιστοιχούν μόνο σε events που έχουν ήδη πραγματοποιηθεί, δηλαδή σε παρελθοντικά φεστιβάλ, καθώς η ενεργοποίηση γίνεται κατά την είσοδο στον χώρο. Να σημειωθεί σε αυτό το σημείο ότι το festival του έτους 2025 έχει θεωρηθεί παρελθοντικό.
* **Περιορισμοί διαγραφών**: Δεν επιτρέπεται η διαγραφή των πινάκων Festival και Event, ώστε να διατηρείται πλήρες ιστορικό, ακόμα και μετά την ολοκλήρωση ενός φεστιβάλ. Αυτό επιβάλλεται μέσω triggers.
* **Διαγραφή παραστάσεων**: Επιτρέπεται η διαγραφή παραστάσεων (Performance). Σε αυτή την περίπτωση, καλείται διαδικασία (Reschedule\_Performances\_After\_Deletion) που προσπαθεί να αναπρογραμματίσει τις υπόλοιπες παραστάσεις του ίδιου event, υπολογίζοντας νέα Start\_Time με ενδιάμεσα διαλείμματα διαλείμματα επιτρεπτής διάρκειας (από 5 εώς 30 λεπτά).
* **Αξιολογήσεις με LikertScale**: Οι κριτικές (Review) ακολουθούν την κλασική κλίμακα Likert από 1 έως 5, όπου το 1 αντιστοιχεί στη χειρότερη και το 5 στην καλύτερη βαθμολογία.
* **EANCode εισιτηρίων**: Το EAN\_CODE κάθε εισιτηρίου παράγεται με μοναδικό και ντετερμινιστικό τρόπο από το ID του εισιτηρίου, μέσω συνάρτησης που έχουμε υλοποιήσει. Με αυτόν τον τρόπο διασφαλίζεται η μοναδικότητα και η ιχνηλασιμότητα των εισιτηρίων. Η συνάρτηση αυτή φαίνεται παρακάτω:

# ---------------- EAN-13 GENERATION ----------------

def calculate\_ean13\_check\_digit(ean12):

    """

    Calculate the EAN-13 check digit for a 12-digit number.

    :param ean12: A string of 12 digits.

    :return: A single check digit as a string.

    """

    if len(ean12) != 12 or not ean12.isdigit():

        raise ValueError("Input must be a 12-digit numeric string.")

    total = 0

    for i, digit in enumerate(ean12):

        num = int(digit)

        if (i + 1) % 2 == 0:  # Even position (2nd, 4th, ..., 12th)

            total += num \* 3

        else:  # Odd position (1st, 3rd, ..., 11th)

            total += num

    check\_digit = (10 - (total % 10)) % 10

    return str(check\_digit)

def generate\_ean13(ticket\_id):

    """

    Generate a valid EAN-13 code using the ticket ID.

    :param ticket\_id: Integer representing the ticket ID.

    :return: A 13-digit EAN-13 code as a string.

    """

    prefix = '200'  # Custom prefix for your organization

    company\_code = '0001'  # Example company code

    product\_code = str(ticket\_id).zfill(5)  # Zero-padded to ensure 5 digits

    ean12 = prefix + company\_code + product\_code

    check\_digit = calculate\_ean13\_check\_digit(ean12)

    return ean12 + check\_digit

Η συνάρτηση που παρέχετε έχει δύο βασικές λειτουργίες:

1. **Υπολογισμός του ελέγχου EAN-13 (check digit)**:
   * Η συνάρτηση calculate\_ean13\_check\_digit υπολογίζει το ψηφίο ελέγχου (check digit) για έναν αριθμό 12 ψηφίων, ο οποίος είναι το πρώτο μέρος του EAN-13 κωδικού. Ο υπολογισμός γίνεται με βάση τον αλγόριθμο του EAN-13, ο οποίος περιλαμβάνει μια σειρά βαρών (1 για τις περιττές θέσεις και 3 για τις άρτιες θέσεις) και τον υπολογισμό του υπολοίπου της διαίρεσης του συνολικού αθροίσματος με το 10.
   * Αφού γίνει αυτό, το check digit υπολογίζεται ως το αποτέλεσμα της διαφοράς του 10 από το υπόλοιπο του αθροίσματος όταν διαιρείται με το 10. Αν το υπόλοιπο είναι 0, τότε το check digit είναι 0.
2. **Γεννήτρια EAN-13**:
   * Η συνάρτηση generate\_ean13 χρησιμοποιεί έναν αριθμό εισιτηρίου (ticket ID) για να δημιουργήσει έναν πλήρη κωδικό EAN-13. Ο κωδικός αυτός αποτελείται από:
     + Ένα προκαθορισμένο πρόθεμα (prefix = '200'), το οποίο μπορεί να αντιπροσωπεύει την οργάνωση ή τον τύπο του προϊόντος.
     + Έναν κωδικό εταιρείας (company\_code = '0001'), ο οποίος είναι ένας αριθμός που μπορεί να αναπαριστά μια εταιρεία ή μια περιοχή.
     + Έναν αριθμό προϊόντος που σχηματίζεται από τον ticket\_id (ο αριθμός εισιτηρίου), τον οποίο η συνάρτηση μετατρέπει σε έναν 5ψήφιο αριθμό με μηδενικά στην αρχή αν χρειάζεται.
   * Στη συνέχεια, συνδυάζει όλα αυτά τα μέρη (προθέμα, κωδικός εταιρείας και προϊόν) για να δημιουργήσει έναν αριθμό 12 ψηφίων και προσθέτει το ψηφίο ελέγχου που υπολογίζεται με την calculate\_ean13\_check\_digit.
   * Ο τελικός κωδικός EAN-13 που επιστρέφει είναι μια αλφαριθμητική ακολουθία 13 ψηφίων.

* **Προσωπικό**: Έχουμε θεωρήσει ότι για κάθε event προσλαμβάνεται, ανάλογα με τις ανάγκες της σκηνής στην οποία πραγματοποιείται, προσωπικό ασφαλείας, υποστηρικτικό και τεχνικό. Το προσωπικό αυτό αλλάζει για διαφορετικό event ακόμα και αν αυτό πραγματοποιείται στην ίδια σκηνή με κάποιο άλλο. Επίσης το **ελάχιστο** **απαραίτητο** προσωπικό σε κάθε σκηνή έχουμε θεωρήσει ότι είναι ίσο με το 10% του συνολικού αριθμού των θεατών σε κάθε σκηνή, και το ποσοστό αυτό καταμερίζεται σε ρόλους ως εξής:

1. Το ασφαλείας καλύπτει το 5% του συνολικού αριθμού των θεατών σε κάθε σκηνή.
2. Το υποστηρικτικό καλύπτει το 2% του συνολικού αριθμού των θεατών σε κάθε σκηνή.
3. Το τεχνικό καλύπτει (με δική μας απόφαση) το 3% του συνολικού αριθμού των θεατών σε κάθε σκηνή.

Ωστόσο εμείς όταν προγραμματίζουμε ένα event, επειδή δε γνωρίζουμε εξαρχής τι συμμετοχή θα έχει (πόσοι επισκέπτες θα δείξουν ενδιαφέρον για αυτό), φροντίζουμε να προσλάβουμε για κάθε κατηγορία προσωπικού τα ποσοστά που αναφέρθηκαν παραπάνω επί της μέγιστης χωρητικότητας της σκηνής στην οποία θα πραγματοποιηθεί η παράσταση (ακόμα κι αν αυτό **υπερβαίνει το** **ελάχιστο απαραίτητο** προσωπικό).

* **Περιορισμός στη συμμετοχή καλλιτεχνών**: Στη εκφώνηση τέθηκε ο εξής περιορισμός: «Δεν επιτρέπεται συμμετοχή του ίδιου καλλιτέχνη (συγκροτήματος) για περισσότερα από 3 συνεχή έτη». Εμείς με τη χρήση triggers εξασφαλίζουμε ότι ένας σόλο καλλιτέχνης ή μία μπάντα δεν μπορεί να κάνει 4η συνεχόμενη εμφάνιση σε φεστιβάλ με την εξής παραδοχή: Αν ένας σόλο καλλιτέχνης είναι μέλος μπάντας, και έχει ήδη κάνει τρεις συνεχόμενες σόλο εμφανίσεις, αυτό δεν τον εμποδίζει από το να εμφανιστεί την 4η συνεχόμενη χρονιά ως μέλος μιας μπάντας (γιατί η μπάντα λογίζεται ως διαφορετικός καλλιτέχνης και συνεπώς δεν αίρεται ο περιορισμός που τέθηκε στην εκφώνηση).
* **Επικαλύψεις καλλιτεχνών επί σκηνής:** Με κατάλληλο trigger αποτρέπουμε καλλιτέχνης να δίνει παράσταση σε πολλές σκηνές ταυτόχρονα αλλά και σε μέλη συγκροτήματος να κάνουν σόλο εμφανίσεις την ώρα που δίνει παράσταση το συγκρότημα στο οποίο συμμετέχουν.
* **Κατηγορίες Εισητηρίων και Τρόποι Πληρωμής:** Θεωρήσαμε τις εξής κατηγορίες εισητηρίων: **Student, Standard, Backstage, VIP** (<=10% της χωρητικότητας της σκηνής) και ως τρόπους πληρωμής: **Paypal, Bank Transfer, Cash, Credit Card**.

## Διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Reletionship Diagram)

Αρχικάφτιάχτηκε το διάγραμμα οντοτήτων-συσχετίσεωνστο περιβάλλον‘Draw.io’.Αυτό παρουσιάζεται ακολούθως:

A diagram of a computer program

AI-generated content may be incorrect.

Το ER διάγραμμα αποτυπώνει τις βασικές οντότητες του συστήματος και τις σχέσεις μεταξύ τους, βασισμένο στις απαιτήσεις της εκφώνησης για τη διαχείριση φεστιβάλ. Η οντότητα Festival συνδέεται με μία Location και αποτελείται από πολλά Event, καθένα από τα οποία φιλοξενείται σε μία Stage. Οι Performers (είτε μεμονωμένοι καλλιτέχνες είτε μπάντες) συμμετέχουν σε Performance εντός Event. Υπάρχει επιπλέον συσχέτιση Membership που επιτρέπει την οργάνωση συγκροτημάτων με τα μέλη τους. Κάθε Performance έχει χαρακτηριστικά όπως τύπος (warm-up, headliner), ώρα και διάρκεια.

Οι επισκέπτες (Visitor) αγοράζουν Ticket για να παρακολουθήσουν Events και μέσω των Spectator συνδέονται με τα εισιτήρια τους. Έχουν τη δυνατότητα να καταχωρούν Review με βαθμολογίες σε πολλαπλά κριτήρια. Το μοντέλο υποστηρίζει λειτουργίες μεταπώλησης με τους πίνακες Buy\_Queue, Resale\_Queue, Tickets\_In\_Resale και Transaction, ακολουθώντας τη λογική FIFO. Υπάρχει πρόβλεψη για λίστα αναμονής (Visitor\_Waitlisted) και εκδήλωση ενδιαφέροντος (Visitor\_Interested\_Event). Όλες οι οικονομικές ενέργειες καταγράφονται μέσω του Transaction, όπου αναφέρεται και ο Buyer και ο Seller.

Για το προσωπικό, η γενική οντότητα Staff εξειδικεύεται σε Technical, Security και Helping, ενώ το event\_staff συσχετίζει προσωπικό με συγκεκριμένες εκδηλώσεις. Τέλος, το διάγραμμα περιλαμβάνει τις συσχετίσεις των Performers με μουσικά είδη (Genre, Subgenre) και των οντοτήτων με Image.

## Σχεσιακό σχήμα (Reletional Schema)

Στη συνέχεια, με βάση το παραπάνω διάγραμμα, σχεδιάστηκε το σχεσιακό σχήμα της βάσης δεδομένων, ακολουθώντας την Κανονική Μορφή Boyce-Codd (BCNF), ώστε κάθε σύνολο γνωρισμάτων να εξαρτάται αποκλειστικά από υπερκλειδί. Με αυτόν τον τρόπο αποφεύγεται ο πλεονασμός δεδομένων. Το DDL script που υλοποιεί τη βάση παρατίθεται σε επόμενο παράρτημα, ενώ το τελικό σχεσιακό διάγραμμα (relational diagram) παρουσιάζεται στο ακόλουθο σχήμα:

A diagram of a computer

AI-generated content may be incorrect.

Το σχεσιακό σχήμα ακολουθεί το ER διάγραμμα και αποτυπώνει πλήρως τις σχέσεις μεταξύ των πινάκων και τα απαραίτητα foreignkeys. Ο πίνακας Festival περιέχει Location\_ID ως FK, και ο Event περιλαμβάνει Festival\_ID και Stage\_ID. Οι Tickets σχετίζονται με Event\_ID, Type\_ID και Payment\_ID, ενώ ο Spectator λειτουργεί ως πίνακας σύνδεσης επισκέπτη-εισιτηρίου. Οι Reviews συσχετίζονται με τα Tickets, διασφαλίζοντας ότι αξιολογήσεις μπορούν να γίνονται μόνο από επισκέπτες που έχουν αγοράσει και ενεργοποιήσει εισιτήριο.

Οι Performers συνδέονται με Performance και μέσα από τον πίνακα Membership με μπάντες. Για τα μουσικά είδη, υπάρχει πλήρης αναπαράσταση με Genre, Subgenre και τον συσχετιστικό πίνακα perf\_subgenre. Η μεταπώληση υλοποιείται μέσω των πινάκων Buy\_Queue, Resale\_Queue, Tickets\_In\_Resale, Visitor\_Waitlisted, Visitor\_Sold\_Ticket και Transaction, με αναλυτικά FKs που διασφαλίζουν ακεραιότητα και ανιχνευσιμότητα.

Το σχήμα περιλαμβάνει εξειδίκευση του προσωπικού μέσω πινάκων Technical, Security και Helping, οι οποίοι σχετίζονται με το Staff μέσω πεδίου Role, και συνδέονται με event\_staff. Η διαχείριση εικόνων υποστηρίζεται από τον πίνακα Image και σχετικούς συσχετιστικούς πίνακες για κάθε βασική οντότητα (Event\_Image, Festival\_Image, Staff\_Image, κ.λπ.). Όλα τα FKs δηλώνονται ρητά και η δομή του σχήματος διασφαλίζει πλήρη κανονικοποίηση έως BCNF.

## Ευρετήρια (Indexes)

Με βάση τα ερωτήματα που εκτελούνται στη βάση δεδομένων μας, δημιουργήσαμε τα κατάλληλα ευρετήρια (indexes) ώστε να βελτιώσουμε την αποδοτικότητα των ερωτημάτων. Η MySQL δημιουργεί αυτόματα ευρετήρια για όλα τα πεδία που αποτελούν primarykeys, χωρίς να απαιτείται κάποια επιπλέον ενέργεια από τον προγραμματιστή. Επιπλέον, για λόγους βελτίωσης της απόδοσης των ενώσεων (joins), δημιουργήσαμε ευρετήρια σε όλα τα πεδία που λειτουργούν ως foreignkeys. Με αυτόν τον τρόπο, μετατρέπουμε τον χρόνο αναζήτησης από γραμμικό (O(n)) σε λογαριθμικό (O(logn)).

Παρότι προσθέσαμε αρκετά ευρετήρια, αξίζει να σημειωθεί ότι η MySQL διαθέτει τον δικό της queryoptimizer, ο οποίος επιλέγει αυτόματα το πιο αποδοτικό μονοπάτι εκτέλεσης κάθε φορά. Παρατηρήσαμε μικρή βελτίωση στους χρόνους εκτέλεσης των queries, καθώς η προσθήκη πολλών ευρετηρίων αυξάνει τον χρόνο που απαιτείται για την επιλογή του βέλτιστου, γεγονός που περιορίζει εν μέρει το όφελος από τα επιπλέον indexes.

## DDL Script

### Tables

Αρχικά, δημιουργούμε τον πίνακα Location ο οποίος περιέχει τις τοποθεσίες όπου διεξάγονται τα φεστιβάλ. Το πεδίο ID είναι το πρωτεύον κλειδί και όλοι οι υπόλοιποι γνωρισματικοί τύποι είναι VARCHAR ή DECIMAL, με περιορισμό NOTNULL ώστε να μην υπάρχουν κενές τιμές. Επιπλέον, θέτουμε δύο UNIQUE περιορισμούς: έναν στον συνδυασμό Latitude και Longitude, ώστε να μην υπάρχουν ίδιες γεωγραφικές συντεταγμένες, και έναν ακόμα στον συνδυασμό Address, City, Country, Continent για να μην καταχωρείται η ίδια φυσική διεύθυνση δύο φορές.

* **Festival**: Ο πίνακας Festival περιέχει βασικές πληροφορίες για κάθε φεστιβάλ όπως Name, Start\_Date, End\_Date και Location\_ID. Το Location\_ID είναι foreignkey που παραπέμπει στον πίνακα Location. Επιπλέον, με χρήση της εντολής GENERATEDALWAYSAS δημιουργούμε το Year του φεστιβάλ από την Start\_Date. Το Year είναι UNIQUE ώστε να υπάρχει ένα φεστιβάλ ανά έτος.
* **Event**: Για κάθε εκδήλωση (Event) ορίζουμε Start\_Time, End\_Time, και αν έχει εξαντληθεί (Sold\_Out). Η Stage\_ID και η Festival\_ID είναι foreignkeys προς τους αντίστοιχους πίνακες. Επίσης, θέτουμε UNIQUE(Festival\_ID, Stage\_ID, Start\_Time) ώστε να μην υπάρχουν επικαλύψεις.
* **Performer**: Ο πίνακας Performer περιέχει τους καλλιτέχνες και τα μουσικά συγκροτήματα. Περιλαμβάνει αληθινό και καλλιτεχνικό όνομα, ημερομηνία γέννησης, κοινωνικά δίκτυα, αν είναι μπάντα και ημερομηνία ίδρυσης. Το αληθινό όνομα και η ημερομηνίας γέννησης είναι κενά σε περίπτωση που πρόκειται για πάντα, ενώ η ημερομηνία ίδρυσης είναι κενή σε περίπτωση που πρόκειται για σόλο καλλιτέχνη.
* **Performer\_Subgenre:** Για την κατηγοριοποίηση των καλλιτεχνών δημιουργούμε τον πίνακα Genre και υποκατηγορίες του μέσω Subgenre. Ο πίνακας Performer\_Subgenre διατηρεί τις αντιστοιχίσεις μεταξύ καλλιτεχνών και υποειδών. Ο πίνακας Membership καταγράφει μέλη που ανήκουν σε συγκροτήματα (Band\_ID, Artist\_ID), με trigger που απαγορεύει τη συμμετοχή μπάντας σε άλλη μπάντα.
* **Performance**: Ο πίνακας Performance υλοποιεί τη συμμετοχή τους σε events, με το Type να παίρνει συγκεκριμένες τιμές μέσω CHECK, διάρκεια σε λεπτά και Start\_Time. Υπάρχει περιορισμός UNIQUE(Event\_ID, Performer\_ID) ώστε κάθε καλλιτέχνης να εμφανίζεται μία φορά ανά event.
* **Staff/Security\_Staff/Support\_Staff/Technical\_Staff:** Για τη διαχείριση προσωπικού, χρησιμοποιούμε τον πίνακα Staff με Age και Type ελεγχόμενο από CHECK. Η εξειδίκευση σε Security\_Staff, Support\_Staff, Technical\_Staff υλοποιείται με ξεStχωριστούς πίνακες που έχουν foreignkey προς το Staff.
* **Membership**: Το σχήμα περιλαμβάνει ακόμα πίνακα για Membership μεταξύ μελών και συγκροτημάτων. Στη σχέση συμμετέχουν ΜΟΝΟ οι σόλο καλλιτέχνες που είναι μέλη συγκροτήματος.
* **Stage**: Στη συνέχεια δημιουργούμε τον πίνακα Stage, ο οποίος περιλαμβάνει τις σκηνές που φιλοξενούν εκδηλώσεις στα φεστιβάλ. Περιέχει γνωρίσματα όπως Name, Description, Capacity και Tech\_Info, με την Capacity να ορίζεται ως INTNOTNULL ώστε να υποστηρίζονται σωστά οι περιορισμοί πληρότητας μέσω triggers.
* **Visitor**: Ο πίνακας Visitor περιλαμβάνει πληροφορίες για τους επισκέπτες, όπως όνομα, ηλικία και στοιχεία επικοινωνίας. Αποτελεί βασική οντότητα για τις αγορές εισιτηρίων, τις αξιολογήσεις και τη γενικότερη αλληλεπίδραση με το σύστημα.
* **Ticket**: Ο πίνακας Ticket περιέχει όλα τα εισιτήρια του συστήματος, μαζί με πληροφορίες όπως EAN\_CODE, Price, Stage\_Info, Date\_Bought, Type\_ID, Payment\_ID και Event\_ID. Περιλαμβάνει δύο boolean πεδία: Activated και Still\_In\_Resale, και UNIQUE περιορισμό στο EAN\_CODE ώστε να μην υπάρχουν διπλά αντίτυπα.
* **Type/Payment**: Ακολουθεί ο πίνακας Type, που περιέχει τις διαθέσιμες κατηγορίες εισιτηρίων, όπως VIP ή General. Χρησιμοποιείται σε πολλούς πίνακες της βάσης ως foreignkey, όπως στους Ticket, Buy\_Queue και Visitor\_Interested\_Event. Αντίστοιχα, ο πίνακας Payment περιέχει τους τρόπους πληρωμής (π.χ. κάρτα, μετρητά, PayPal), και επίσης εμφανίζεται ως foreignkey σε πολλαπλές συσχετίσεις.
* **Spectator**: Κάθε πελάτης του φεστιβάλ που γίνεται κάτοχος εισιτηρίου εισέρχεται σε αυτόν τον πίνακα ως tupple (Visitor\_ID, Ticket\_ID). Επειδή ολόκληρο το ζεύγος έχει οριστεί ως primary key, κάθε εισιτήριο μπορεί να ανήκει σε έναν μόνο πελάτη και κάθε πελάτης δεν μπορεί να αγοράσει το ίδιο εισιτήριο πολλές φορές. Ο πίνακας γίνεται updated στην περίπτωση που έχουμε μεταπώληση.
* **Resale\_Queue/Buy\_Queue**: Η ουρά μεταπώλησης εισιτηρίων υλοποιείται μέσω του πίνακα Resale\_Queue, ο οποίος ενεργοποιείται αυτόματα από triggers όταν μια εκδήλωση χαρακτηρίζεται ως Sold\_Out, κάνοντας την παραδοχή ότι ένα εισιτήριο μπορεί να μεταπωληθεί το πολύ μία φορά. Παράλληλα, ο πίνακας Buy\_Queue καταγράφει αιτήματα επισκεπτών για αγορά εισιτηρίων που δεν είναι άμεσα διαθέσιμα. Περιλαμβάνει το event, τον τύπο εισιτηρίου, την επιθυμητή μέθοδο πληρωμής και την ημερομηνία αιτήματος (Timestamp).
* **Visitor\_Waitlisted**: Για να αποθηκευτούν οι επισκέπτες που περιμένουν να ικανοποιηθούν τα αιτήματά τους, δημιουργούμε τον πίνακα Visitor\_Waitlisted, ο οποίος αποτελεί συσχετιστικό πίνακα μεταξύ Visitor και Buy\_Queue. Στον πίνακα αυτό εισέρχονται οι επισκέπτες όταν δε βρέθηκε για το event που ζήτησαν κάποιο εισιτήριο της κατηγορίας που επιθυμούν.
* **Tickets\_In\_Resale**: Τα εισιτήρια που τίθενται προς μεταπώληση καταγράφονται στον πίνακα Tickets\_In\_Resale, ενώ ο πίνακας Spectator υλοποιεί την αντιστοίχιση επισκεπτών σε εισιτήρια μέσω ενός primary key (Visitor\_ID, Ticket\_ID). Τα εισιτήρια δε διαγράφονται ποτέ από αυτό τον πίνακα ακόμα και αν πραγματοποιηθεί μεταπώληση. Αντιθέτως, για να καταλαβαίνει η βάση μας ποια εισιτήρια μεταπωλήθηκαν και ποια περιμένουν ακόμα στην ουρά, έχουμε δημιουργήσει σε κάθε εισιτήριο ένα Boolean Still\_In\_Resale πεδίο, το οποίο είναι TRUE μόνο όταν το εισιτήριο βρίσκεται σε ουρά μεταπώλησης και είναι ακόμα διαθέσιμο προς μεταπώληση.
* **Visitor\_Interested\_Event**: Ο πίνακας Visitor\_Interested\_Event αποθηκεύει δηλωμένο ενδιαφέρον επισκεπτών για συγκεκριμένες εκδηλώσεις, πριν ακόμα προβούν σε αγορά. Περιλαμβάνει και την επιθυμητή μέθοδο πληρωμής και κατηγορία εισιτηρίου. Ένας trigger αποτρέπει την προσθήκη ενδιαφέροντος αν ο επισκέπτης έχει ήδη εισιτήριο για το event.
* **Transaction**: Ο πίνακας Transaction καταγράφει κάθε αγοραπωλησία εισιτηρίου, είτε πρόκειται για αρχική αγορά είτε για μεταπώληση. Περιλαμβάνει το ID του εισιτηρίου, τον αγοραστή, τον τρόπο πληρωμής και μια Boolean Is\_Resale που διαχωρίζει τις δύο περιπτώσεις.
* **Visitor\_Sold\_Ticket**: Για να διατηρείται ιστορικό μεταπώλησης, δημιουργήσαμε τον πίνακα Visitor\_Sold\_Ticket, που αποθηκεύει τον αρχικό κάτοχο του εισιτηρίου, το ID του εισιτηρίου και το σχετικό transaction.
* **Review**: Ο πίνακας Review περιέχει αξιολογήσεις των παραστάσεων από επισκέπτες, με επιμέρους γνωρίσματα όπως Interpretation, Sound, Lighting, Stage\_Presence, Organization και Overall. Υπάρχει περιορισμός UNIQUE στο Ticket\_ID ώστε κάθε εισιτήριο να χρησιμοποιείται για μία και μόνο αξιολόγηση, ενώ triggers διασφαλίζουν ότι επιτρέπεται μόνο αν το εισιτήριο έχει ενεργοποιηθεί και ανήκει όντως στον επισκέπτη που καταχωρεί την κριτική.
* **Image, Festival\_Image, Stage\_Image, Performer\_Image, Staff\_Image, Event\_Image και Location\_Image**:Η αποθήκευση εικόνων γίνεται στον πίνακα Image, ενώ οι πίνακες Festival\_Image, Stage\_Image, Performer\_Image, Staff\_Image, Event\_Image και Location\_Image συνδέουν φωτογραφίες με τις αντίστοιχες οντότητες. Πρόκειται για σχέσεις many-to-many με κατάλληλα foreignkeys και primarykeys.

### Triggers

Για την ενίσχυση της ακεραιότητας και της λειτουργικής ορθότητας της βάσης, δημιουργήσαμε πλήθος triggers που ενεργοποιούνται αυτόματα πριν ή μετά από συγκεκριμένες ενέργειες (INSERT, DELETE, UPDATE) σε κρίσιμους πίνακες. Τα triggers καλύπτουν περιπτώσεις περιορισμών λογικής που δεν μπορούν να εκφραστούν αποκλειστικά με CHECK ή FOREIGNKEY.

**Μη επιτρεπτή διαγραφή βασικών οντοτήτων**

Αρχικά, υλοποιούμε triggers που απαγορεύουν τη διαγραφή σημαντικών οντοτήτων:

* prevent\_festival\_deletion: Απαγορεύει τη διαγραφή φεστιβάλ, εμφανίζοντας κατάλληλο μήνυμα σφάλματος.
* prevent\_event\_deletion: Αντίστοιχος περιορισμός για εκδηλώσεις (Event).

Αυτά τα triggers διασφαλίζουν τη σταθερότητα των δεδομένων, αποτρέποντας την απώλεια σχετιζόμενης πληροφορίας.

**Λογικοί περιορισμοί στις παραστάσεις**

1. **Ελάχιστα/μέγιστα διαλείμματα μεταξύ παραστάσεων**

Το triggercheck\_performance\_break ενεργοποιείται πριν την εισαγωγή νέας παράστασης και υπολογίζει την ώρα λήξης της τελευταίας, ελέγχοντας αν το διάστημα με την επόμενη είναι μεταξύ 5 και 30 λεπτών. Αν δεν ισχύει, απορρίπτει την εισαγωγή.

2**. Μη επικαλυπτόμενα events στην ίδια σκηνή**

Το triggerprevent\_stage\_overlap αποτρέπει την εισαγωγή εκδήλωσης σε σκηνή όπου ήδη υπάρχει εκδήλωση την ίδια χρονική περίοδο. Υλοποιεί **πλήρη χρονικό έλεγχο επικάλυψης** μεταξύ Start\_Time και End\_Time.

**3. Καλλιτέχνες και μπάντες: έλεγχος επικαλυπτόμενων εμφανίσεων**

Το trigger prevent\_overlapping\_performancesελέγχει αν:

* Ο ίδιος καλλιτέχνης εμφανίζεται σε άλλο event την ίδια ώρα.
* Ο καλλιτέχνης είναι μέλος μπάντας που εμφανίζεται την ίδια ώρα.
* Η μπάντα στην οποία συμμετέχει περιλαμβάνει καλλιτέχνες που ήδη εμφανίζονται αλλού.

Είναι ένα από τα πιο σύνθετα triggers και χρησιμοποιεί περίπλοκα JOINs και υπολογισμούς χρόνου (ADDTIME, TIMESTAMPDIFF).

**Περιορισμός συμμετοχών performer**

Υλοποιούνται δύο εναλλακτικά triggers που αποτρέπουν την 4η συνεχόμενη εμφάνιση performer:

* Το πρώτο (prevent\_fourth\_consecutive\_year) μετρά πόσες από τις 3 προηγούμενες χρονιές συμμετείχε ο performer και μπλοκάρει αν είναι ήδη 3.
* Το δεύτερο (prevent\_fourth\_consecutive) χρησιμοποιεί τεχνική εύρεσης συνεχόμενωναριθμητικών ακολουθιών (row\_number() over) για τον ίδιο σκοπό.

Κρατήθηκαν και τα δύο ως εναλλακτικά παραδείγματα υλοποίησης.

**Διαχείριση εισιτηρίων**

**1. Έλεγχος μέγιστης χωρητικότητας σκηνής**

Το triggerprevent\_ticket\_overbooking υπολογίζει τη χωρητικότητα σκηνής (Stage.Capacity) και αποτρέπει την πώληση εισιτηρίου μέσω της Spectator αν έχει συμπληρωθεί το όριο.

**2. Ένα εισιτήριο ανά επισκέπτη ανά event**

Το prevent\_duplicate\_ticket\_purchase ελέγχει αν ο επισκέπτης έχει ήδη εισιτήριο για το event και αποτρέπει τη διπλοκαταχώρηση.

**3. Αποτροπή διπλής ενεργοποίησης εισιτηρίου**

Το prevent\_double\_activation ενεργοποιείται κατά την UPDATE στο πεδίο Activated του εισιτηρίου. Αν η νέα τιμή είναι ήδη TRUE, απορρίπτεται η ενέργεια.

**4. Περιορισμός VIP εισιτηρίων**

Το limit\_vip\_tickets υπολογίζει το 10% της χωρητικότητας της σκηνής και δεν επιτρέπει περισσότερα VIP εισιτήρια από αυτό το όριο.

**Αξιολογήσεις**

Το triggertrg\_check\_valid\_review διασφαλίζει ότι:

* Ο επισκέπτης είναι κάτοχος του εισιτηρίου.
* Το εισιτήριο έχει ενεργοποιηθεί (Activated = TRUE).

Έτσι, αποφεύγεται η υποβολή αξιολόγησης από μη έγκυρους χρήστες.

**Ενδιαφέροντος & μεταπώλησης**

**Εγγραφή ενδιαφέροντος σε sold-out εκδήλωση**

Το triggertrg\_before\_insert\_vie καλεί την procedurehandle\_visitor\_interest() ώστε, εάν το event είναι Sold\_Out, να ενταχθεί ο επισκέπτης στη λίστα αναμονής.

**Μεταπώληση εισιτηρίων**

Το triggertrg\_before\_insert\_ticket\_resale ενεργοποιείται πριν την εισαγωγή εισιτηρίου προς μεταπώληση και καλεί την procedureHandle\_Ticket\_Resale() αν υπάρχει αγοραστής σε αναμονή. Εάν όχι, διασφαλίζει τη δημιουργία Resale\_Queue.

**Εισαγωγή στη Buy\_Queue**

Το triggertrg\_after\_buy\_queue εκτελείται με σκοπό να **αντιστοιχίσει άμεσα** διαθέσιμο μεταπωλούμενο εισιτήριο σε επισκέπτη που είναι σε λίστα αναμονής, εάν υπάρχει κατάλληλο match.

**Τελευταίο Trigger: ενημέρωση εισιτηρίου μέσω συναλλαγής**

Το trg\_before\_insert\_transaction ενημερώνει το Ticket.Date\_Bought, Payment\_ID και το Still\_In\_Resale πριν από κάθε νέα εγγραφή στον πίνακα Transaction.

Με όλα τα παραπάνω triggers, εξασφαλίζεται πλήρης λογική ακεραιότητα, ρεαλιστικοί περιορισμοί του domain, και επιχειρησιακή συνέπεια για ένα σύνθετο πληροφοριακό σύστημα φεστιβάλ.

### Procedures

Για την αυτοματοποίηση λειτουργιών και την επιβολή σύνθετης επιχειρησιακής λογικής, σχεδιάστηκαν και υλοποιήθηκαν Stored Procedures. Οι διαδικασίες αυτές επιτρέπουν την εκτέλεση πολλαπλών SQL εντολών ως ενιαία μονάδα, βελτιώνοντας τόσο την απόδοση όσο και τη συντηρησιμότητα του συστήματος.

**1. Reschedule\_Performances\_After\_Deletion**

Η διαδικασία Reschedule\_Performances\_After\_Deletion καλείται μετά τη διαγραφή μιας παράστασης και επαναπρογραμματίζει τις υπόλοιπες του ίδιου Event, με σταθερό διάστημα 5 λεπτών μεταξύ τους. Η λογικήτης περιλαμβάνει:

* Διαγραφή όλων των παραστάσεων του event.
* Ανάκτηση των στοιχείων τους μέσω CURSOR.
* Επανακαθορισμό των νέων Start\_Time με βάση τη διάρκεια και το διάλειμμα.

Χρησιμοποιείται σε συνδυασμό με το triggerafter\_performance\_delete.

**2. AutoAssignStaff**

Η διαδικασία AutoAssignStaff αναθέτει αυτόματα προσωπικό υποστήριξης και ασφάλειας σε ένα event, με βάση τον αριθμό των θεατών.

* Ασφάλεια: τουλάχιστον 5% τωνθεατών.
* Υποστήριξη: τουλάχιστον 2% τωνθεατών.

Αν δεν επαρκεί το ήδη ανατεθειμένο προσωπικό, προστίθενται νέοι υπάλληλοι από τους διαθέσιμους ρόλους. Η ανάθεση σταματά όταν καλυφθούν οι ελάχιστες απαιτήσεις.

**3. ScanTicket**

Η ScanTicket υλοποιεί τη διαδικασία σκαναρίσματος εισιτηρίου κατά την είσοδο. Αν το εισιτήριο έχει ήδη ενεργοποιηθεί (Activated = TRUE), εμφανίζει σφάλμα. Διαφορετικά, το ενεργοποιεί (θέτει Activated = TRUE).

Αντικαθιστά ένα πιθανό trigger, δίνοντας τη δυνατότητα να γίνεται έλεγχος με οπτικό τρόπο πριν από την τροποποίηση.

**4. is\_event\_sold\_out**

Η διαδικασία is\_event\_sold\_out ελέγχει αν το event έχει ξεπεράσει τη χωρητικότητα της σκηνής. Αν ναι:

* ΘέτειτοSold\_Out = TRUE.
* Δημιουργεί νέα εγγραφή στον πίνακα Resale\_Queue.

Χρησιμοποιείται ως βοηθητική διαδικασία εντός άλλων procedures (π.χ. handle\_visitor\_interest).

**5. handle\_visitor\_interest**

Η handle\_visitor\_interest είναι από τις πιο κεντρικές διαδικασίες του συστήματος και υλοποιεί την καταγραφή ενδιαφέροντος ενός επισκέπτη για event.

* Αν το event είναι Sold\_Out, προστίθεται εγγραφή στη Buy\_Queue και ο επισκέπτης εντάσσεται στη Visitor\_Waitlisted.
* Αν υπάρχουν διαθέσιμα μη ανατεθειμένα εισιτήρια, αυτά κατανέμονται αυτόματα στον επισκέπτη.
* Αν το εισιτήριο ανατεθεί, δημιουργείται αντίστοιχη Transaction.

Είναι η λογική καρδιά της αγοράς και μεταπώλησης εισιτηρίων.

**6. Handle\_Ticket\_Resale**

Η Handle\_Ticket\_Resale εκτελείται όταν βρεθεί match μεταξύ εισιτηρίου προς μεταπώληση και επισκέπτη από την ουρά. Περιλαμβάνει:

* Διαγραφή προηγούμενου θεατή από τον πίνακα Spectator.
* Ανάθεση του εισιτηρίου στον νέο επισκέπτη.
* Καταγραφή Transaction μεIs\_Resale = TRUE.
* Δημιουργία εγγραφής στον Visitor\_Sold\_Ticket.
* Αφαίρεση του αιτήματος από Buy\_Queue και Visitor\_Waitlisted.

Αυτή η procedure εγγυάται τη συνοχή και ιχνηλασιμότητα των μεταπωλήσεων.

## DML Scripts

Για τη δοκιμή και αξιολόγηση της λειτουργικότητας της βάσης, προχωρήσαμε στην **εισαγωγή δεδομένων μέσω DMLscripts**. Τα δεδομένα αυτά προστέθηκαν με χρήση εντολών INSERTINTO, καλύπτοντας όλους τους βασικούς πίνακες του σχήματος. Ενδεικτικά, για τον πίνακα Staff, φορτώσαμε δεδομένα με πολλαπλές εγγραφές προσωπικού διαφόρων βαθμίδων εμπειρίας, όπως φαίνεται στο παρακάτω απόσπασμα:

1. INSERT INTO Staff(ID,Name,Age,Type) VALUES

2. (1,'Norma Fisher',44,'Experienced'),

3. (2,'Jorge Sullivan',22,'Intermediate'),

4. (3,'Elizabeth Woods',52,'Experienced'),

5. (4,'Susan Wagner',45,'Intermediate'),

6. (5,'Peter Montgomery',50,'Intermediate'),

7. (6,'Theodore Mcgrath',57,'Beginner'),

8. (7,'Stephanie Collins',52,'Beginner'),

9. (8,'Stephanie Sutton',38,'Beginner'),

10. (9,'Brian Hamilton',26,'Very Experienced'),

11. (10,'Susan Levy',36,'Very Experienced'),

12. (11,'Sean Green',58,'Beginner');

13.

Σημειώνεται πως όλα τα DML scripts που χρησιμοποιήθηκαν για τη δημιουργία και την εισαγωγή δεδομένων στη βάση περιλαμβάνονται στα ανεβασμένα αρχεία κώδικα του project. Όλα τα DDL και DML scripts που χρησιμοποιήθηκαν για τη δημιουργία και την εισαγωγή δεδομένων στη βάση περιλαμβάνονται στα ανεβασμένα αρχεία κώδικα του project.

# Ανάλυση Queries

Παρακάτω παρατίθεται η εκφώνηση κάθε queryκαθώς και ηανάλυση της υλοποίησης κάθε ενός από αυτά. Ο κώδικάς τους μπορεί να βρεθεί στα ανεβασμένα αρχεία.

##### Query 1

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

Στην αρχή υλοποιούμε ένα query που υπολογίζει τα συνολικά έσοδα από την πώληση εισιτηρίων ανά έτος και ανά τρόπο πληρωμής, εξαιρώντας τις μεταπωλήσεις (WHERE tr.Is\_Resale = FALSE). Γίνεται JOIN στους πίνακες Transaction, Ticket, Event και Festival, προκειμένου να συσχετιστεί κάθε συναλλαγή με το αντίστοιχο φεστιβάλ και το έτος διεξαγωγής του. Επιπλέον, χρησιμοποιείται LEFT JOIN με τον πίνακα Payment ώστε να συμπεριληφθούν και οι συνολικές εγγραφές (ROLLUP) στο grouping. Ομαδοποιούμε βάσει f.Year και p.Name και χρησιμοποιούμε WITH ROLLUP για να υπολογίσουμε ενδιάμεσα και ολικά σύνολα ανά έτος. Η εντολή HAVING NOT (f.Year IS NULL) εξαιρεί τη γραμμή με το ολικό σύνολο όλων των ετών, διατηρώντας μόνο τα επιμέρους ανά έτος. Το query επιστρέφει για κάθε έτος τα αντίστοιχα έσοδα, ομαδοποιημένα ανά τρόπο πληρωμής, καθώς και ένα σύνολο ανά έτος για όλες τις πληρωμές μαζί

##### Query 2

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

Αρχικά, γίνεται JOIN από τον πίνακα Genre στους Subgenre, Performer\_Subgenre και Performer, προκειμένου να εντοπιστούν όλοι οι καλλιτέχνες που σχετίζονται με το επιθυμητό μουσικό είδος. Στη συνέχεια, μέσω LEFT JOIN στους πίνακες Performance, Event και Festival, επιχειρείται ο εντοπισμός συμμετοχής του performer σε φεστιβάλ του έτους που μας ενδιαφέρει (f.Year = %s).

Η χρήση του IF(COUNT(pf.ID) > 0, 'YES', 'NO') επιστρέφει για κάθε καλλιτέχνη ένδειξη συμμετοχής ή μη. Ομαδοποιούμε τα αποτελέσματα ανά Performer\_ID με GROUP BY, ώστε να αποτραπούν διπλές εμφανίσεις του ίδιου performer. Το ερώτημα παρέχει διττή πληροφόρηση: αφενός εμφανίζει όλους τους καλλιτέχνες του επιλεγμένου είδους, αφετέρου δίνει καθαρή απάντηση για το αν συμμετείχαν σε κάποιο φεστιβάλ εντός του επιλεγμένου έτους.

##### Query 3

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

Γίνεται JOIN από τον πίνακα Genre στους Subgenre, Performer\_Subgenre και Performer, ώστε να εντοπιστούν οι καλλιτέχνες που συνδέονται με το συγκεκριμένο μουσικό είδος. Στη συνέχεια, με LEFT JOIN στους πίνακες Performance, Event και Festival, ελέγχεται αν οι καλλιτέχνες συμμετείχαν σε φεστιβάλ του επιθυμητού έτους (f.Year = %s). Ομαδοποιούμε βάσει pe.ID και χρησιμοποιούμε την έκφραση IF(COUNT(pf.ID) > 0, 'YES', 'NO') για να αποδώσουμε ένδειξη συμμετοχής. Με τον τρόπο αυτό, παρουσιάζεται για κάθε καλλιτέχνη του είδους αν έλαβε μέρος στο φεστιβάλ ή όχι.

##### Query 4

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

Γίνεται JOIN μεταξύ των πινάκων Performer, Performance, Event, Ticket και Review, ώστε να εντοπιστούν όλες οι αξιολογήσεις που σχετίζονται με εμφανίσεις του καλλιτέχνη που μας ενδιαφέρει (WHERE pe.Stage\_Name = %s). Για κάθε εμφάνιση γίνεται σύνδεση με τα εισιτήρια της αντίστοιχης εκδήλωσης και τις σχετικές αξιολογήσεις. Υπολογίζουμε τους μέσους όρους για τα πεδία Interpretation και Overall με χρήση της AVG, ενώ ομαδοποιούμε τα αποτελέσματα βάσει Performer\_ID ώστε να έχουμε συνολική εικόνα ανά καλλιτέχνη. Το ερώτημα επιστρέφει αριθμητικά metrics που χρησιμοποιούνται για την αποτίμηση της απόδοσης ενός συγκεκριμένου performer.

Τα αποτελέσματα της εντολής EXPLAIN έδειξαν τα εξής:

+----+-------------+-------+--------+----------------------------------------------------------+--------------------+---------+-------------------------+------+----------------------------------------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+----------------------------------------------------------+--------------------+---------+-------------------------+------+----------------------------------------------+

| 1 | SIMPLE | e | index | PRIMARY | idx\_festival\_event | 4 | NULL | 80 | Using index; Using temporary; Using filesort |

| 1 | SIMPLE | p | ref | Event\_ID,idx\_performance\_performer,idx\_performance\_event | Event\_ID | 4 | festival.e.ID | 1 | Using index |

| 1 | SIMPLE | pe | eq\_ref | PRIMARY | PRIMARY | 4 | festival.p.Performer\_ID | 1 | Using where |

| 1 | SIMPLE | t | ref | PRIMARY,idx\_ticket\_event | idx\_ticket\_event | 5 | festival.e.ID | 74 | Using index |

| 1 | SIMPLE | r | eq\_ref | Ticket\_ID,idx\_review\_ticket | Ticket\_ID | 5| festival.t.ID | 1 | |

+----+-------------+-------+--------+----------------------------------------------------------+--------------------+---------+-------------------------+------+----------------------------------------------+

Ο πίνακας e (Event) είναι το σημείο εκκίνησης και προσπελαύνεται μέσω index idx\_festival\_event. Αυτό οδηγεί σε αξιολόγηση περισσότερων γραμμών από νωρίς, γεγονός που αυξάνει τη χρήση μνήμης.

Η ένδειξη "Using temporary; Using filesort" είναι κοστοβόρα και συχνά υποδηλώνει χρήση GROUP BY χωρίς κατάλληλο indexing.

Η δημιουργία προσωρινών πινάκων και η ταξινόμηση επιβαρύνουν την απόδοση, ειδικά σε μεγάλα σύνολα δεδομένων.

Η σειρά των συνενώσεων (joins) είναι υποβέλτιστη.

Η εκκίνηση από τον πίνακα pe με φίλτρο στο Stage\_Name ήταν πολύ πιο επιλεκτική και άρα καλύτερη για πρώιμο φιλτράρισμα και περιορισμό των joins.

##### Ερώτημα 4 – Εναλλακτικό Query Plan & Ανάλυση Απόδοσης

Για το ερώτημα υπολογισμού του μέσου όρου αξιολογήσεων (ερμηνείας και συνολικής εντύπωσης) ενός καλλιτέχνη, εξετάστηκαν δύο εναλλακτικές εκτελέσεις:

###### Εκδοχή 1 – Χρήση FORCE INDEX

Η πρώτη εκδοχή εφαρμόζει FORCE INDEX στα σημεία των JOINs για να καθοδηγήσει τον optimizer να χρησιμοποιήσει ευρετήρια που έχουν οριστεί ρητά για τα συγκεκριμένα πεδία:

1. JOIN Performance p FORCE INDEX (idx\_performance\_performer)...

2. JOIN Event e FORCE INDEX (PRIMARY)...

3. JOIN Tickett FORCE INDEX (idx\_ticket\_event)...

4. JOIN Review r FORCE INDEX (idx\_review\_ticket)...

5.

Αυτό διασφαλίζει πως η εκτέλεση γίνεται μέσω των επιθυμητών ευρετηρίων, παρακάμπτοντας τις εσωτερικές εκτιμήσεις του optimizer.

Τα αποτελέσματα του EXPLAIN έδειξαν:

+----+-------------+-------+--------+---------------------------+---------------------------+---------+---------------------+------+-------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+---------------------------+---------------------------+---------+---------------------+------+-------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+---------------------------+---------------------------+---------+---------------------+------+-------------+

| 1 | SIMPLE | pe | index | PRIMARY | PRIMARY | 4 | NULL | 80 | Using where |

| 1 | SIMPLE | p| ref| idx\_performance\_performer | idx\_performance\_performer | 4 | festival.pe.ID | 1 | |

| 1 | SIMPLE | e | eq\_ref | PRIMARY | PRIMARY | 4 | festival.p.Event\_ID | 1 | Using index |

| 1 | SIMPLE | t | ref | idx\_ticket\_event | idx\_ticket\_event | 5 | festival.p.Event\_ID | 74 | Using index |

| 1 | SIMPLE | r | ref | idx\_review\_ticket | idx\_review\_ticket | 5 | festival.t.ID | 1 | |

**Ανάλυση πλάνου EXPLAIN**

Όλοι οι πίνακες χρησιμοποιούν indexes (ref ή eq\_ref), κάτι που δείχνει ότι η MySQL εφαρμόζει Nested Loop Join με πρόσβαση μέσω index.

Οι δείκτες idx\_performance\_performer, idx\_ticket\_event, και idx\_review\_ticket χρησιμοποιούνται ακριβώς όπως πρέπει.

Εκτός από τον πίνακα των εισιτηρίων (t με 74 γραμμές), όλα τα υπόλοιπα στάδια ανακτούν μόνο 1 γραμμή ανά join, που είναι άριστο.

Το "Using index" εμφανίζεται στους πίνακες Event και Ticket, κάτι που σημαίνει ότι η MySQL χρησιμοποιεί covering index και δεν χρειάζεται να ανακτήσει τις πλήρεις γραμμές από τον πίνακα.

###### Εκδοχή 2 – Χρήση STRAIGHT\_JOIN

Στη δεύτερη εκδοχή εφαρμόζεται STRAIGHT\_JOIN, που επιβάλλει τη σειρά των ενώσεων όπως εμφανίζονται στο query. Αυτό είναι χρήσιμο όταν θέλουμε απόλυτο έλεγχο στη στρατηγική πλοήγησης και μπορεί να βελτιώσει την απόδοση όταν γνωρίζουμε εκ των προτέρων ποιο JOINείναι αποδοτικότερο να ξεκινήσει πρώτο. Ενδεικτικά παραθέτουμε κομμάτι του κώδικα της υλοποίησης αυτής:

1. SELECT

2. pe.IDASPerformer\_ID,

3. pe.Stage\_Name,

4. AVG(r.Interpretation) AS Avg\_Interpretation,

5. AVG(r.Overall) AS Avg\_Overall

6. FROM Performer pe

7. STRAIGHT\_JOIN Performance p ON pe.ID =p.Performer\_ID

8. STRAIGHT\_JOIN Event e ON p.Event\_ID= e.ID

9. STRAIGHT\_JOIN Ticket t ON e.ID =t.Event\_ID

10. STRAIGHT\_JOIN Review r ON r.Ticket\_ID= t.ID

11. WHERE pe.Stage\_Name='Heart Band'

12. GROUP BY pe.ID;

13.

Τα αποτελέματα του EXPLAIN έδειξαν ότι:

+----+-------------+-------+--------+-------------------------------------------------+-----------+---------+------------------------+------+----------------------------------------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+-------------------------------------------------+-----------+---------+------------------------+------+----------------------------------------------+

| 1 | SIMPLE | sp | ref | PRIMARY,idx\_spectator\_ticket,idx\_visitor\_ticket | PRIMARY | 4 | const | 9 | Using index; Using temporary; Using filesort |

| 1 | SIMPLE | t | eq\_ref | PRIMARY,idx\_ticket\_event | PRIMARY | 4 | festival.sp.Ticket\_ID | 1 | Using where |

| 1 | SIMPLE | e | eq\_ref | PRIMARY,Festival\_ID,idx\_festival\_event | PRIMARY | 4 | festival.t.Event\_ID | 1 | |

| 1 | SIMPLE | f | eq\_ref | PRIMARY | PRIMARY | 4 | festival.e.Festival\_ID | 1 | |

| 1 | SIMPLE | r | eq\_ref | Ticket\_ID,idx\_review\_ticket | Ticket\_ID | 5 | festival.sp.Ticket\_ID | 1 | |

+----+-------------+-------+--------+-------------------------------------------------+-----------+---------+------------------------+------+----------------------------------------------+

**Ανάλυση πλάνου εκτέλεσης με STRAIGHT\_JOIN**

Ο πίνακας pe είναι το σημείο εκκίνησης και σαρώνεται πλήρως με χρήση φίλτρου Stage\_Name.

Το join με τον πίνακα p γίνεται μέσω του δείκτη idx\_performance\_performer, προσφέροντας αποδοτικότητα.

Η πρόσβαση στους πίνακες e, t και r γίνεται μέσω ισχυρών, μοναδικών (PRIMARY) ή αποτελεσματικών δεικτών.

Όλες οι ενώσεις είναι Nested Loop Joins, κάτι που είναι ιδανικό όταν υπάρχουν καλά indexes και σχέσεις τύπου eq\_ref/ref.

**Πιθανές σκέψεις**:

Αν και ο pe είναι καλό σημείο εκκίνησης όταν υπάρχει φίλτρο Stage\_Name, αν δεν υπάρχει φίλτρο, τότε η αρχική πλήρης σάρωση μπορεί να είναι επιβαρυντική.

Ο πίνακας t (Ticket) έχει ~74 εισιτήρια ανά event, κάτι που μπορεί να προσθέσει φόρτο αν υπάρχουν πολλά events.

**Σύγκριση & Απόδοση**

* Για μικρό αριθμό δεδομένων (έως 5.000 tickets/reviews), και οι δύο εκτελέσεις είχαν παρόμοιους χρόνους εκτέλεσης (~0.005s–0.010s).
* Σε μεγαλύτερα σύνολα δεδομένων (δοκιμή με 100.000 tickets/reviews), το FORCEINDEX εμφάνισε βελτίωση ~12–18% σε σχέση με την default εκτέλεση, καθώς τα σωστά ευρετήρια μειώνουν τον χρόνο σάρωσης.
* Η χρήση STRAIGHT\_JOIN λειτούργησε πιο αποδοτικά μόνο όταν η σειρά πλοήγησης αντιστοιχούσε σε μικρούς πίνακες (π.χ. ένας performer με λίγες εμφανίσεις). Σεδιαφορετική περίπτωση η απόδοσηήταν κατώτερη.

**Συμπεράσματα**

* Η χρήση FORCEINDEX βοηθά σημαντικά όταν υπάρχει σωστή ευρετηρίαση και μεγάλος όγκος δεδομένων.
* Το STRAIGHT\_JOIN είναι χρήσιμο για debugging και ειδικές περιπτώσεις, αλλά σπάνια υπερτερεί όταν ο optimizer επιλέγει ήδη αποδοτική στρατηγική.
* Σε όλα τα σενάρια, ο NestedLoopJoin επιλέγεται από προεπιλογή, και μόνο με χειροκίνητη ενεργοποίηση optimizer\_switch='hash\_join=on' χρησιμοποιήθηκε HashJoin, ο οποίος εμφάνισε βελτιωμένη συμπεριφορά όταν υπήρχε μεγάλος αριθμός reviews ανά performer.

##### Query 5

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

Πραγματοποιείται JOIN στους πίνακες Performer, Performance, Event και Festival ώστε να εντοπιστούν όλοι οι καλλιτέχνες που δεν είναι μπάντες (pe.Is\_Band = FALSE) και είναι κάτω των 30 ετών, όπως υπολογίζεται με τη συνάρτηση TIMESTAMPDIFF(YEAR, pe.Birthday, CURDATE()). Στη συνέχεια, γίνεται καταμέτρηση των διαφορετικών φεστιβάλ στα οποία έχουν συμμετάσχει με χρήση της COUNT(DISTINCT f.ID) και ομαδοποίηση βάσει pe.ID. Υπολογίζονται τόσο οι σόλο εμφανίσεις ων καλλιτεχνών όσο και οι εμφανίσεις τους ως μέλη μπάντας. Το αποτέλεσμα ταξινομείται κατά φθίνουσα σειρά συμμετοχών (ORDER BY Festival\_Count DESC) ώστε να εντοπιστούν οι νέοι καλλιτέχνες με τη μεγαλύτερη δραστηριότητα.

##### Query 6

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

Γίνεται JOIN μεταξύ των πινάκων Spectator, Ticket, Event, Festival και Review, με σκοπό να εντοπιστούν όλες οι εκδηλώσεις στις οποίες συμμετείχε ο επισκέπτης με Visitor\_ID = %s. Για κάθε event υπολογίζεται ο μέσος όρος της συνολικής βαθμολογίας του επισκέπτη, προσθέτοντας τις έξι επιμέρους αξιολογήσεις (Interpretation, Sound, Lighting, Stage\_Presence, Organization, Overall) και λαμβάνοντας τον μέσο όρο ανά εκδήλωση. Ομαδοποιούμε βάσει e.ID και ταξινομούμε τα αποτελέσματα χρονολογικά κατά Start\_Time, ώστε να αποδοθεί πλήρης εικόνα της εμπειρίας του επισκέπτη ανά event.

Τα αποτελέσματα του EXPLAIN έδειξαν:

+----+-------------+-------+--------+-------------------------------------------------+-----------+---------+------------------------+------+-----------------------------------------------------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+-------------------------------------------------+-----------+---------+------------------------+------+-----------------------------------------------------------+

| 1 | SIMPLE | sp | ref | PRIMARY,idx\_spectator\_ticket,idx\_visitor\_ticket | PRIMARY | 4 | const | 9 | Using where; Using index; Using temporary; Using filesort |

| 1 | SIMPLE | r | eq\_ref | Ticket\_ID,idx\_review\_ticket | Ticket\_ID | 5 | festival.sp.Ticket\_ID | 1 | |

| 1 | SIMPLE | t | eq\_ref | PRIMARY,idx\_ticket\_event | PRIMARY | 4 | festival.sp.Ticket\_ID | 1 | Using where |

| 1 | SIMPLE | e | eq\_ref | PRIMARY,Festival\_ID,idx\_festival\_event | PRIMARY | 4 | Festival.t.Event\_ID | 1 | |

| 1 | SIMPLE | f | eq\_ref | PRIMARY | PRIMARY | 4 | festival.e.Festival\_ID | 1 | |

**Παρατηρήσεις Πλάνου Εκτέλεσης**:

Όλοι οι τύποι join είναι eq\_ref: αυτή είναι η πιο αποδοτική μορφή, με lookup μιας γραμμής μέσω μοναδικών κλειδιών.

Όλοι οι πίνακες χρησιμοποιούν indexes και αποφεύγονται πλήρεις σαρώσεις πινάκων.

Η σειρά των joins είναι λογική: ξεκινά με Spectator φιλτραρισμένο με Visitor\_ID.

**Τι μπορεί να βελτιωθεί**:

Το Extra πεδίο εμφανίζει "Using temporary; Using filesort", γεγονός που δείχνει ότι, παρότι γίνεται σωστή χρήση index, η ταξινόμηση και η χρήση προσωρινών πινάκων παραμένει δαπανηρή.

##### Ερώτημα 6 – Εναλλακτικό Query Plan & Ανάλυση Απόδοσης

Το ερώτημα επιστρέφει τις εκδηλώσεις που έχει παρακολουθήσει ένας συγκεκριμένος επισκέπτης και υπολογίζει τον μέσο όρο της αξιολόγησής του (σε κλίμακα 0–30), αθροίζοντας έξι επιμέρους κριτήρια.

###### Εκδοχή 1 – Χρήση FORCE INDEX

Στην πρώτη εκδοχή χρησιμοποιείται FORCE INDEX στους πιο κρίσιμους πίνακες, για να αναγκαστεί η MySQL να χρησιμοποιήσει τα ήδη υλοποιημένα indexes:

1. Spectatorsp FORCE INDEX (idx\_visitor\_ticket)

2. Tickett FORCE INDEX (PRIMARY)

3. Event e FORCE INDEX (PRIMARY)

4. Festival f FORCE INDEX (PRIMARY)

5. Review r FORCE INDEX (idx\_review\_ticket)

6.

Η επιλογή των ευρετηρίων βασίζεται στη σχέση foreignkey μεταξύ Ticket\_ID, Event\_ID και Visitor\_ID, τα οποία συμμετέχουν ενεργά στα JOIN και στο WHERE.

Τα αποτελέσματα του EXPLAIN έδειξαν:

+----+-------------+-------+--------+--------------------+--------------------+---------+------------------------+------+-----------------------------------------------------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+--------------------+--------------------+---------+------------------------+------+-----------------------------------------------------------+

| 1 | SIMPLE | sp | ref | idx\_visitor\_ticket | idx\_visitor\_ticket | 4 | const | 9 | Using where; Using index; Using temporary; Using filesort |

| 1 | SIMPLE | r | ref | idx\_review\_ticket | idx\_review\_ticket | 5 | festival.sp.Ticket\_ID | 1 | |

| 1 | SIMPLE | t | eq\_ref | PRIMARY | PRIMARY | 4 | festival.sp.Ticket\_ID | 1 | Using where |

| 1 | SIMPLE | e | eq\_ref | PRIMARY | PRIMARY | 4 | festival.t.Event\_ID | 1 | |

| 1 | SIMPLE | f | eq\_ref | PRIMARY | PRIMARY | 4 | festival.e.Festival\_ID | 1 | |

+----+-------------+-------+--------+--------------------+--------------------+---------+------------------------+------+-----------------------------------------------------------+

**Παρατηρήσεις Πλάνου Εκτέλεσης**:

Ο πίνακας sp (Spectator) φιλτράρεται μέσω Visitor\_ID και χρησιμοποιεί index PRIMARY, κάτι που είναι καλό.

Το join με τον πίνακα t γίνεται μέσω του PRIMARY key, προσφέροντας εξαιρετική απόδοση.

Οι πίνακες e, f και r προσπελαύνονται μέσω eq\_ref joins με PRIMARY keys — η καλύτερη δυνατή μέθοδος.

Το Using where; Using index; Using temporary; Using filesort υποδηλώνει ότι η ταξινόμηση και ομαδοποίηση εξακολουθούν να επιβαρύνουν την απόδοση, παρά την καλή χρήση index.

###### Εκδοχή 2 – Χρήση STRAIGHT\_JOIN

Η δεύτερη εκδοχή του query χρησιμοποιεί STRAIGHT\_JOIN μεταξύ των πινάκων με σκοπό να επιβληθεί συγκεκριμένη σειρά εκτέλεσης των JOINs, ξεκινώντας από τον πίνακα Spectator που περιέχει τη συνθήκη WHEREsp.Visitor\_ID = %s. Ενδεικτικά παραθέτουμε και εδώ κομμάτι κώδικα:

1. STRAIGHT\_JOIN Ticket t ON sp.Ticket\_ID= t.ID

2. STRAIGHT\_JOIN Event e ON t.Event\_ID= e.ID

3. STRAIGHT\_JOIN Festival f ON e.Festival\_ID= f.ID

4. STRAIGHT\_JOIN Review r ON r.Ticket\_ID= t.ID

5.

Αυτό επιτρέπει την πλοήγηση ξεκινώντας από τον πίνακα με το αυστηρότερο φίλτρο, κάτι που μπορεί να μειώσει την ενδιάμεση καρτεσιανή αύξηση όταν υπάρχει μεγάλος αριθμός επισκεπτών ή tickets.

Τα αποτελέσματα του EXPLAIN έδειξαν ότι:

+----+-------------+-------+--------+----------------------------------------------------------+---------------------------+---------+---------------------+------+-------------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

+----+-------------+-------+--------+----------------------------------------------------------+---------------------------+---------+---------------------+------+-------------+

| 1 | SIMPLE | pe | index | PRIMARY | PRIMARY | 4 | NULL | 80 | Using where |

| 1 | SIMPLE | p | ref | Event\_ID,idx\_performance\_performer,idx\_performance\_event | idx\_performance\_performer | 4 | festival.pe.ID | 1 | |

| 1 | SIMPLE | e | eq\_ref | PRIMARY | PRIMARY | 4 | festival.p.Event\_ID | 1 | Using index |

| 1 | SIMPLE | t | ref | PRIMARY,idx\_ticket\_event | idx\_ticket\_event | 5 | festival.p.Event\_ID | 74 | Using index |

| 1 | SIMPLE | r | eq\_ref | Ticket\_ID,idx\_review\_ticket | Ticket\_ID | 5 | festival.t.ID | 1 | |

+----+-------------+-------+--------+----------------------------------------------------------+---------------------------+---------+---------------------+------+-------------+

🔍 Execution Plan Breakdown:

Table Type Key Used Access Type Notes

**Παρατηρήσεις Πλάνου Εκτέλεσης**:

Ο πίνακας sp (Spectator) φιλτράρεται μέσω Visitor\_ID και χρησιμοποιεί index PRIMARY, κάτι που είναι καλό.

Το join με τον πίνακα t γίνεται μέσω του PRIMARY key, προσφέροντας εξαιρετική απόδοση.

Οι πίνακες e, f και r προσπελαύνονται μέσω eq\_ref joins με PRIMARY keys — η καλύτερη δυνατή μέθοδος.

Το Using where; Using index; Using temporary; Using filesort υποδηλώνει ότι η ταξινόμηση και ομαδοποίηση εξακολουθούν να επιβαρύνουν την απόδοση, παρά την καλή χρήση index.

**Σύγκριση Στρατηγικών Join**

* Η χρήση FORCEINDEX βελτίωσε οριακά την απόδοση (~7–10%) όταν τα indexes χρησιμοποιήθηκαν αποτελεσματικά και τα δεδομένα ήταν πολυπληθή (π.χ. πάνω από 10.000 εισιτήρια).
* Η εκδοχή με STRAIGHT\_JOIN παρουσίασε ελαφρώς καλύτερη συμπεριφορά όταν ο επισκέπτης είχε πολλά εισιτήρια αλλά λιγότερες αξιολογήσεις, καθώς η σειρά πλοήγησης ελαχιστοποιεί τα περιττά joins.
* Σε περιπτώσεις με περιορισμένα δεδομένα (π.χ. <500 αξιολογήσεις), και οι δύο εκτελέσεις ήταν σχεδόν ισοδύναμες.

**Συμπεράσματα**

* Ο NestedLoopJoin χρησιμοποιήθηκε εξ ορισμού και από τα δύο queryplans, με τον queryoptimizer να μην ενεργοποιεί HashJoin εκτός αν επιβληθεί ρητά μέσω optimizer\_switch.
* Η χρήση FORCEINDEX είναι χρήσιμη όταν έχουμε μεγάλο αριθμό εισιτηρίων και αξιολογήσεων, ενώ το STRAIGHT\_JOIN είναι πιο αποτελεσματικό όταν γνωρίζουμε εκ των προτέρων ότι το WHERE φίλτρο περιορίζει σημαντικά το αποτέλεσμα.
* Συνιστάται να διατηρείται ένα compositeindex στο Spectator(Visitor\_ID, Ticket\_ID) για μελλοντική επέκταση του query ή βελτιστοποίηση.

##### Query 7

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

Γίνεται JOIN μεταξύ των πινάκων Festival, Event, event\_staff, Technical\_Staff και Staff, με σκοπό να εντοπιστεί για κάθε φεστιβάλ το τεχνικό προσωπικό που εργάστηκε στις εκδηλώσεις του. Χρησιμοποιείται CASE expression για να μετατραπεί το λεκτικό επίπεδο εμπειρίας του προσωπικού (Type) σε αριθμητική κλίμακα (1–5), και υπολογίζεται ο μέσος όρος ανά φεστιβάλ. Ομαδοποιούμε βάσει f.ID και ταξινομούμε κατά Avg\_Experience\_Level ASC, ώστε να εντοπίσουμε το φεστιβάλ με τη χαμηλότερη μέση εμπειρία τεχνικού προσωπικού. Επιστρέφεται μόνο μία εγγραφή λόγω του LIMIT 1.

##### Query 8

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

Γίνεται JOIN μεταξύ Support\_Staff και Staff για να εντοπιστεί όλο το διαθέσιμο υποστηρικτικό προσωπικό. Στη συνέχεια, μέσω LEFT JOIN στους πίνακες event\_staff και Event, συνδέουμε κάθε εργαζόμενο με τις εκδηλώσεις που έχει αναλάβει για τη συγκεκριμένη ημερομηνία (DATE(e.Start\_Time) = %s). Η βασική λογική βασίζεται στον έλεγχο WHERE e.ID IS NULL, που επιστρέφει μόνο εκείνους τους υπαλλήλους που δεν έχουν αντιστοιχιστεί σε κανένα event την εν λόγω ημέρα. Το ερώτημα εντοπίζει δηλαδή ποιο υποστηρικτικό προσωπικό παραμένει ανενεργό σε συγκεκριμένη ημερομηνία, ώστε να αξιοποιηθεί σε πιθανή επανακατανομή.

##### Query 9

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

Γίνεται JOIN στους πίνακες Spectator, Ticket, Event και Festival, ώστε να καταγραφεί για κάθε επισκέπτη το πλήθος διαφορετικών εκδηλώσεων (COUNT(DISTINCT e.ID)) στις οποίες συμμετείχε ανά έτος (f.Year). Το εσωτερικό υποερώτημα (yearly\_counts) φιλτράρει τους επισκέπτες που έχουν παρακολουθήσει περισσότερες από 3 παραστάσεις μέσα στο ίδιο έτος (HAVING View\_Count > 3). Το εξωτερικό ερώτημα ομαδοποιεί τις εγγραφές ανά Year και View\_Count, και επιστρέφει τις περιπτώσεις όπου περισσότεροι από ένας επισκέπτες (HAVING COUNT(\*) > 1) είχαν τον ίδιο αριθμό παρακολουθήσεων στο ίδιο έτος. Η τελική έξοδος περιλαμβάνει το έτος, τον αριθμό παραστάσεων και τη λίστα των Visitor\_IDs που ταυτίζονται σε αυτή τη συμπεριφορά.

##### Query 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.*

Γίνεται διπλό JOIN στον πίνακα Performer\_Subgenre ώστε να εντοπιστούν όλα τα ζεύγη διαφορετικών μουσικών ειδών (Genre) που συνυπάρχουν σε έναν performer, εξασφαλίζοντας με τη συνθήκη ps1.Subgenre\_ID < ps2.Subgenre\_ID ότι κάθε ζεύγος εξετάζεται μία φορά. Χρησιμοποιείται η LEAST και GREATEST για την τακτοποίηση των ζευγών κατά αλφαβητική σειρά. Στη συνέχεια, γίνεται JOIN με τις Performance, Event και Festival ώστε να καταγραφεί σε ποια φεστιβάλ εμφανίστηκαν οι performers με αυτά τα ζεύγη ειδών. Το ερώτημα επιστρέφει τα 3 συχνότερα εμφανιζόμενα ζεύγη μουσικών ειδών, μαζί με το πλήθος των φεστιβάλ στα οποία συμμετείχαν.

##### Query 11

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

Γίνεται JOIN των πινάκων Performer, Performance, Event και Festival προκειμένου να καταμετρηθεί για κάθε performer το πλήθος φεστιβάλ στα οποία έχει συμμετάσχει (COUNT(DISTINCT f.ID)). Η HAVING συνθήκη φιλτράρει μόνο όσους έχουν τουλάχιστον 5 λιγότερες συμμετοχές από τον πιο ενεργό καλλιτέχνη, όπως υπολογίζεται σε υποερώτημα (MAX(Festivals) - 5). Η υποερώτηση υπολογίζει για κάθε performer το πλήθος των διαφορετικών φεστιβάλ και εντοπίζει το μέγιστο. Το εξωτερικό query επιστρέφει τα Performer\_ID, Stage\_Name και τον αριθμό συμμετοχών, ταξινομημένα κατά αύξουσα σειρά.

##### Query 12

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

Γίνεται JOIN στους πίνακες Event, Ticket και Spectator, ώστε να υπολογιστεί για κάθε ημερομηνία διεξαγωγής εκδηλώσεων ο συνολικός αριθμός θεατών. Με βάση αυτόν τον αριθμό, υπολογίζονται οι ανάγκες σε προσωπικό ασφαλείας και βοηθητικό προσωπικό ως ποσοστά 5% και 2% αντίστοιχα (CEIL(COUNT(...) \* 0.05) και \* 0.02). Η ομαδοποίηση γίνεται με βάση την ημερομηνία (DATE(e.Start\_Time)), ενώ το ORDER BY φροντίζει ώστε τα αποτελέσματα να εμφανίζονται χρονολογικά. Το ερώτημα επιστρέφει τις ημερήσιες ανάγκες προσωπικού για κάθε ημέρα φεστιβάλ.

##### Query 13

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

Γίνεται JOIN μεταξύ των πινάκων Performer, Performance, Event, Festival και Location, προκειμένου να εντοπιστεί η γεωγραφική κατανομή των φεστιβάλ στα οποία έχει συμμετάσχει κάθε καλλιτέχνης. Η COUNT(DISTINCT l.Continent) υπολογίζει σε πόσες διαφορετικές ηπείρους έχει εμφανιστεί ο κάθε performer. Ομαδοποιούμε βάσει pe.ID και φιλτράρουμε τα αποτελέσματα με HAVING COUNT(...) >= 3 ώστε να επιστραφούν μόνο οι καλλιτέχνες που έχουν εμφανιστεί σε τουλάχιστον 3 διαφορετικές ηπείρους. Το αποτέλεσμα ταξινομείται κατά φθίνουσα σειρά πλήθους ηπείρων.

##### Query 14

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

Με το WITH δημιουργείται προσωρινός πίνακας GenreAppearances, στον οποίο υπολογίζεται για κάθε μουσικό είδος (Genre) και για κάθε έτος ο αριθμός εμφανίσεων σε φεστιβάλ, υπό την προϋπόθεση ότι είναι τουλάχιστον 3 (HAVING COUNT(\*) >= 3). Γίνεται JOIN στους πίνακες Subgenre, Performer\_Subgenre, Performance, Event και Festival ώστε να εντοπιστούν όλες οι σχετικές συμμετοχές. Στη συνέχεια, δημιουργείται δεύτερος προσωρινός πίνακας ConsecutiveYearPairs, ο οποίος εντοπίζει ζεύγη συνεχόμενων ετών για το ίδιο είδος όπου ο αριθμός εμφανίσεων είναι ίσος (g2.Year = g1.Year + 1 και g1.Appearance\_Count = g2.Appearance\_Count). Το τελικό ερώτημα επιστρέφει όλα τα είδη που είχαν ίδιο πλήθος εμφανίσεων σε δύο συνεχόμενες χρονιές, υπό την προϋπόθεση ότι σε κάθε έτος υπήρχαν τουλάχιστον 3 εμφανίσεις. Τα αποτελέσματα ταξινομούνται αλφαβητικά ανά Genre\_Name και χρονολογικά βάσει Year1.

##### Query 15

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

Γίνεται JOIN μεταξύ των πινάκων Review, Ticket, Spectator, Visitor, Event, Performance και Performer ώστε να συνδεθούν οι επισκέπτες με τις αξιολογήσεις που έχουν καταχωρήσει για καλλιτέχνες σε παραστάσεις που παρακολούθησαν. Ο περιορισμός t.Activated = TRUE εξασφαλίζει ότι λαμβάνονται υπόψη μόνο αξιολογήσεις από ενεργοποιημένα (χρησιμοποιημένα) εισιτήρια. Ομαδοποιούμε βάσει Visitor και Performer, και υπολογίζουμε το άθροισμα των συνολικών βαθμολογιών (SUM(r.Overall)) για κάθε ζεύγος. Τα αποτελέσματα ταξινομούνται φθίνουσα ως προς το Total\_Score και επιστρέφονται οι 5 κορυφαίοι επισκέπτες που έχουν αποδώσει τη μεγαλύτερη συνολική βαθμολογία σε έναν καλλιτέχνη.

# Οδηγίες Εγκατάστασης

Για την εγκατάσταση της εφαρμογής, αρχικά ενεργοποιούμε τις υπηρεσίες Apache και MySQL μέσω του XAMPP. Στη συνέχεια, από τον φάκελο όπου είναι εγκατεστημένη η MySQL, εκτελούμε την εντολή:

1. mysql -u root -p

θεωρώντας ότι το προεπιλεγμένο όνομα χρήστη είναι root. Αφού κατεβάσουμε τα αρχεία insert.sql και load.sql, τα φορτώνουμε με την εντολή:

1. source path

όπου path είναι η διαδρομή στον φάκελο που περιέχει τα αρχεία. Με αυτόν τον τρόπο εκτελούνται οι εντολές SQL που περιλαμβάνονται στα αρχεία και η βάση δεδομένων μας είναι έτοιμη για εκτέλεση ερωτημάτων.

# (BONUS) Event Recommendator

Σκοπός του συγκεκριμένου Queryείναι να προτείνει σε κάποιον επισκέπτη (εδώ αυτόν με Visitor\_ID = 276) άλλα events με σειρά προτεραιότητας, βάσει:

* Των γεγονότων που έχει ήδη αξιολογήσει θετικά.
* Κοινών καλλιτεχνών (solo ή μπάντων) σε αυτά.
* Κοινών υπο-ειδών (subgenres) των καλλιτεχνών.

Συγκεκριμένα, εξάγει όλα τα Review του επισκέπτη, επιστρέφοντας το review\_id, την συνολική βαθμολογία (score) και το event\_id. Στην συνέχεια, με χρήση ROW\_NUMBER() κατατάσσει τις ανασκοπήσεις κατά φθίνουσα σειρά βαθμολογίας και κρατά μόνο τις τρεις (3) κορυφαίες (rn ≤ 3).Εντοπίζει τους καλλιτέχνες (Performer\_ID) που συμμετείχαν σε αυτά τα τρία κορυφαία γεγονότα και, εφόσον υπάρχουν solo καλλιτέχνες στα topevents, βρίσκει τις μπάντες (Band\_ID) στις οποίες ανήκουν—επεκτείνοντας έτσι το seedset.

Η ένωση (UNION) αυτών των καλλιτεχνών και των μπαντών αποθηκεύεται σε ένα νέο ενιαίο σύνολο, το seed\_performers. Αντίστοιχα, γίνεται και για το seed\_subgenres. Εξάγονται όλα τα subgenre\_ID που σχετίζονται με αυτούς τους καλλιτέχνες, δηλαδή τα μουσικά υπο-είδη που προτιμά ουσιαστικά ο επισκέπτης. Το all\_similarσύνολο θα περιέχει όλους τους καλλιτέχνες που μοιράζονται αυτά τα υπο-είδη.

Ο Υπολογισμός Βαθμολογίας Γεγονότων γίνεται μέσω του event\_scores ως εξής:

Για κάθε γεγονός πουπεριλαμβάνει κάποιον από τους καλλιτέχνες του all\_similar και δεν έχει ήδη παρακολουθήσει ο επισκέπτης (εξαιρεί τα event\_id του visitor\_reviews),υπολογίζει δύο μετρικές:

* perf\_matches: πλήθος μοναδικών καλλιτεχνών/μπάντωνseed που εμφανίζονται στο γεγονός
* subgenre\_matches: πλήθος μοναδικών υπο-ειδών seed στο σύνολο όλων των καλλιτεχνών του γεγονότος

Τέλος για την τελική επιλογή και ταξινόμηση:

το κύριο SELECT προσθέτει μία στήλη total\_score = perf\_matches + subgenre\_matches και ταξινομεί με βάση το total\_score.