

2025

Αναφορά Εξαμηνιαίας Εργασίας Βάσεων Δεδομένων

ΜΑΤΘΑΙΟΣ ΑΓΓΕΛΗΣ (el22027)

ΝΙΚΟΛΑΟΣ ΑΘΑΝΑΣΟΠΟΥΛΟΣ (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. Διάγραμμα Οντοτήτων-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 (Activated = TRUE) αντιστοιχούν μόνο σε events που έχουν ήδη πραγματοποιηθεί, δηλαδή σε παρελθοντικά φεστιβάλ, καθώς η ενεργοποίηση γίνεται κατά την είσοδο στον χώρο.
* **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.
* **Αξιολογήσεις με Likert Scale**: Οι κριτικές (Review) ακολουθούν την κλασική κλίμακα Likert από 1 έως 5, όπου το 1 αντιστοιχεί στη χειρότερη και το 5 στην καλύτερη βαθμολογία.
* **EAN Code 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:

1. def calculate\_ean13\_check\_digit(base12: str) -> str:

2.     total = 0

3.     for idx, digit\_char in enumerate(reversed(base12), start=1):

4.         digit = int(digit\_char)

5.         total += digit \* (3 if idx % 2 == 0 else 1)

6.     return str((10 - (total % 10)) % 10)

7.

8. def generate\_ean13() -> str:

9.     base12 = ''.join(random.choices(string.digits, k=12))

10.     return base12 + calculate\_ean13\_check\_digit(base12)

11.

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

Αρχικά φτιάχτηκε το διάγραμμα οντοτήτων-συσχετίσεων στο 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 foreign keys. 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 primary keys, 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 foreign keys. Με αυτόν τον τρόπο, μετατρέπουμε τον χρόνο αναζήτησης από γραμμικό (O(n)) σε λογαριθμικό (O(log n)).

Παρότι προσθέσαμε αρκετά ευρετήρια, αξίζει να σημειωθεί ότι η MySQL διαθέτει τον δικό της query optimizer, 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 NOT NULL 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.

Ο πίνακας Festival περιέχει βασικές πληροφορίες για κάθε φεστιβάλ όπως Name, Start\_Date, End\_Date και Location\_ID. Το Location\_ID είναι foreign key που παραπέμπει στον πίνακα Location. Επιπλέον, με χρήση της εντολής GENERATED ALWAYS AS δημιουργούμε το Year του φεστιβάλ από την Start\_Date. Το Year είναι UNIQUE ώστε να υπάρχει ένα φεστιβάλ ανά έτος.

Για κάθε εκδήλωση (Event) ορίζουμε Start\_Time, End\_Time, και αν έχει εξαντληθεί (Sold\_Out). Η Stage\_ID και η Festival\_ID είναι foreign keys προς τους αντίστοιχους πίνακες. Επίσης, θέτουμε UNIQUE(Festival\_ID, Stage\_ID, Start\_Time) ώστε να μην υπάρχουν επικαλύψεις.

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. 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.

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 foreign key προς το Staff.

Το σχήμα περιλαμβάνει ακόμα πίνακες για Ticket, Spectator, Review, Transaction, Membership μεταξύ μελών και συγκροτημάτων, Genre, Subgenre, και αντιστοίχιση καλλιτεχνών με Performer\_Subgenre. Οι πίνακες Image και οι αντίστοιχοι συσχετιστικοί πίνακες (Performer\_Image, Stage\_Image, κ.λπ.) διαχειρίζονται φωτογραφίες σχετικών οντοτήτων.

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 INT NOT NULL ώστε να υποστηρίζονται σωστά οι περιορισμοί πληρότητας μέσω triggers.

Ακολουθεί ο πίνακας Type, που περιέχει τις διαθέσιμες κατηγορίες εισιτηρίων, όπως VIP ή General. Χρησιμοποιείται σε πολλούς πίνακες της βάσης ως foreign key, 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 foreign key σε πολλαπλές συσχετίσεις.

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.

Η ουρά μεταπώλησης εισιτηρίων υλοποιείται μέσω του πίνακα Resale\_Queue, ο οποίος ενεργοποιείται αυτόματα από triggers όταν μια εκδήλωση χαρακτηρίζεται ως Sold\_Out. 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).

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.

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.

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).

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.

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.

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.

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.

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 foreign keys και primary keys.

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.

### 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 FOREIGN KEY.

**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 trigger check\_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 trigger prevent\_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\_performances ελέγχει 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 trigger prevent\_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 trigger trg\_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 trigger trg\_before\_insert\_vie καλεί την procedure handle\_visitor\_interest() ώστε, εάν το event είναι Sold\_Out, να ενταχθεί ο επισκέπτης στη λίστα αναμονής.

**Μεταπώληση εισιτηρίων**

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

**Εισαγωγή στη Buy\_Queue**

Το trigger trg\_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 λογική της 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 trigger after\_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% των θεατών.
* Υποστήριξη: τουλάχιστον 2% των θεατών.

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:

* Θέτει το Sold\_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 με Is\_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 DML scripts**. Τα δεδομένα αυτά προστέθηκαν με χρήση εντολών INSERT INTO, 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 query καθώς και η ανάλυση 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.

##### 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 Ticket t 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.

###### 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 JOIN είναι αποδοτικότερο να ξεκινήσει πρώτο. Ενδεικτικά παραθέτουμε κομμάτι του κώδικα της υλοποίησης αυτής:

1. SELECT

2. pe.ID AS Performer\_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.

**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 FORCE INDEX 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). Σε διαφορετική περίπτωση η απόδοση ήταν κατώτερη.

**Συμπεράσματα**

* Η χρήση FORCE INDEX 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 Nested Loop Join επιλέγεται από προεπιλογή, και μόνο με χειροκίνητη ενεργοποίηση optimizer\_switch='hash\_join=on' χρησιμοποιήθηκε Hash Join, 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. 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.

##### 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. Spectator sp FORCE INDEX (idx\_visitor\_ticket)

2. Ticket t 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 foreign key μεταξύ Ticket\_ID, Event\_ID και Visitor\_ID, τα οποία συμμετέχουν ενεργά στα JOIN και στο WHERE.

###### 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 WHERE sp.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.

**Σύγκριση Στρατηγικών Join**

* Η χρήση FORCE INDEX 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 Nested Loop Join χρησιμοποιήθηκε εξ ορισμού και από τα δύο query plans, με τον query optimizer να μην ενεργοποιεί Hash Join εκτός αν επιβληθεί ρητά μέσω optimizer\_switch.
* Η χρήση FORCE INDEX 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 composite index 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.