Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων

Ακ.έτος: 2024-2025, 6ο εξάμηνο, ΣΗΜΜΥ

Βάσεις Δεδομένων
Εξαμηνιαία Εργασία

Ομάδα 19
Πηνελόπη Κόπανου, ΑΜ: 03121871
Αικατερίνη Δανάη Τάτση ΑΜ: 03121146

GitHub Repository: https://github.com/kattatsi/db_festival

Εισαγωγή

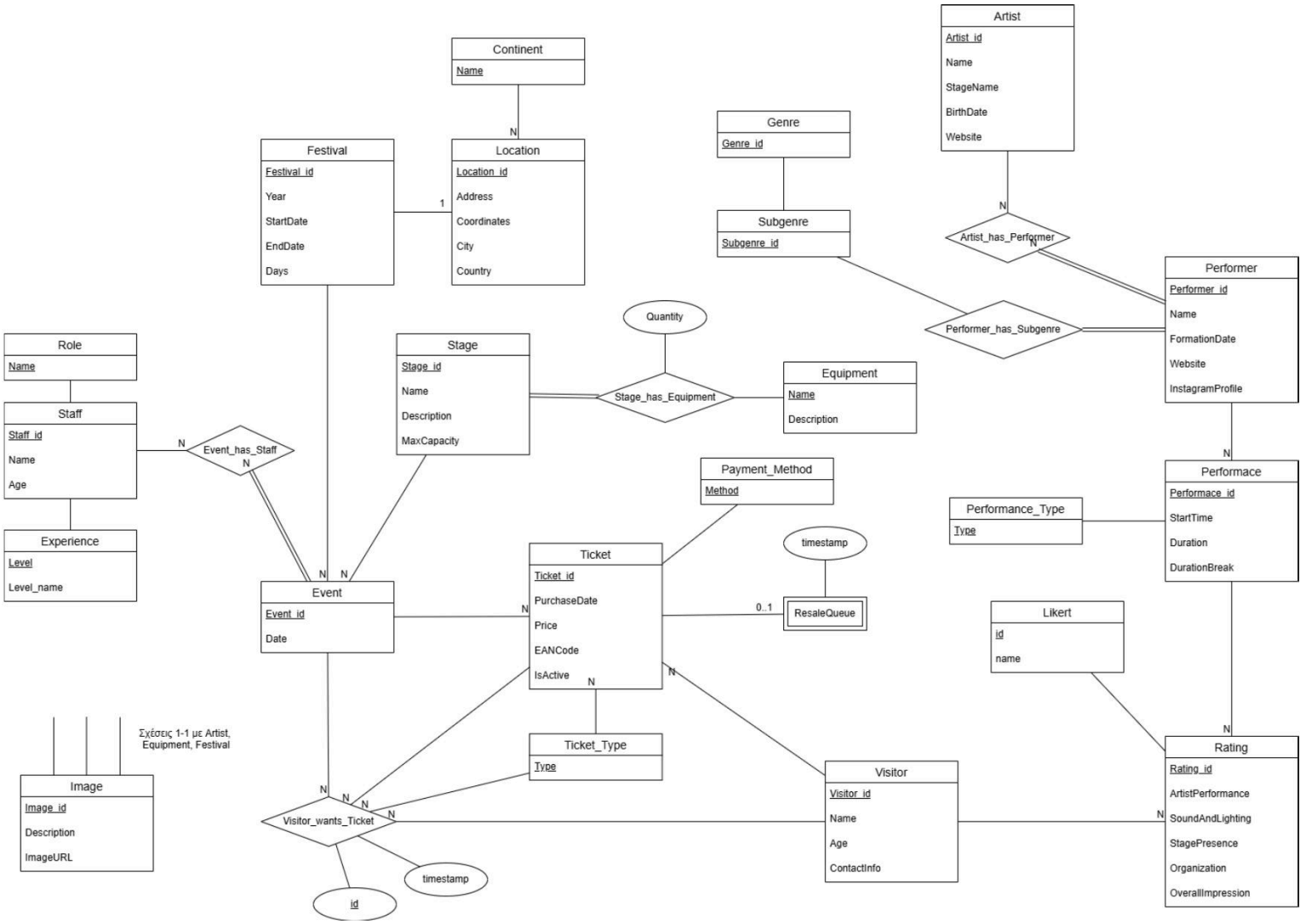
Η πορεία που ακολουθήσαμε ώστε να φέρουμε εις πέρας την παρούσα εργασία ξεκίνησε από την δημιουργία του ER διαγράμματος της βάσης σε χαρτί, ως προσχέδιο. Ακολούθησε η δημιουργία μιας πρώτης έκδοσης του σχεσιακού διαγράμματος στο MySQL Workbench το οποίο κάναμε export και φτιάξαμε την πρώτη έκδοση του DDL script μας. Έπειτα προσθέσαμε triggers και constraints για όλους τους περιορισμούς που αναφέρονται στην εκφώνηση και έτσι φτιάξαμε την πρώτη έκδοση της βάσης μας.

Στη συνέχεια, συλλέξαμε ένα δείγμα από τα τελικά δεδομένα ώστε να μπορέσουμε να αρχίσουμε να δουλεύουμε στα επόμενα κομμάτια της εργασίας. Μετά την συλλογή των δεδομένων τα επεξεργαστήκαμε ώστε να έρθουν στη μορφή της βάσης μας και δημιουργήσαμε scripts για να γεμίσουν με τυχαίο τρόπο,τηρώντας τους περιορισμούς, οι υπόλοιποι πίνακες. Κατά τη διάρκεια αυτή χρειάστηκε αρκετές φορές να γίνουν αλλαγές στο DDL script.

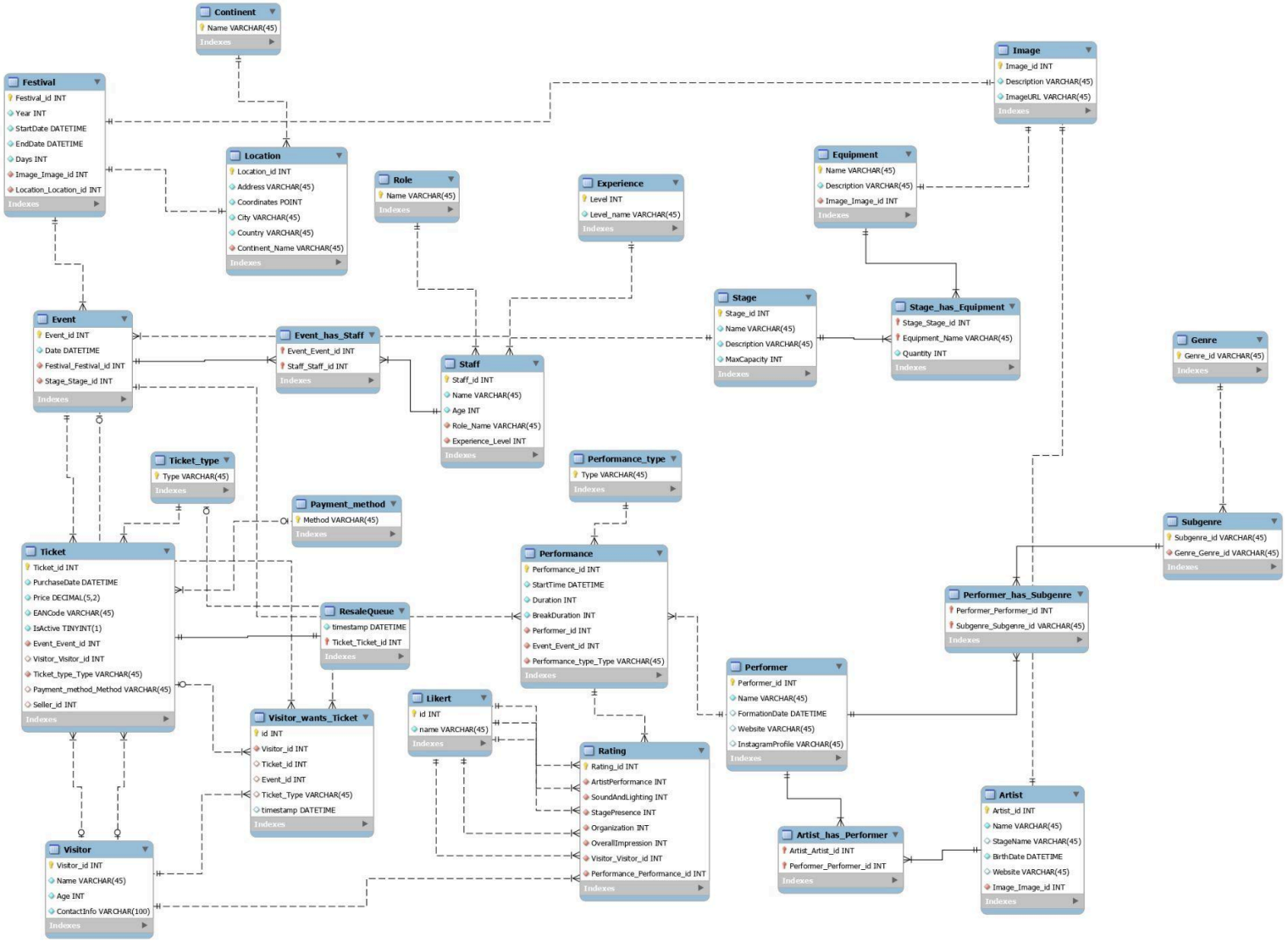
Αφού είχαμε ένα δείγμα δεδομένων έτοιμο, εργαστήκαμε πάνω στα queries που μας ζητούνται και δοκιμάσαμε τα triggers και τα constraints που έχουμε υλοποιήσει για να βεβαιωθούμε ότι δουλεύουν όπως πρέπει. Με βάση τα queries υλοποιήσαμε και συγκεκριμένα indexes για βελτιστοποίηση. Τέλος συλλέξαμε, επεξεργαστήκαμε και φορτώσαμε παραπάνω δεδομένα στη βάση ώστε να πληρούμε τους ελάχιστους περιορισμούς.

Η βάση μας δημιουργήθηκε σε έναν MySQL server version 9.0.1 και το link για το GitHub repository μας είναι: https://github.com/kattatsi/db_festival. Στο αρχείο README.md, το οποίο υπάρχει στο repository, αναγράφονται οδηγίες για τη δημιουργία και φόρτωση της βάσης καθώς και τις παραδοχές μας για την υλοποίηση της εργασίας.

ER Διάγραμμα



Σχεσιακό Διάγραμμα



Tables:

Image: Περιλαμβάνει id ως PK, description για περιγραφή εικόνας και image_url για το url της εικόνας.

Continent: Περιλαμβάνει name για το όνομα της ηπείρου ως PK.

Location: Περιλαμβάνει location_id ως PK, Address, Coordinates, City, Country, για τον ακριβή προσδιορισμό της τοποθεσίας και Continent_name ως FK στο Continent για την ήπειρο στην οποία βρίσκεται.

Festival: Περιλαμβάνει Festival_id ως PK, Year καθώς το φεστιβάλ είναι ετήσιο, StartDate, EndDate για τον προσδιορισμό των ημερομηνιών του φεστιβάλ, Days για τις συνολικές μέρες, Image_id και Location_id ως FK στους αντίστοιχους πίνακες για τον προσδιορισμό του poster για το φεστιβάλ και της τοποθεσίας που είναι διαφορετική για κάθε φεστιβάλ.

Stage: Περιλαμβάνει Stage_id ως PK, name μοναδικό όνομα για κάθε σκηνή, Description για την περιγραφή του vibe της σκηνής και MaxCapacity όπου είναι η μέγιστη χωρητικότητα της σκηνής.

Event: Περιλαμβάνει Event_id ως PK, Day καθώς κάθε φεστιβάλ πραγματοποιείται μία μέρα, Festival_id και Stage_id ως FK στους αντίστοιχους πίνακες για το φεστιβάλ στο οποίο ανήκει και το stage στο οποίο γίνεται.

Equipment: Περιλαμβάνει name ως PK που αποτελεί το όνομα του εξοπλισμού, Description περιγραφή για το συγκεκριμένο εξάρτημα και Image_id ως FK στο Image που αντιστοιχεί στην εικόνα του εξαρτήματος.

Role: Περιλαμβάνει μόνο name ως PK και αποτελεί τον ρόλο του προσωπικού. Υποθέσαμε το εξής:
Οι ρόλοι ταξινομούνται σε τρεις βασικές κατηγορίες: **τεχνικό προσωπικό** *('Lighting Technician', 'Sound Engineer', 'Stage Manager')*, **προσωπικό ασφαλείας** *('Security', 'First Aid Responder')* και **βοηθητικό προσωπικό** *('Artist Liaison', 'Cleanup Crew', 'Ticket Scanner', 'Vendor Coordinator' και 'Parking Attendant').*

Experience: Περιλαμβάνει Level ως PK και είναι το επίπεδο εμπειρίας του προσωπικού (1-5) και Level_name που αντιστοιχεί στην λεκτική περιγραφή του experience level.

Staff: Περιλαμβάνει staff_id ως PK, name, age για το όνομα και την ηλικία του, Role_name και Experience_level ως FK στους αντίστοιχους πίνακες για τον ρόλο και τον βαθμό εμπειρίας του κάθε εργαζόμενου προσωπικού.

Performer: Περιλαμβάνει performer_id ως PK για κάθε σόλο καλλιτέχνη και συγκρότημα, Name είτε το όνομα του καλλιτέχνη είτε του συγκροτήματος, FormationDate ως ημερομηνία debut της καριέρας του ή σχηματισμού του συγκροτήματος, website και Instagram profile που περιλαμβάνουν links για την αντίστοιχη ιστοσελίδα του performer και του instagram account του.

Performance_type: Περιλαμβάνει μόνο type ως PK και περιγράφει τον τύπο της εμφάνισης.

Performance: Περιλαμβάνει Performance_id ως PK, StartTime για την ώρα που ξεκινάει, Duration η διάρκεια της εμφάνισης, BreakDuration διάρκεια διαλείμματος μετά το performance, Performer_id, Event_id, Performance_type ως FK στους αντίστοιχους πίνακες καθώς κάθε performace έχει ένα performer, είναι μέρος ενός event και έχει συγκεκριμένο τύπο.

Artist: Περιλαμβάνει Artist_id ως PK, Name για ολόκληρο το ονοματεπώνυμο του καλλιτέχνη, StageName το καλλιτεχνικό ψευδώνυμο των καλλιτεχνών, BirthDate ημερομηνία γέννησης, Website την ιστοσελίδα του και Image_id ως FK καθώς υπάρχει φωτογραφία για τον κάθε καλλιτέχνη.

Visitor: Περιλαμβάνει visitor_id ως PK, name όνοματεπώνυμο , age ηλικία και contact info (με τηλέφωνο και email) για τον κάθε επισκέπτη του φεστιβάλ.

Artist_has_Performer: Περιλαμβάνει Artist_id και Performer_id ως PK, ώστε να γίνεται η αντιστοίχιση μεταξύ των performers και των artist καθώς και των μελών ενός συγκροτήματος με αυτό.

Ticket_type: Περιλαμβάνει Type ως PK, δηλαδή τον τύπο που θα μπορεί να έχει ένα εισιτήριο.

Payment_method: Περιλαμβάνει method ως PK για τους πιθανούς τρόπους πληρωμής των εισητηρίων.

Ticket: Περιλαμβάνει Ticket_id ως PK, PurchaseDate για την ημερομηνία αγοράς του εισητηρίου, Price για την τιμή του καθώς διαφέρει ανά event και τύπο εισιτηρίου, EANCode έναν μοναδικό EAN-13 κωδικό, IsActive ώστε να γνωρίζουμε αν είναι ενεργοποιημένο (scanned=> isActive=1) ή όχι (isActive=0) το εισιτήριο, Event_id, Visitor_id, Ticket_type και Payment_method ως FK στους αντίστοιχους πίνακες καθώς κάθε εισητήριο είναι για ένα event, ανήκει σε έναν επισκέπτη, έχει συγκεκριμένο τύπο και τρόπο πληρωμής και διαθέτει seller_id που είναι FK στον πίνακα Visitor σε περίπτωση που κάποιος θέλει να μεταπωλήσει το εισιτήριο του αφού γίνουν sold out για το συγκεκριμένο event και τότε το visitor id του υπάρχει και στο seller id.

ResaleQueue: Περιλαμβάνει Ticket_id ως PK και timestamp. Πρόκειται για την ουρά μεταπώλησης των εισιτηρίων όπου καταχωρούνται τα εισιτήρια όσων θέλουν να τα πουλήσουν και έχει timestamp ώστε να τηρείται ουρά FIFO.

Likert: Περιλαμβάνει Id ως PK και Name για την λεκτική περιγραφή της τιμής του Id και πρόκειται για την κλίμακα Likert με 5 επίπεδα.

Rating: Περιλαμβάνει Rating_id ως PK, ArtistPerformance, SoundAndLighting, StagePresenc, Organization και OverallImpression που αποτελούν τα 5 χαρακτηριστικά μίας εμφάνισης που βαθμολογεί ο επισκέπτης στην κλίμακα Likert(FK στον πίνακα Likert), Visitor_id και Performance_id ως FKστους αντίστοιχους πίνακες καθώς κάθε αξιολόγηση κάποιου χρήστη είναι για ένα συγκεκριμένο performance.

Event_has_Staff: Περιλαμβάνει Event_id και Staff_id ως PK (ως FK στους αντίστοιχους πίνακες). Κάθε event έχει συγκεκριμένο προσωπικό που δουλεύει σε αυτό και με αυτόν τον τρόπο δηλώνεται ποιοί δουλεύουν σε κάθε event.

Genre: Περιλαμβάνει το genre_id ως PK που περικλείει όλα τα μουσικά είδη τα οποία θα μπορούσε να παίζει ένας καλλιτέχνης.

Subgenre: Περιλαμβάνει το subgenre_id ως PK και το genre_id ως FK στον πίνακα genre, καθώς κάθε υποείδος ανήκει σε ένα είδος μουσικής.

Performer_has_Subgenre: Περιλαμβάνει το Performer_id και το subgenre_id ως PK (FK στους αντίστοιχους πίνακες performer και subgenre) δείχνοντας ποιά μουσικά υποείδη παίζει κάθε καλλιτέχνης.

Stage_has_Equipment: Περιλαμβάνει Stage_id και Equipment_name ως PK (FKs στους αντίστοιχους πίνακες) και quantity, δηλαδή την ποσότητα κάθε εξοπλισμού που χρειάζεται κάθε event.

Visitor_wants_Ticket: Περιλαμβάνει id ως P, visitor_id το id του επισκέπτη που ενδιαφέρεται για το συγκεκριμένο event, ticket_id FK στο ticket, event_id FK στον πίνακα event, Ticket_type FK στον πίνακα ticket_type και timestamp η ημερομηνία ενδιαφέροντας για το εισιτήριο καθώς θέλουμε να τηρούμε σειρά FIFO.

DDL script

Στο DDL script (sql/install.sql στο repository μας) πέρα από την υλοποίηση των tables που φαίνονται και στο σχεσιακό μας διάγραμμα παραπάνω, χρειάστηκε να δημιουργήσουμε κάποια triggers και constraints για τους περιορισμούς που αναφέρονται στην εκφώνηση καθώς και μερικά indexes για την καλύτερη απόδοση των queries μας. Τα triggers και τα constraints που γράψαμε για τους περιορισμούς είναι τα εξής:

Triggers:

check_stage_capacity: Για να διασφαλίσουμε ότι ο αριθμός των εισιτηρίων για ένα event δεν θα ξεπεράσει την μέγιστη χωρητικότητα (maxCapacity) της σκηνής δημιουργήσαμε αυτό το before insert trigger στον πίνακα ticket. Πριν την εκτέλεση ενός insert ενός εισιτηρίου λοιπόν ελέγχεται η μέγιστη χωρητικότητα της σκηνής του event για το οποίο προορίζεται το εισιτήριο και συγκρίνεται με το πλήθος των εισιτηρίων που έχουμε για αυτό το event. Εάν έχουν εκδοθεί περισσότερα ή ίσα εισιτήρια με την χωρητικότητα της σκηνής, απορρίπτεται η εισαγωγή και βγαίνει κατάλληλο μήνυμα σφάλματος.

trg_Check_Event_Duration: Αυτό το trigger διασφαλίζει ότι κάθε event τηρεί τη μέγιστη διάρκεια που θα μπορούσε να έχει, που διευκρινίζεται στην εκφώνηση. Πιο συγκεκριμένα πρόκειται για ένα before insert trigger στον πίνακα Performance, όπου αθροίζεται η διάρκεια όλων των performances του συγκεκριμένου Event και σε περίπτωση που προσθέσουμε τη διάρκεια του νέου Performance και υπερβούμε τα 720 λεπτά απορρίπτεται η εισαγωγή και βγαίνει κατάλληλο μήνυμα σφάλματος.

trg_Check_Festival_Day_Duration: Αυτό το before insert trigger στον πίνακα Performance διασφαλίζει ότι η συνολική διάρκεια όλων των performances που πραγματοποιούνται την ίδια ημερομηνία ενός festival δεν ξεπερνάει τις 13 ώρες (δηλαδή 780 λεπτά). Για κάθε νέα εμφάνιση, εντοπίζεται η ημερομηνία του event της και υπολογίζεται η αθροιστική διάρκεια όλων των ήδη καταχωρημένων εμφανίσεων την ίδια μέρα. Αν η προσθήκη της νέας εμφάνισης υπερβεί το όριο των 780 λεπτών, η εισαγωγή απορρίπτεται με κατάλληλο μήνυμα σφάλματος.

trg_Check_Performance_Break: Αυτό το before insert trigger στον πίνακα Performance ελέγχει αν τηρείται ο περιορισμός για το διάλειμμα μεταξύ δύο performances. Τσεκάρει αν στο insert μας το BreakDureation είναι από 5 μέχρι 30 λεπτά και αν ισχύει προχωράει κανονικά το insert. Διαφορετικά απορρίπτεται η εισαγωγή και βγαίνει κατάλληλο μήνυμα σφάλματος.

trg_Check_Performer_Overlap: Αυτό το before insert trigger στον πίνακα Performance διασφαλίζει ότι ένας performer (καλλιτέχνης ή συγκρότημα) δεν εμφανίζεται ταυτόχρονα σε δύο σκηνές. Συγκεκριμένα, ελέγχει αν το χρονικό διάστημα της νέας εμφάνισης επικαλύπτεται με οποιαδήποτε άλλη εμφάνιση του ίδιου performer. Αν εντοπιστεί επικάλυψη, η εισαγωγή απορρίπτεται με κατάλληλο μήνυμα σφάλματος.

trg_Limit_Performer_3_Consecutive_Years: Το trigger αυτό διασφαλίζει ότι κανένας καλλιτέχνης δεν μπορεί να συμμετάσχει στο φεστιβάλ για περισσότερα από 3 συνεχόμενα χρόνια. Είναι ένα before insert trigger στον πίνακα Performance όπου βρίσκει το έτος συμμετοχής του performer που γίνεται insert και ελέγχει αν υπάρχουν performances του και τα 3 προηγούμενα χρόνια. Αν

ναι, τότε απορρίπτει την εισαγωγή του και βγάζει κατάλληλο μήνυμα σφάλματος.

prevent_double_activation: Αυτό είναι ένα before update trigger στον πίνακα Ticket. Με αυτόν τον τρόπο αποφεύγεται η ενεργοποίηση ενός εισιτηρίου 2 φορές. Κοιτάει τον πίνακα isActive για να δει αν είναι ήδη ενεργοποιημένο και σε αυτή την περίπτωση απορρίπτεται η εισαγωγή και βγαίνει αντίστοιχο μήνυμα λάθους.

trg_Visitor_One_Ticket_Per_Event: Αυτό το before insert trigger στον πίνακα Visitor_wants_Ticket διασφαλίζει ότι ένας επισκέπτης δεν μπορεί να εκδηλώσει ενδιαφέρον για εισιτήριο συγκεκριμένου event, αν ήδη έχει αγοράσει εισιτήριο για το ίδιο event. Ελέγχεται αν υπάρχει ήδη καταχώρηση στο Ticket με το ίδιο Visitor_id και Event_id, και αν ναι, η εισαγωγή απορρίπτεται με μήνυμα σφάλματος.

trg_Rating_OnlyWithActiveTicket: Πρόκειται για ένα before insert trigger στον πίνακα Rating που διασφαλίζει ότι δεν μπορεί να αξιολογήσει κάποιο performance ένας Visitor με μη ενεργό εισιτήριο. Για τον συγκεκριμένο visitor ελέγχουμε αν έχει ενεργοποιημένο εισιτήριο και μόνο τότε γίνεται κανονικά το insert. Διαφορετικά ακυρώνεται και βγαινει μήνυμα λάθους.

prevent_resale_of_active_ticket: Αυτό το before insert trigger στον πίνακα ResaleQueue ελέγχει αν είναι ενεργοποιημένο το εισιτήριο το οποίο προσπαθεί να μπει στην ουρά μεταπώλησης. Προφανώς ένα ενεργοποιημένο εισιτήριο δεν πρέπει να μπορεί να μεταπωληθεί οπότε αυτό το trigger ελέγχει αν το συγκεκριμένο εισιτήριο έχει ενεργοποιηθεί και τότε ακυρώνει το insert και βγάζει μήνυμα λάθους.

trg_Limit_Technician_Performances: Με αυτό το trigger διασφαλίζεται ότι κάθε τεχνικός από το προσωπικό μπορεί να δουλέψει το πολύ σε 2 performances την ίδια μέρα. Είναι ένα before insert trigger στον πίνακα Event_has_Staff όπου όταν πάει να γίνει insert αν η ειδικότητα του είναι τεχνικός ελέγχει την ημέρα του event που θα του ανατεθεί και μετά υπολογίζει πόσα performances έχει εκείνη την ημέρα. Αν έχει ήδη >= performances τότε ακυρώνεται η εισαγωγή και βγαίνει μήνυμα λάθους.

trg_OnlyOneEventAtTime_PerStage: Αυτό το before insert trigger στον πίνακα Performance ελέγχει ότι κάθε σκηνή μπορεί να φιλοξενεί μόνο μία παράσταση την ίδια στιγμή. Βρίσκει τη μέρα στην οποία θα γίνει η νέα εμφάνιση και ελέγχει αν υπάρχουν άλλες εμφανίσεις που επικαλύπτονται με την καινούργια. Αν υπάρχει επικάλυψη ακυρώνεται το insert και βγαίνει μήνυμα λάθους.

sequential_performances_per_event: Με αυτό το trigger ελέγχεται ότι οι εμφανίσεις για ένα event είναι σειριακές και δεν έχουν κενά, πέρα του διαλείμματος, μεταξύ τους. Βρίσκει την ώρα λήξης της τελευταίας εμφάνισης (προσθέτοντας στην ώρα έναρξης της την διάρκεια της και το διάλειμμα της) του οποίου θέλουμε να προσθέσουμε νέα. Εάν η ώρα έναρξης της νέας είναι ακριβώς εκείνη την ώρα που υπολογίσαμε πριν τότε γίνεται κανονικά το insert καθώς ικανοποιείται ο περιορισμός. Διαφορετικά ακυρώνεται το insert και βγαίνει μήνυμα λάθους.

fifo_only_when_sold_out: Αυτό το before insert trigger στον πίνακα Visitor_wants_Ticket διασφαλίζει ότι ένας επισκέπτης μπορεί να ενταχθεί στην ουρά μεταπώλησης (FIFO) μόνο όταν όλα τα εισιτήρια ενός event έχουν πουληθεί, δηλαδή όταν κάθε εισιτήριο έχει αντιστοιχιστεί σε κάποιον επισκέπτη (Visitor_Visitor_id IS NOT NULL). Για το συγκεκριμένο Event_id, μετριούνται τα συνολικά εισιτήρια που έχουν εκδοθεί και συγκρίνονται με εκείνα που έχουν αγοραστεί. Αν βρεθεί έστω και ένα εισιτήριο που δεν έχει αποδοθεί σε επισκέπτη, τότε απορρίπτεται η καταχώρηση στην ουρά και εμφανίζεται κατάλληλο μήνυμα σφάλματος.

Η **μεταπώληση των εισιτηρίων** με FIFO υλοποιείται κυρίως με τα παρακάτω 2 triggers:

before_resale_queue_insert: Το συγκεκριμένο trigger είναι before insert στον πίνακα ReasaleQueue. Το trigger εξασφαλίζει την σωστή λειτουργία δύο περιπτώσεων. Στην πρώτη περίπτωση αν υπάρχει κάποιος στον πίνακα Visitor_wants_Ticket που θέλει εισιτήριο αυτού του τύπου για αυτό το event βρίσκει τον πρώτο που το καταχώρησε με βάση την ημερομηνία (για να τηρείται FIFO) και του το “πουλάει” αυτόματα. Δηλαδή αλλάζει το visitor_id στον Ticket στο id του αγοραστή και μπαίνει null το seller_id και διαγράφεται η καταγραφή ενδιαφέροντος του χρήστη αυτού για αυτό το event και ticket_type

από τον πίνακα visitor_wants_ticket. Το insert στον πίνακα ResaleQueue δεν πραγματοποιείται καθώς η αγορά έγινε αυτόματα και βγαίνει μήνυμα που ενημερώνει ότι το εισιτήριο αγοράστηκε επιτυχώς. Στην δεύτερη περίπτωση όπου δεν υπάρχει κανένας επισκέπτης που να έχει δηλώσει ενδιαφέρον για αυτό το event και ticket_type ενημερώνεται το seller_id στο εισιτήριο (μπαίνει το visitor_id του εισιτηρίου που μπαίνει για μεταπώληση) και γίνεται το insert στον πίνακα ResaleQueue κανονικά βάζοντας το τωρινό timestamp για να τηρηθεί FIFO αργότερα όταν χρειαστεί.

before_visitor_wants_ticket_insert: Το trigger αυτό είναι before insert στον πίνακα Visitor_wants_Ticket. Καλύπτει τις 2 δυνατές περιπτώσεις όταν γίνεται εισαγωγή σε αυτόν τον πίνακα. Στην πρώτη περίπτωση ο επισκέπτης ενδιαφέρεται για ένα συγκεκριμένο εισιτήριο που υπάρχει διαθέσιμο στο ResaleQueue. Τότε αν το εισιτήριο είναι υπάρχει όντως στην ουρά πωλείται σε αυτόν τον επισκέπτη καθώς είναι ο πρώτος που το ζήτησε. Πιο συγκεκριμένα αλλάζουμε το visitor_id σε εκείνο του αγοραστή και θέτουμε seller_id null, διαγράφεται το εισιτήριο από το ResaleQueue και ακυρώνεται το insert αφού αγόρασε εισιτήριο ενημερώνοντας με ένα μήνυμα ότι έγινε η αγορά αυτόματα. Στην δεύτερη περίπτωση ο επισκέπτης ζητάει εισιτήριο για ένα συγκεκριμένο event και ticket_type. Ο διαχωρισμός τον περιπτώσεων γίνεται με βάση το αν το ticket id στο insert είναι null ή όχι (αν είναι τότε πρόκειται για την δεύτερη περίπτωση αλλιώς για την πρώτη). Σε αυτή την περίπτωση λοιπόν ψάχνουμε στο ResaleQueue να βρούμε το πιο παλιο εισιτήριο (με βάση timestamp) για αυτο το event και ticket_type (αν υπάρχει). Εφόσον υπάρχει πωλείται στον αγοραστή και αντίστοιχα με την προηγούμενη περίπτωση ακολουθείται η ίδια διαδικασία. Αν δεν υπάρχει τότε γίνεται κανονικά το insert στον πίνακα προσθέτοντας το τωρινό timestamp.

Constraints:

chk_Performance_MaxDuration: Ο περιορισμός αυτός εφαρμόζεται στον πίνακα Performance και διασφαλίζει ότι η διάρκεια κάθε εμφάνισης δεν υπερβαίνει τα 180 λεπτά (δηλαδή τις 3 ώρες).

resale_fifo: Ο περιορισμός αυτός εξασφαλίζει ότι το πεδίο timestamp του πίνακα ResaleQueue παίρνει αυτόματα την τρέχουσα ημερομηνία και ώρα κατά

την εισαγωγή, ώστε να διατηρείται σωστά η χρονική σειρά στην ουρά μεταπώλησης (FIFO).

unique_eancode: Με τον συγκεκριμένο περιορισμό διασφαλίζουμε την μοναδικότητα του EAN13 code για κάθε εισιτήριο.

unique_visitor_performance: Με αυτόν τον περιορισμός διασφαλίζουμε ότι μπορεί να υπάρχει μόνο ένα ζευγάρι visitor-performance στον πίνακα rating. Κάθε visitor μπορεί να αξιολογήσει μόνο μία φορά ένα performance δηλαδή.

unique_visitor_event: Το constraint στον πίνακα Ticket διασφαλίζει ότι ένας επισκέπτης μπορεί να έχει ένα μόνο εισιτήριο για ένα συγκεκριμένο event. Δηλαδή το ζευγάρι visitor_id, event_id είναι μοναδικό στον πίνακα Ticket.

chk_equipment_quantity: Αυτός ο περιορισμός εξασφαλίζει ότι στον πίνακα Stage_has_Equipment δεν μπορούμε να έχουμε εξοπλισμό με ποσότητα μικρότερη του 0.

chk_consecutive_days: Το φεστιβάλ πρέπει να γίνεται μόνο συνεχόμενες μέρες. Οπότε δεν θα πρέπει να υπάρχουν κενά μεταξύ των ημερομηνιών που διεξάγεται κάθε φεστιβάλ.

unique_festival_location: Το constraint αυτο εξασφαλίζει ότι κάθε φεστιβάλ πραγματοποιείται σε διαφορετική τοποθεσία ανά έτος.

unique_festival_year: Διασφαλίζεται ότι κάθε φεστιβάλ έχει μοναδική χρονιά, καθώς το φεστιβάλ είναι ετήσιο.
(ισως δεν χρειαζεται εδω γιατι εχουμε ορισει το year ως unique στο er)

Procedures:

CheckSecurityCoverageByStage(): Η procedure αυτή ελέγχει αν το προσωπικό ασφαλείας (δηλαδή όσοι έχουν ρόλο *'Security' ή 'First Aid Responder'*) καλύπτει τουλάχιστον το 5% του ενεργού κοινού για κάθε σκηνή και ημερομηνία. Το ενεργό κοινό υπολογίζεται ως ο αριθμός των επισκεπτών με ενεργοποιημένα εισιτήρια (IsActive = 1) για τα events της εκάστοτε σκηνής. Αν

ο αριθμός των τεχνικών ασφαλείας είναι μικρότερος από το απαιτούμενο 5%, η παράβαση καταγράφεται στον προσωρινό πίνακα SecurityViolations.

CheckSupportCoverageByStage(): Η procedure αυτή ελέγχει αν το βοηθητικό προσωπικό (δηλαδή όσοι έχουν ρόλο *'Artist Liaison', 'Cleanup Crew', 'Ticket Scanner', 'Vendor Coordinator', 'Parking Attendant'*) καλύπτει τουλάχιστον το 2% του ενεργού κοινού για κάθε σκηνή και ημερομηνία. Το κοινό υπολογίζεται με βάση τους επισκέπτες που έχουν ενεργά εισιτήρια (IsActive = 1) για τα events της σκηνής. Αν το προσωπικό υποστήριξης είναι λιγότερο από το απαιτούμενο όριο, η σκηνή καταγράφεται στον προσωρινό πίνακα SupportViolations.

CheckVipLimitPerEvent(): Η procedure αυτή ελέγχει αν ο αριθμός των VIP εισιτηρίων που έχουν εκδοθεί για κάθε event υπερβαίνει το 10% της χωρητικότητας της σκηνής στην οποία διεξάγεται το event. Στην περίπτωση που εντοπίζεται υπέρβαση του ορίου, καταγράφεται σχετική εγγραφή στον προσωρινό πίνακα VipViolations.

Indexes:

Στο πλαίσιο της υλοποίησης της βάσης δεδομένων για το Pulse Festival, πραγματοποιήθηκε ανάλυση όλων των ζητούμενων ερωτημάτων (Q01–Q15) με στόχο την επιλογή κατάλληλων ευρετηρίων (indexes) που θα βελτιώσουν την αποδοτικότητα των ερωτημάτων και τη συνολική απόδοση του συστήματος.

Παρακάτω παρουσιάζονται αναλυτικά τα indexes που δημιουργήθηκαν στη βάση, με σύντομη περιγραφή της λειτουργίας και του σκοπού τους. Ιδιαίτερη έμφαση δίνεται σε εκείνα που αξιοποιούνται άμεσα στα ερωτήματα Q01–Q15.

Τα υπόλοιπα indexes ορίστηκαν για την υποστήριξη integrity constraints, triggers και σχέσεων μεταξύ των πινάκων, εξασφαλίζοντας τη συνέπεια και σωστή λειτουργία της βάσης.

Indexes που ορίστηκαν αυτόματα και απαιτούνται για integrity/triggers

Πίνακας	Index	Queries	Αιτιολόγηση
Event	fk_Event_Festival1_idx	Q01, Q03, Q12, Q13, Q14	Εξασφαλίζει το JOIN Event→Festival
Performance	fk_Performance_Band1_idx	Q04	JOIN Performance → Performer, για integrity
Rating	fk_Rating_Performance1_idx	Q04	JOIN Rating → Performance, για integrity
Event	fk_Event_Stage1_idx		Υποστηρίζει JOIN από Event → Stage
Festival	fk_Festival_Image1_idx		Υποστηρίζει αναφορά σε εικόνες (Image → Festival)
Festival	fk_Festival_Location1_idx		JOIN Festival → Location για γεωγραφική ανάλυση
Staff	fk_Staff_Role1_idx		Επιτρέπει filtering staff ανά ρόλο
Staff	fk_Staff_Experience1_idx		Απαιτείται για constraint εμπειρίας
Artist	fk_Artist_Image1_idx		Αναφορά σε εικόνες καλλιτεχνών
Performance	fk_Performance_Performance_type1_idx		Υποστηρίζει filtering/ανάλυση ανά είδος εμφάνισης
Ticket	fk_Ticket_Event1_idx		JOIN Ticket → Event (π.χ. έλεγχος πλήθους)

Ticket	fk_Ticket_Visitor1_idx		Απαιτείται για συνδέσεις Visitor → Ticket
Ticket	fk_Ticket_Ticket_type1_idx		Ανάλυση ανά τύπο εισιτηρίου
Ticket	fk_Ticket_Payment_method1_idx		Χρήσιμο για grouped revenue ανά μέθοδο πληρωμής
Ticket	fk_Ticket_Visitor2_idx		Σύνδεση με Seller για μεταπώληση
Visitor_wants_Ticket	fk_Visitor_has_Ticket_Ticket1_idx		Αξιολόγηση διαθεσιμότητας συγκεκριμένου εισιτηρίου
Visitor_wants_Ticket	fk_Visitor_wants_Ticket_Event1_idx		JOIN προς Event για FIFO λογική
Visitor_wants_Ticket	fk_Visitor_wants_Ticket_Ticket_type1_idx		Ανάλυση προτιμήσεων εισιτηρίων
ResaleQueue	fk_ResaleQueue_Ticket1_idx		Βοηθά στον έλεγχο εισιτηρίων που μπήκαν σε μεταπώληση
Subgenre	fk_Subgenre_Genre1_idx		Υποστηρίζει σύνδεση υποείδους με είδος
Stage_has_Equipment	fk_Stage_has_Equipment_Equipment1_idx		Σύνδεση εξοπλισμού ανά σκηνή
Stage_has_Equipment	fk_Stage_has_Equipment_Stage1_idx		JOIN Stage → Stage_has_Equipment
Artist_has_Performer	fk_Artist_has_Band_Artist_idx		JOIN Artist → Performer (για είδη)

Artist_has_Performer	fk_Artist_has_Band_Band1_idx		JOIN Performer → Artist
Performer_has_Subgenre	fk_Performer_has_Subgenre_Subgenre1_idx		JOIN με subgenres για αναλύσεις ανά είδος
Event_has_Staff	fk_Event_has_Staff_Event1_idx		Υποστηρίζει JOIN Event → Event_has_Staff
Event_has_Staff	fk_Event_has_Staff_Staff1_idx		JOIN Staff → Event_has_Staff
Rating	fk_Rating_Visitor1_idx		JOIN Visitor → Rating
Rating	fk_Rating_Performance1_idx		JOIN Performance → Rating
Rating	fk_Rating_Likert1_idx έως fk_Rating_Likert5_idx		Για integrity στους βαθμούς Likert

Επιπλέον Indexes που σχεδιάστηκαν για την επιτάχυνση των queries

Πίνακας	**Index**	**Queries**	**Αιτιολόγηση**
Performer_has_Subgenre	idx_phs_performer_subgenre	Q02, Q10, Q14	Επιταχύνει join μεταξύ καλλιτεχνών και μουσικών ειδών.
Subgenre	idx_subgenre_genre_sub	Q02, Q10, Q14	Βοηθά στα join από subgenre σε genre για αναλύσεις ανά είδος.

Performance	idx_performance_performer_event	Q02, Q03, Q05, Q09, Q10, Q11, Q15	Καλύπτει JOIN με Event και Group By σε Performer.
Rating	idx_rating_visitor_perf	Q06	Αξιοποιείται σε groupings και aggregations ανά performance ή visitor.
Staff	idx_staff_role_staffid	Q07, Q12	Για filtering προσωπικού με βάση τον ρόλο.
Event_has_Staff	idx_event_staff	Q07, Q08	Επιταχύνει joins σε staff ανά event για έλεγχο απασχόλησης.

Παρατηρήσεις από τη χρήση των Indexes

Κατά τη μελέτη των query plans με την εντολή EXPLAIN, διαπιστώθηκε ότι σε κάποια ερωτήματα όπου έχουν οριστεί ευρετήρια (indexes), ο βελτιστοποιητής της MySQL τα αξιοποιεί αυτόματα, αποφεύγοντας περιττούς full table scans.

Σε επιλεγμένα queries (όπως Q04 και Q06), εφαρμόστηκε εναλλακτική εκτέλεση με FORCE INDEX για πειραματική αξιολόγηση. Παρατηρήθηκε σχετικά μικρή βελτίωση στους δείκτες απόδοσης (rows, filtered, Extra). Αυτό επιβεβαιώνει ότι το αρχικό σχέδιο εκτέλεσης ήταν ήδη αποδοτικό και τα indexes ενεργοποιούνται όπως αναμενόταν.

Παρακάτω παρατίθεται το DDL script (install.sql):

DML script

Συλλογή δεδομένων

Δεν υπάρχει έτοιμο dataset που να περιέχει τα δεδομένα που χρειαζόμαστε, οπότε φτιάξαμε ένα python script και χρησιμοποιήσαμε το DeepSeek API για να κάνουμε generate δεδομένα για τους κύριους πίνακες της βάσης μας. Έπειτα μετατρέψαμε τα αρχεία σε CSV και τα εισάγουμε στη βάση.

Μετατροπή δεδομένων
Έχοντας δεδομένα σε CSV για τους βασικούς πίνακες γράψαμε κάποια scripts (Jupyter Notebooks) ώστε να μπορέσουμε να τα επεξεργαστούμε και να κάνουμε generate με τυχαίο τρόπο, τηρώντας τους περιορισμούς, δεδομένα για τους υπόλοιπους πίνακες (που προκύπτουν από τους βασικούς).

Φόρτωση δεδομένων στη βάση
Έτσι φτάνουμε στο σημείο που έχουμε αρχεία csv για κάθε πίνακα της βάσης μας. Γράψαμε λοιπόν ένα ακόμα notebook το οποίο διαβάζει τα αρχεία αυτά και δημιουργεί ένα sql αρχείο (sql/load.sql στο repository) το οποίο περιέχει insert queries για κάθε ένα row από τα αρχεία csv. Με αυτόν τον τρόπο τρέχοντας μόνο αυτό το αρχείο φορτώνουμε τη βάση μας με τα δεδομένα.

Queries

Q01: Ζητείται να βρούμε τα συνολικά έσοδα του φεστιβάλ από πωλήσεις εισιτηρίων ομαδοποιημένα ανά έτος και τρόπο πληρωμής. Συνδέουμε τους πίνακες Ticket, Event, Festival, Payment_method και παίρνουμε μόνο τα εισιτήρια όπου το visitor_id δεν είναι null για να αθροίσουμε την τιμή τους (μόνο τα αγορασμένα εισιτήρια έχουν visitor id) και ομαδοποιούμε ανά έτος και τρόπο πληρωμής. Τέλος τα προβάλουμε σε φθίνουσα σειρά ως προς το έτος πρώτα και έπειτα ως προς το συνολικό κέρδος με τον εκάστοτε τρόπο πληρωμής.

Q02: Το ερώτημα αυτό εντοπίζει όλους τους performers που σχετίζονται με ένα συγκεκριμένο είδος μουσικής (π.χ. "Rock") και ελέγχει αν συμμετείχαν στο φεστιβάλ ενός συγκεκριμένου έτους (π.χ. 2024). Το έτος ελέγχεται δυναμικά μέσω μεταβλητής (@year) και ενσωματώνεται τόσο στο φίλτρο όσο και στον τίτλο της στήλης με χρήση δυναμικού SQL (PREPARE). Για κάθε performer εμφανίζονται το id, το όνομα, το μουσικό είδος και μία ένδειξη "Yes" ή "No" ως προς τη συμμετοχή του στο φεστιβάλ του συγκεκριμένου έτους. Το αποτέλεσμα ταξινομείται αλφαβητικά ανά performer και αφορά μόνο όσους ανήκουν στο

επιλεγμένο μουσικό είδος. Φυσικά μπορούμε να προσαρμόσουμε το είδος μουσικής και το έτος ανάλογα με τα δεδομένα που μας ενδιαφέρει να λάβουμε.

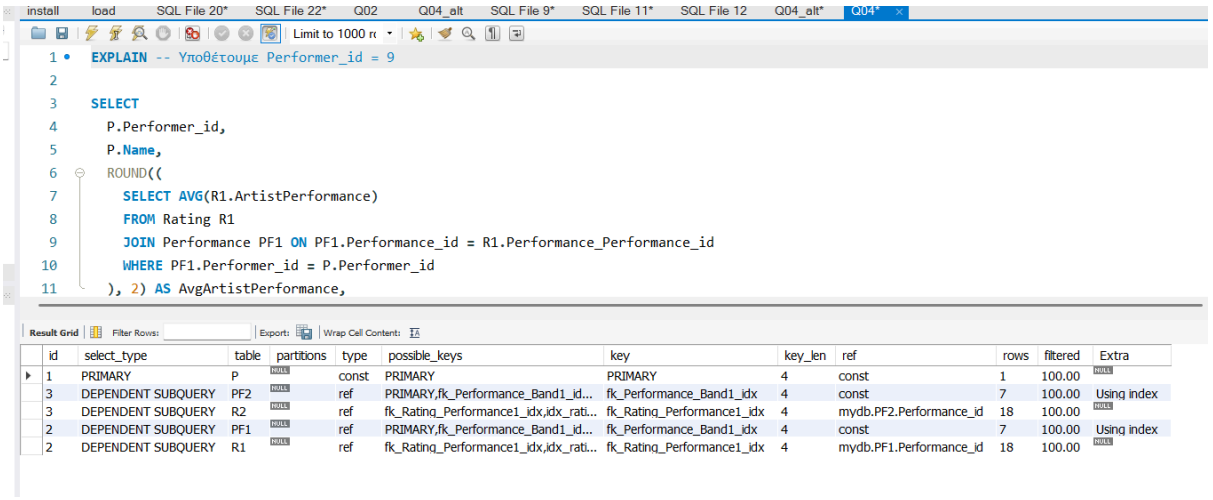
Q03: Για να βρούμε τους καλλιτέχνες (σόλο και συγκροτήματα) που έχουν εμφανιστεί περισσότερες από δύο φορές ως warm up στο ίδιο φεστιβάλ, συνδέουμε τον πίνακα Performance με το Perfromer για να βρούμε το όνομα του καλλιτέχνη (πέρα από το id), το event για να βρούμε εν τέλη το festival id καθώς θέλουμε αποτελέσματα για το ίδιο φεστιβάλ. Με το where clause επιλέγουμε μόνο τα performances τύπου 'warm up'. Χρησιμοποιούμε το index για βελτίωση απόδοσης (έχει αναλυθεί παραπάνω). Τέλος ομαδοποιούμε τα αποτελέσματα με βάση το performer id και το festival id, κρατάμε μόνο τους performers που έχουν περισσότερες από δύο εμφανίσεις warm up από αυτές που μετρήσαμε πριν και τα ταξινομούμε σε φθίνουσα σειρά warm up.

Q04: Ζητείται να υπολογιστεί ο μέσος όρος της αξιολόγησης που έλαβε κάθε performer για τις εμφανίσεις του, στα πεδία "Ερμηνεία καλλιτεχνών" και "Συνολική εντύπωση". Για τον υπολογισμό αυτό συνδέουμε τους πίνακες Rating και Performance μέσω του πεδίου Performance_Performance_id, ώστε να γνωρίζουμε ποια εμφάνιση αντιστοιχεί σε ποιον performer. Έπειτα φιλτράρουμε ως προς το Performer_id και ομαδοποιούμε ανά performer ώστε να εφαρμόσουμε τον μέσο όρο (AVG) στα αντίστοιχα πεδία αξιολόγησης.

Το ερώτημα υλοποιήθηκε με δύο τρόπους:

- Στην πρώτη προσέγγιση, χρησιμοποιούνται δύο υποερωτήματα SELECT AVG(...), ένα για κάθε είδος αξιολόγησης. Η μέση τιμή υπολογίζεται ξεχωριστά μέσω συνδέσεων Rating–Performance. Αν και το ερώτημα είναι λειτουργικό, δεν γίνεται αξιοποίηση των ευρετηρίων που έχουν οριστεί στους σχετικούς πίνακες, γεγονός που το καθιστά λιγότερο αποδοτικό σε μεγαλύτερα σύνολα δεδομένων.
- Στην δεύτερη προσέγγιση, γίνεται χρήση JOIN και GROUP BY, ενώ εφαρμόζεται και FORCE INDEX για να εξαναγκαστεί η χρήση των ευρετηρίων fk_Performance_Band1_idx και fk_Rating_Performance1_idx. Το ερώτημα εκτελείται πιο αποδοτικά, αξιοποιεί πλήρως τα ευρετήρια και κλιμακώνεται καλύτερα σε περίπτωση πολλαπλών εγγραφών ή μεγαλύτερου όγκου δεδομένων.

Ανάλυση Εκτέλεσης (EXPLAIN & TRACE)
Q04.sql (πρώτη προσέγγιση):

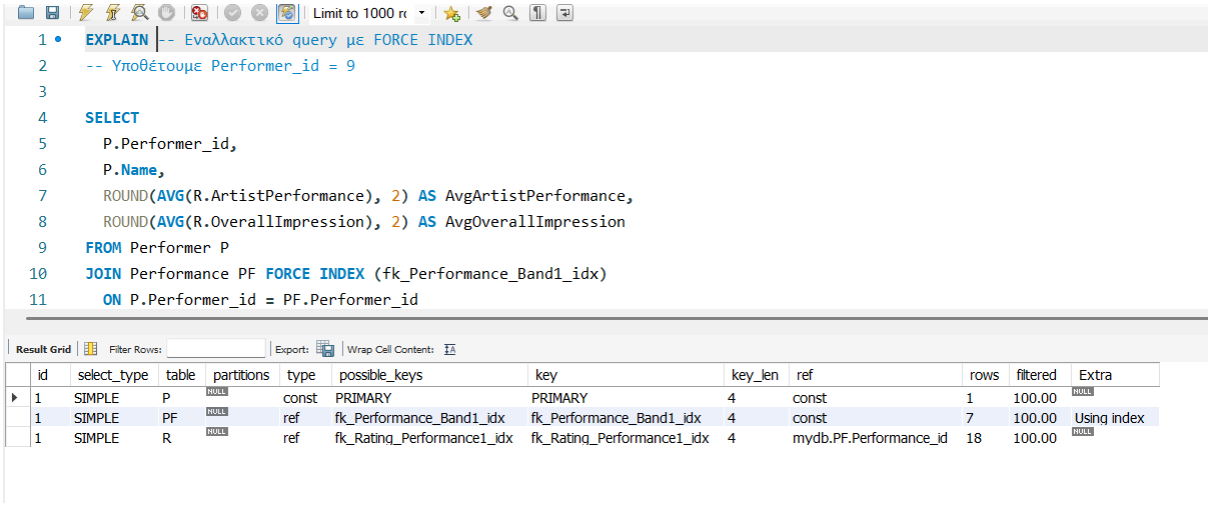


```
EXPLAIN -- Υποθέτουμε Performer_id = 9

SELECT
  P.Performer_id,
  P.Name,
  ROUND((
    SELECT AVG(R1.ArtistPerformance)
    FROM Rating R1
    JOIN Performance PF1 ON PF1.Performance_id = R1.Performance_Performance_id
    WHERE PF1.Performer_id = P.Performer_id
  ), 2) AS AvgArtistPerformance,
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	P	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
3	DEPENDENT SUBQUERY	PF2	NULL	ref	PRIMARY,fk_Performance_Band1_id...	fk_Performance_Band1_idx	4	const	7	100.00	Using index
3	DEPENDENT SUBQUERY	R2	NULL	ref	fk_Rating_Performance1_idx,idx_rati...	fk_Rating_Performance1_idx	4	mydb.PF2.Performance_id	18	100.00	NULL
2	DEPENDENT SUBQUERY	PF1	NULL	ref	PRIMARY,fk_Performance_Band1_id...	fk_Performance_Band1_idx	4	const	7	100.00	Using index
2	DEPENDENT SUBQUERY	R1	NULL	ref	fk_Rating_Performance1_idx,idx_rati...	fk_Rating_Performance1_idx	4	mydb.PF1.Performance_id	18	100.00	NULL

Q04_alt.sql (δεύτερη προσέγγιση):



```
EXPLAIN -- Εναλλακτικό query με FORCE INDEX
-- Υποθέτουμε Performer_id = 9

SELECT
  P.Performer_id,
  P.Name,
  ROUND(AVG(R.ArtistPerformance), 2) AS AvgArtistPerformance,
  ROUND(AVG(R.OverallImpression), 2) AS AvgOverallImpression
FROM Performer P
JOIN Performance PF FORCE INDEX (fk_Performance_Band1_idx)
  ON P.Performer_id = PF.Performer_id
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	P	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	PF	NULL	ref	fk_Performance_Band1_idx	fk_Performance_Band1_idx	4	const	7	100.00	Using index
1	SIMPLE	R	NULL	ref	fk_Rating_Performance1_idx	fk_Rating_Performance1_idx	4	mydb.PF.Performance_id	18	100.00	NULL

Το σχέδιο εκτέλεσης (EXPLAIN) έδειξε ότι στη δεύτερη υλοποίηση με JOIN και GROUP BY χρησιμοποιούνται κανονικά τα ευρετήρια fk_Rating_Performance1_idx και fk_Performance_Band1_idx. Η στήλη Extra περιλαμβάνει την ένδειξη Using index, γεγονός που υποδεικνύει τη χρήση covering indexes και επιβεβαιώνει την αποδοτικότητα του ερωτήματος.

Η επιβολή ευρετηρίου μέσω FORCE INDEX οδήγησε στο ίδιο πλάνο εκτέλεσης, χωρίς μεταβολή στο πλήθος προσπελαζόμενων γραμμών ή στο εκτιμώμενο κόστος, γεγονός που δείχνει ότι optimizer επέλεξε ούτως ή άλλως τα κατάλληλα ευρετήρια, άρα η σχεδίαση της βάσης είναι ήδη βελτιστοποιημένη για το συγκεκριμένο ερώτημα.

Το OPTIMIZER TRACE επιβεβαιώνει ότι τα ευρετήρια αξιολογούνται και επιλέγονται σωστά, ακόμη και χωρίς εξαναγκασμό. Συνεπώς η χρήση του FORCE INDEX δεν κρίνεται άκρως απαραίτητη στο συγκεκριμένο dataset,

αλλά προσφέρει σταθερότητα απόδοσης σε περιπτώσεις αυξημένου όγκου δεδομένων ή παρουσίας πολλαπλών ευρετηρίων στο μέλλον.

Ανάλυση Στρατηγικών JOIN και Επίδραση στην Απόδοση

Παραλλαγή	JOIN Στρατηγική	Ευρετήρια	EXPLAIN/TRACE	Σχόλια Απόδοσης
Q04.sql (αρχική)	**Nested Loop Join** (μέσω υποερωτημάτων)	Δεν χρησιμοποιούνται ευρετήρια αποτελεσματικά	DEPENDENT SUBQUERY ×3	Κάθε εγγραφή performer επανεκτελεί το υποερώτημα → αυξημένος αριθμός προσβάσεων και υπολογιστικού κόστους
Q04_alt.sql	**Merge Join με FORCE INDEX**	fk_Performance_Band1_idx, fk_Rating_Performance1_idx	JOIN με SIMPLE + Using index	Πολύ πιο αποδοτικό: ομαδοποιεί σωστά, καλύπτει τα πεδία με covering indexes, αποφεύγεται το redundancy των υποερωτημάτων

Η χρήση JOIN με GROUP BY και FORCE INDEX προσφέρει σημαντική αύξηση στην απόδοση και μειώνει τη χρήση επαναλαμβανόμενων υποερωτημάτων. Το Nested Loop υποερώτημα είναι πολύ ακριβό όταν αυξηθούν οι performer.

Q05: Θέλουμε να βρούμε τους καλλιτέχνες κάτω των 30 ετών που έχουν τις περισσότερες συμμετοχές σε φεστιβάλ. Θεωρούμε ότι καλλιτέχνες είναι είτε σόλο είτε συγκροτήματα (όπου ημερομηνία γέννησης εννοούμε ημερομηνία σχηματισμού του) και 1 συμμετοχή σε φεστιβάλ = 1 εμφάνιση. Οπότε αρχικά βρίσκουμε ποιοι καλλιτέχνες είναι σόλο (εκείνοι όπου το performer id συνδέεται μόνο με ένα artist id) και για αυτούς παίρνουμε το BirthDate από τον πίνακα Artist. Έπειτα για να γίνει σωστός υπολογισμός της ηλικίας, αναλόγως την περίπτωση, αν πρόκειται για τους solo artists που βρήκαμε πριν έχουν BirthDate οπότε υπολογίζεται από εκεί, ενώ αν πρόκειται για συγκρότημα δεν

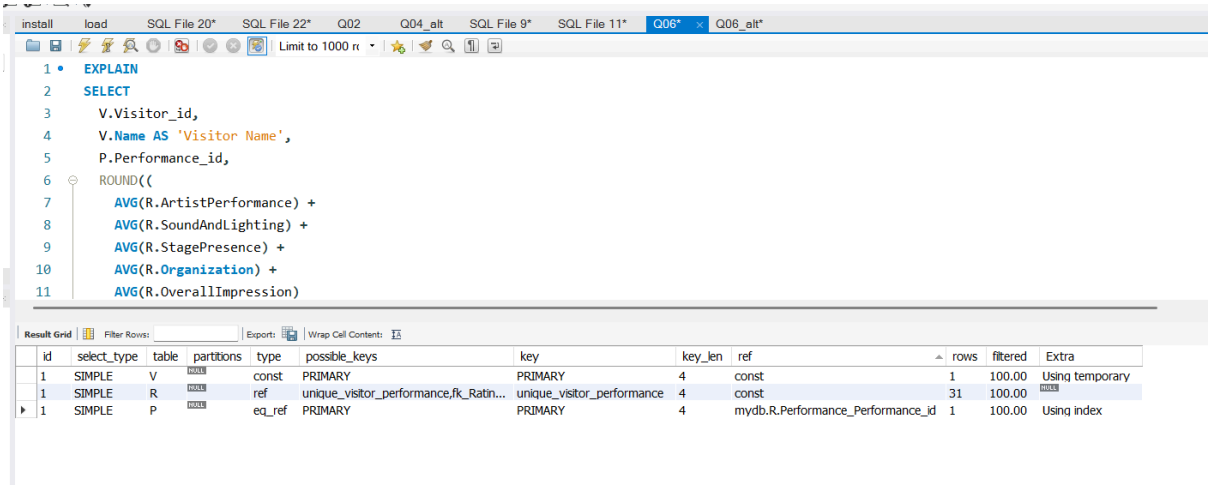
βρίσκει BirthDate και υπολογίζει τα χρόνια από το FormationDate του Performer (με το left join παίρνουμε ουσιαστικά όλους τους performers με BirthDate null ή όχι αναλόγως). Και από αυτό κρατάμε μόνο εκείνους κάτω των 30. Έπειτα κάνουμε join με τον πίνακα performance και μετράμε πόσες συμμετοχές έχει καθέ ένας από τους νέους καλλιτέχνες που βρήκαμε πριν. Τέλος εμφανίζουμε με leaderboard τα αποτελέσματα σε φθίνουσα σειρά συμμετοχών.

Q06: Ζητείται να υπολογιστεί ο μέσος όρος της αξιολόγησης που έχει δώσει ένας επισκέπτης σε κάθε εμφάνιση (performance) που παρακολούθησε, με βάση τα πέντε κριτήρια αξιολόγησης. Συνδέουμε τους πίνακες Rating, Visitor και Performance, ώστε να γνωρίζουμε ποιες εμφανίσεις έχει αξιολογήσει κάθε επισκέπτης και να εφαρμόσουμε την AVG() συνάρτηση πάνω στα επιμέρους πεδία.
Το ερώτημα υλοποιήθηκε με δύο τρόπους:

- Στην πρώτη προσέγγιση, γίνεται πλήρης σύνδεση (JOIN) μεταξύ των πινάκων Visitor, Rating και Performance, με ομαδοποίηση ανά επισκέπτη και εμφάνιση. Η σύνδεση με τον πίνακα Performance είναι χρήσιμη για την προβολή περισσότερων πληροφοριών ή για μελλοντική επέκταση του ερωτήματος, ωστόσο δεν επηρεάζει άμεσα τον υπολογισμό των μέσων όρων. Παρατηρείται χρήση προσωρινών πινάκων (Using temporary) λόγω του σύνθετου GROUP BY.

- Στη δεύτερη προσέγγιση, αποφεύγεται το επιπλέον JOIN με τον πίνακα Performance, διατηρώντας μόνο τη σύνδεση Rating–Visitor. Το ερώτημα παραμένει λειτουργικά ισοδύναμο, αλλά εκτελείται ελαφρώς πιο αποδοτικά. Επιπλέον, χρησιμοποιείται το ευρετήριο idx_rating_visitor_perf, το οποίο βελτιστοποιεί τις προσπελάσεις στον πίνακα Rating.

Ανάλυση Εκτέλεσης (EXPLAIN & TRACE)
Q06.sql (πρώτη προσέγγιση):

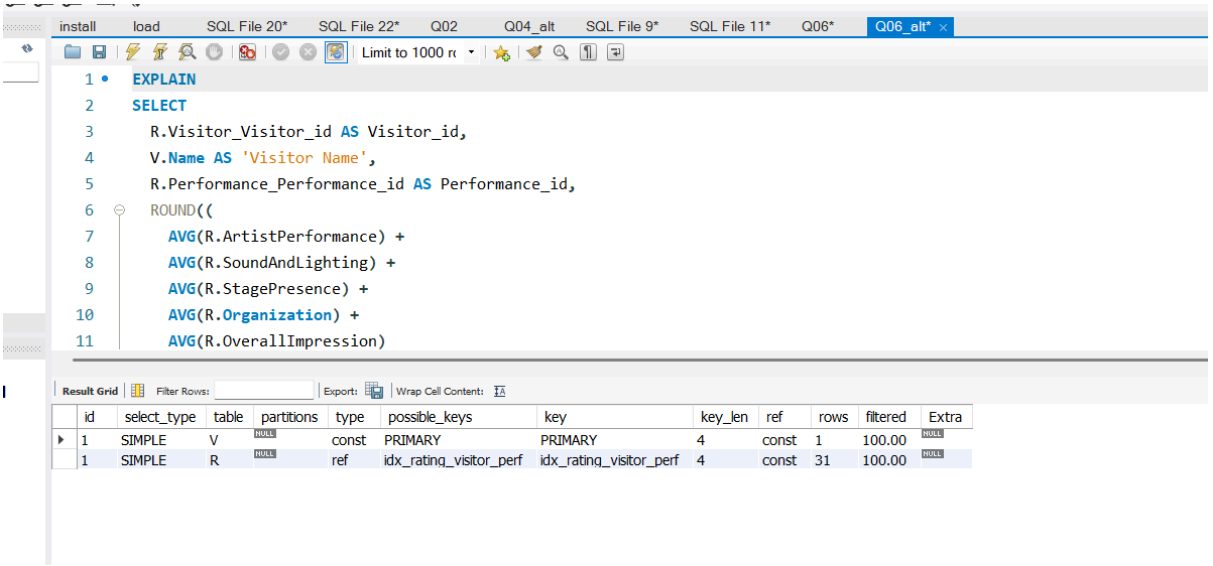


```
EXPLAIN
SELECT
  V.Visitor_id,
  V.Name AS 'Visitor Name',
  P.Performance_id,
  ROUND((
    AVG(R.ArtistPerformance) +
    AVG(R.SoundAndLighting) +
    AVG(R.StagePresence) +
    AVG(R.Organization) +
    AVG(R.OverallImpression)
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	V	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	Using temporary
1	SIMPLE	R	NULL	ref	unique_visitor_performance,fk_Ratin...	unique_visitor_performance	4	const	31	100.00	NULL
1	SIMPLE	P	NULL	eq_ref	PRIMARY	PRIMARY	4	mydb.R.Performance_Performance_id	1	100.00	Using index

Χρησιμοποιείται το index fk_Rating_Visitor1_idx, αλλά λόγω του σύνθετου JOIN, εμφανίζεται Using temporary στο Extra, κάτι που δείχνει χρήση ενδιάμεσου **temporary table**.

Q06_alt.sql (δεύτερη προσέγγιση):



```
EXPLAIN
SELECT
  R.Visitor_Visitor_id AS Visitor_id,
  V.Name AS 'Visitor Name',
  R.Performance_Performance_id AS Performance_id,
  ROUND((
    AVG(R.ArtistPerformance) +
    AVG(R.SoundAndLighting) +
    AVG(R.StagePresence) +
    AVG(R.Organization) +
    AVG(R.OverallImpression)
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	V	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	R	NULL	ref	idx_rating_visitor_perf	idx_rating_visitor_perf	4	const	31	100.00	NULL

Χρησιμοποιείται το σύνθετο ευρετήριο idx_rating_visitor_perf, με την ένδειξη Using index στο Extra, που επιβεβαιώνει χρήση covering index και αποφυγή προσωρινής αποθήκευσης. Παρατηρούμε, επίσης, ότι η χρήση **temporary tables** εμφανίζεται στην πρώτη έκδοση, ενώ στην δεύτερη όχι. Συνεπώς, το σχέδιο εκτέλεσης δείχνει ότι η δεύτερη εκδοχή του ερωτήματος είναι πιο αποδοτική, με βελτιωμένη χρήση ευρετηρίων και χωρίς δημιουργία προσωρινών δομών.

Το OPTIMIZER TRACE επιβεβαιώνει πως, στη δεύτερη υλοποίηση, ο optimizer επιλέγει άμεσα το ευρετήριο idx_rating_visitor_perf, ενώ στην πρώτη εξετάζει περισσότερες εναλλακτικές πριν καταλήξει. Συνεπώς, η δεύτερη

προσέγγιση του Q06 αποδεικνύεται αποδοτική, καθώς αξιοποιεί σωστά το ευρετήριο στον πίνακα Rating και περιορίζει την υπολογιστική επιβάρυνση του ερωτήματος.

Ανάλυση Στρατηγικών JOIN και Επίδραση στην Απόδοση

Παραλλαγή	**JOIN Στρατηγική**	**Ευρετήρια**	**EXPLAIN/TRACE**	**Σχόλια Απόδοσης**
Q06.sql (αρχική)	**Nested Loop / Merge Join** με πολλαπλά JOINs	PRIMARY, unique_visitor_performance	Χρήση Using temporary (χαμηλή απόδοση)	Η παρουσία του Performance προσθέτει περιττό βάρος. Το GROUP BY οδηγεί σε προσωρινό πίνακα
Q06_alt.sql	**Simple Hash Join ή Nested Loop** (μόνο Rating–Visitor)	idx_rating_visitor_perf (σύνθετο)	Χρήση covering index + Using index	Περιορίζεται η υπολογιστική επιβάρυνση, χωρίς δημιουργία προσωρινών δομών, ταχύτερη εκτέλεση

Η αφαίρεση περιττών JOINs και η χρήση σύνθετου ευρετηρίου οδηγεί σε καλύτερη χρήση του optimizer. Το JOIN με Performance στον αρχικό σχεδιασμό δημιουργεί προσωρινές δομές (Using temporary), που μειώνουν την αποδοτικότητα.

Συνεπώς, όσο πιο αυξημένος είναι ο όγκος των δεδομένων, τόση μεγαλύτερη σημασία στην απόδοση έχει ο δεύτερος τρόπος υλοποίησης.

Q07: Ζητείται να βρούμε το φεστιβάλ που έχει το χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού. Αρχικά στην εφαρμογή μας τεχνικό προσωπικό θεωρούμε τους ‘Sound Engineer’, ‘Lighting Technician’ και ‘Stage Manager’. Κάνουμε join το φεστιβάλ με τα events, τα events και το staff με το event_has_staff και από το staff επιλέγουμε μόνο το προσωπικό που έχει τιμή

στο Role_name μία από τις παραπάνω ονομασίας. Υπολογίζουμε τους μέσους όρους εμπειρίας τους ανά φεστιβάλ, τα ταξινομούμε από το μικρότερο στο μεγαλύτερο και παίρνουμε μόνο το πρώτο.

Q08: Θέλουμε να βρούμε ποια μέλη του προσωπικού υποστήριξης δεν έχουν προγραμματιστεί να εργαστούν σε κάποια συγκεκριμένη ημερομηνία. Προσωπικό υποστήριξης θεωρούνται οι εργαζόμενοι με ρόλο 'Artist Liaison', 'Cleanup Crew', 'Ticket Scanner', 'Vendor Coordinator' και 'Parking Attendant'. Κάνουμε αρχικά επιλογή όλων των εργαζομένων που ανήκουν σε μία από αυτές τις κατηγορίες. Στη συνέχεια, μέσω συσχέτισης με τον πίνακα Event_has_Staff και τα αντίστοιχα events, αποκλείουμε από τα αποτελέσματα όσους έχουν ήδη τοποθετηθεί σε κάποιο event που διεξάγεται τη συγκεκριμένη ημερομηνία (ημερομηνία που δηλώνεται μέσα στο ερώτημα, π.χ. '2025-06-20'). Το τελικό αποτέλεσμα περιλαμβάνει τα Staff_id και τα ονόματα όλων των διαθέσιμων μελών προσωπικού που δεν έχουν ανατεθεί να εργαστούν εκείνη τη μέρα.

Q09: Θέλουμε να βρούμε ποιοί επισκέπτες έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με πάνω από 3 παρακολουθήσεις. Αρχικά στο εσωτερικό select παίρνουμε για κάθε επισκέπτη το όνομα του, τη χρονία και πόσες παραστάσεις έχει παρακολουθήσει. Ελέγχουμε εννοείται αν το εισιτήριο του ήταν ενεργοποιημένο για να θεωρήσουμε ότι την παρακολούθησε και την χρονιά τη βρίσκουμε από την ημερομηνία του event. Τέλος φιλτράρουμε και κρατάμε μόνο εκείνους που έχουν παρακολουθήσει πάνω από 3 παραστάσεις. Στο εξωτερικό select βρίσκουμε ποιοί επισκέπτες (ονόματα) είχαν ακριβώς τον ίδιο αριθμό παρακολουθήσεων για κάθε συνδυασμό χρονιάς και πλήθους παραστάσεων (πάνω από 3 όμως). Τα εμφανίζουμε σε leaderboard με αύξουσα σειρά χρονιάς.

Q10: Θέλουμε να εντοπίσουμε τα 3 ζεύγη μουσικών ειδών (genres) που εμφανίζονται πιο συχνά μαζί σε φεστιβάλ. Πολλοί καλλιτέχνες σχετίζονται με περισσότερα από ένα είδη μουσικής, οπότε ελέγχουμε για κάθε performer όλους τους συνδυασμούς ειδών που συνδέονται με τα subgenres του. Μέσω LEAST και GREATEST οργανώνουμε τα ζεύγη ώστε να μην υπολογίζονται διπλά (π.χ. Rock-Jazz και Jazz-Rock). Έπειτα γίνεται σύνδεση με τις εμφανίσεις (Performance), τα αντίστοιχα events και τα φεστιβάλ, ώστε να μετρηθεί για κάθε ζεύγος σε πόσα φεστιβάλ έχει εμφανιστεί. Το αποτέλεσμα ταξινομείται

κατά φθίνουσα σειρά με βάση τις συνολικές εμφανίσεις σε φεστιβάλ και επιστρέφονται μόνο τα 3 πιο συχνά εμφανιζόμενα ζεύγη.

Q11: Θέλουμε να βρούμε όλους τους καλλιτέχνες που συμμετείχαν στο φεστιβάλ τουλάχιστον 5 λιγότερες φορές από τον καλλιτέχνη με τις περισσότερες φορές σε φεστιβάλ. Ισχύουν οι ίδιες υποθέσεις για καλλιτέχνες και συμμετοχές όπως αναλύσαμε σε προηγούμενο query. Κάνουμε join τους πίνακες Performer και Performance και υπολογίζουμε πόσες εμφανίσεις έχει κάθε καλλιτέχνης (το όνομα του καλλιτέχνη το παίρνει χάρη στο join). Έπειτα βρίσκουμε τον performer με τις περισσότερες συμμετοχές. Και τέλος επιλέγουμε όλους τους καλλιτέχνες από τους παραπάνω όπου έχουν διαφορά τουλάχιστον 5 εμφανίσεων με τον top. Επιστρέφουμε στο performer id και το όνομα του καθώς και τον αριθμό των performances του (σε φθινουσα σειρά με βάση αυτό).

Q12: Θέλουμε να υπολογίσουμε το προσωπικό που απαιτείται ανά ημέρα του φεστιβάλ, κατηγοριοποιημένο με βάση τον ρόλο του. Αρχικά συνδέουμε τα events με τα μέλη του προσωπικού που έχουν ανατεθεί μέσω του πίνακα Event_has_Staff, και από εκεί προχωρούμε στο Staff για να λάβουμε τον ρόλο και το όνομα κάθε εργαζομένου. Οι ρόλοι ταξινομούνται σε τρεις βασικές κατηγορίες: **τεχνικό προσωπικό** *('Lighting Technician', 'Sound Engineer', 'Stage Manager')*, **προσωπικό ασφαλείας** *('Security', 'First Aid Responder')* και **βοηθητικό προσωπικό** *('Artist Liaison', 'Cleanup Crew', 'Ticket Scanner', 'Vendor Coordinator' και 'Parking Attendant')*. Η ημερομηνία λαμβάνεται από την εκάστοτε εγγραφή του πίνακα Event, ενώ για κάθε κατηγορία και ημερομηνία υπολογίζεται ο αριθμός μελών προσωπικού και εμφανίζεται συγκεντρωτικά και ονομαστικά. Το αποτέλεσμα παρέχει μια πλήρη εικόνα της στελέχωσης του φεστιβάλ ανά κατηγορία και ημερομηνία, ταξινομημένη χρονολογικά.

Q13: Θέλουμε να βρούμε τους καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους. Οπότε κάνουμε join τον πίνακα performance με τον performer για να βρούμε τα ονόματα των καλλιτεχνών, με το event καθώς κάθε performance είναι σε ένα event και έπειτα με το festival και το location για να βρούμε εν τέλη τις ηπείρους των φεστιβάλ. Για κάθε performer μετράει σε πόσες διαφορετικές ηπείρους έχει εμφανιστεί και επιλέγει μόνο εκείνους με πλήθος τουλάχιστον 3. Τέλος τα ταξινομούμε κατά

φθίνουσα σειρά αριθμού εμφανίσεων (έπειτα αλφαβητικά αν πρόκειται για ισοψηφία).

Q14: Θέλουμε να εντοπίσουμε ποια μουσικά είδη (Genre) είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενα έτη, με την προϋπόθεση ότι είχαν τουλάχιστον 3 εμφανίσεις σε κάθε έτος. Αρχικά, χρησιμοποιούμε ένα CTE (WITH GenreYearCounts) για να υπολογίσουμε πόσες εμφανίσεις είχε κάθε είδος μουσικής ανά έτος. Αυτό επιτυγχάνεται με join από τις εμφανίσεις (Performance) προς τον performer, τα υποείδη και τα είδη, και έπειτα προς τα events και τα αντίστοιχα φεστιβάλ, απ' όπου λαμβάνεται το έτος. Στο κύριο ερώτημα γίνεται self-join του CTE για να συγκριθούν τα δεδομένα κάθε είδους μεταξύ διαδοχικών ετών (π.χ. 2022 και 2023). Επιστρέφονται μόνο εκείνα τα είδη που εμφανίστηκαν τις ίδιες ακριβώς φορές και τις δύο χρονιές, και τουλάχιστον 3 φορές ανά έτος. Το αποτέλεσμα περιλαμβάνει το είδος, τα δύο έτη, και τον αριθμό εμφανίσεων που ήταν κοινός.

Q15: Ζητείται να βρούμε τους top-5 επισκέπτες που έχουν δώσει συνολικά τη μεγαλύτερη βαθμολογία σε έναν καλλιτέχνη. Κάθε καλλιτέχνης που παρακολουθεί μία παράσταση μπορεί να την βαθμολογήσει για 5 διαφορετικά χαρακτηριστικά 2 από τα οποία αφορούν τον καλλιτέχνη. Οπότε αυτά (ArtistPerformance και OverallImpression) θα αθροίσουμε για κάθε συνδυασμό επισκέπτη και performer που υπάρχει. Τα ομαδοποιούμε ανά επισκέπτη και καλλιτέχνη και κρατάμε μόνο τους top-5 συνδιασμούς επισκέπτη-καλλιτέχνη με τη μεγαλύτερη συνολική βαθμολογία.

Στα πλαίσια της εργασίας χρησιμοποιήθηκαν LLM έχοντας συμβουλευτικό και βοηθητικό ρόλο. Για παράδειγμα, για την παραγωγή των δεδομένων, για debugging, για τον βοηθητικό κώδικα που γράψαμε για autogenerate τους υπόλοιπους πίνακες και βελτιστοποιήσεις.

Βιβλιογραφία

[1] Συστήματα Βάσεων Δεδομένων - Η Πλήρης Θεωρία των Βάσεων Δεδομένων - 7η Έκδοση, Abraham Silberschatz, Henry F. Korth, S. Sudarshan

[2] Υλικό εργαστηρίου μαθήματος από το Helios