Εξαμηνιαία Εργασία ΒΔ 2024-2025
Ομάδα 103
Αλεξίου Περικλής 03122079
Αναγνωστοπούλου Άννα Ειρήνη 03122004
Τσιμπλάκη Πηνελόπη-Άννα 03122097

Πρόλογος

Η παρούσα αναφορά αποτελεί την συστηματική τεκμηρίωση της εξαμηνιαίας εργασίας του μαθήματος Βάσεις Δεδομένων. Συγκεκριμένα, αφορά λεπτομέρειες σχετικά με τον σχεδιασμό και το σκεπτικό πίσω από την δομή αυτής, τις παραδοχές εργασίας και την υλοποίηση της βάσης δεδομένων για το μουσικό φεστιβάλ Pulse University. Κώδικας δεν θα παρουσιαστεί για τα βήματα κατασκευής της βάσης για λόγους ευανάγνωσης, καθώς αυτός βρίσκεται ολόκληρος στο install.sql.

Μέρος της εργασίας είναι και τα 15 queries για το οποία γίνεται μια ανάλυση, χωρίς πάλι να παρατίθεται κώδικας. Ο κώδικας βρίσκεται στο σχετικό φάκελο στο github μαζί με τα αποτελέσματα των queries που αποθηκεύτηκαν με τη μορφή .csv για να μπορούν να διαβάζονται πιο εύκολα.

Παραδοχές

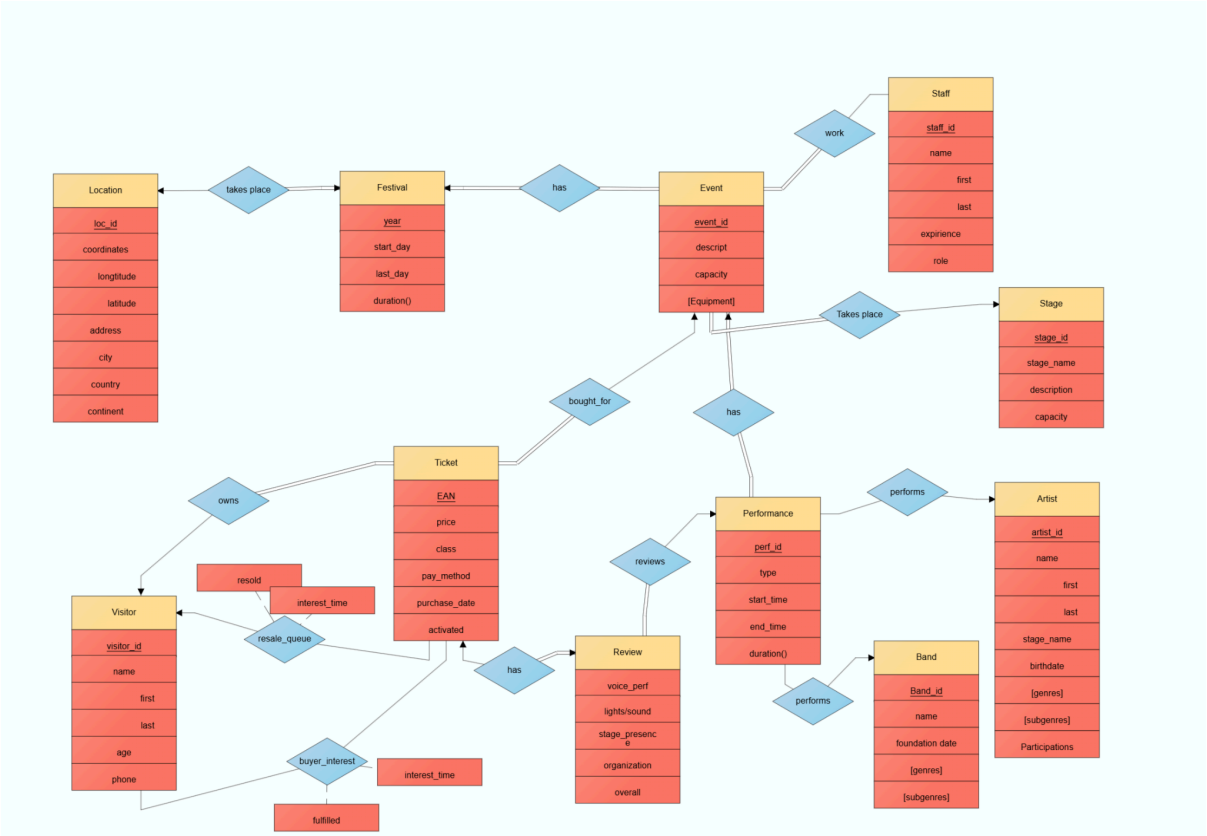
Μία διευκρίνιση που χρειάζεται να γίνει είναι πως αναφερόμαστε ως performer στους καλλιτέχνες/συγκροτήματα που συμμετέχουν στα performances, οι καλλιτέχνες καλούνται artists και τα συγκροτήματα bands. Επομένως, πολλές φορές η ερμηνεία της λέξης "καλλιτέχνης" στην εκφώνηση μπορεί να είναι performer ή artist, ανάλογα με το τι μας ταίριαζε περισσότερο νοηματικά.

Ακόμη, υποθέτουμε πως το φεστιβάλ γίνεται μία φορά το χρόνο τους καλοκαιρινούς μήνες επομένως στο query 9 το διάστημα ως έτους δεν περιέχει ποτέ 2 φεστιβάλ μαζί.

Διαδικασία δημιουργίας βάσης

1. Διάγραμμα οντοτήτων-συσχετίσεων

Το διάγραμμα E/R έγινε πρώτα στο χαρτί και από κάθε μέλος της ομάδας ξεχωριστά. Με τη συνάθροιση αυτών καταλήξαμε στο παρακάτω διάγραμμα που υλοποιήθηκε με την βοήθεια της εφαρμογής ERD Maker[1].



[1] ERD Maker - Home

2. Δημιουργία του Schema

Επόμενο βήμα είναι η δημιουργία του schema της βάσης. Για το schema χρειάστηκε να γίνουν κάποιες αλλαγές στις τις σχέσεις που ορίζει το E/R διάγραμμα ή να προστεθούν κάποιοι πίνακες, οι οποίες φαίνονται και στο relational διαγραμμα παρακάτω. Έγινε διαχωρισμός της βάσης στα τρία και το κάθε μέλος έγραψε το μέρος του. Ο συντονισμός και η συνένωση έγινε μέσω του github. Μόλις είχαν δημιουργηθεί όλα τα tables , η βάση δοκιμάστηκε στο phpMyAdmin.

Παρακάτω γίνεται μία περιγραφή των tables που δημιουργήθηκαν μαζί με μερικά primary keys, constraints και παραδοχές που έγιναν κατά την σχεδίαση.

Table	Παρατηρήσεις	Παραδοχές
Festival	Ως primary key χρησιμοποιήσαμε το έτος του φεστιβάλ καθώς είναι μοναδικό κάθε χρόνο.	Τα φεστιβάλ διαρκούν 2-5 μέρες κατα την διάρκεια των καλοκαιρινών μηνών.
Continent	Βοηθητικός πίνακας με αποθηκευμένες τις ηπείρους ώστε να αποφεύγονται τα redundancies και να είναι εγγυημένη η ορθότητα των δεδομένων.	
Location	Ο πίνακας location διαθέτει τα χαρακτηριστικά της εκφώνησης και συνδέεται με το festival μέσω foreign key. Μια τοποθεσία δεν γίνεται να διαγραφεί ενώ αν γίνει update θα αλλάξει διαδοχικά η τιμή σε όλες τις οντότητες που περιέχουν το loc_id.	
Description	Η περιγραφή είναι μία	

	σύντομη πρόταση που μπορεί να περιγράφει μοναδικά μία μουσική σκηνή π.χ open-air.	
Stage	Κρίναμε αναγκαίο να υπάρχει ξεχωριστό table για την μουσική σκηνή καθώς σε αυτές συμβαίνουν οι παραστάσεις και περιλαμβάνουν πληροφορίες σχετικά με την χωρητικότητα και το όνομα.	
Equipment	Περιέχει ορισμένες στήλες με εξοπλισμό που μπορεί να έχει μία παράσταση και αντιστοιχίζεται με ένα event_id μέσω της σχέσης EventEquip που είναι πολλά προς πολλά.	
Staff	Περιέχει πληροφορίες σχετικά με το προσωπικό όπως αυτές αναγράφονται στην εκφώνηση.	Χρησιμοποιήσαμε τον ακέραιο xp για την αναπαράσταση της εμπειρίας καθώς είναι πιο εύκολη η εγγραφή query με σύγκριση με νούμερα αντί για strings.
Role	Σε αυτόν τον πίνακα αποθηκεύονται οι 3 ρόλοι που μπορεί να έχει ένα μέλος του προσωπικού, δηλαδή τεχνικό, βοηθητικό και ασφαλείας.	
Experience	Αντιστοιχεί τους αριθμούς (1,2,3,4,5) με (Beginner, Intermediate, Advanced, Expert, Veteran)	

Employment	Ενδιάμεσος πίνακας που λειτουργεί ώς το συμβόλαιο μεταξύ συγκεκριμένου staff_id και event_id.	Ένα μέλος του προσωπικού μπορεί να εργάζεται χωρίς περιορισμό σε αριθμό events, αρκεί να μην είναι επικαλυπτόμενα.
Event_P	Η παράσταση συνδέεται με foreign keys με το φεστιβάλ και την μουσική σκηνή.	
EventEquip	Ενδιάμεσος πίνακας για αντιστοίχιση event με equipment. Σε ένα event μπορούν να χρησιμοποιηθούν πολλά equipment.	
Artist	Σε αυτόν τον πίνακα αποθηκεύονται όλοι οι καλλιτέχνες και τα μέλη συγκροτημάτων, ανεξάρτητα από το αν έχουν λάβει μέρος σε παράσταση ακόμα.	
Band	Περιέχει στοιχεία για τις μπάντες.	
BandMembers	Ο ενδιάμεσος πίνακας που αντιστοιχίζει artist_id με band_id.	
Genre	Περιέχει τα βασικά είδη μουσικής ώστε αυτά να μπορούν να αντιστοιχηθούν με τα subgenre.	
Subgenre	Περιέχει τα υποείδη μουσικής.	

Performance	Περιέχει το event_id ως foreign key καθώς μία ερμηνεία αντιστοιχεί σε μοναδικό event_id. Η διάρκεια της ερμηνείας υπολογίζεται ως συνάρτησης του start και end time και έχει τηρηθεί με CHECK να διαρκεί λιγότερο από 3 ώρες και με trigger το να βρίσκεται μέσα στο χρονικό πλαίσιο του event.	
PerformanceTypes	Περιέχει τα καθορισμένα είδη παραστάσεων δηλαδή τα ('warm-up', 'acoustic set', 'main act', 'opening act', 'release party', 'tribute performance')	
Performer	Αποτελεί είτε solo artist είτε band.	Χρησιμοποιήσαμε μία TINY INT μεταβλητή, η οποία είναι 1 όταν ερμηνεύει solo και 0 όταν ερμηνεύει band.
PerformerSubgenre	Ενδιάμεσος πίνακας που συνδέει performer_id με subgenre_id καθώς είναι σχέση πολλά προς πολλά.	
Years	Βοηθητικός πίνακας που συγκεντρώνει τις χρονιές που έχει τρέξει το φεστιβάλ.	

PerformerYears	Βοηθητικός πίνακας που συνδέει έναν performer_id με το ποιες χρονιές έχει ερμηνεύσει σε φεστιβάλ.	Χρησιμεύει στον περιορισμό του να μην συμμετέχει ένας καλλιτέχνης για παραπάνω από 3 συνεχόμενες χρονιές και σε μερικά queries. Αν κάποιος έχει συμμετάσχει σε πολλές ερμηνείες σε έναν χρόνο θα εμφανίζεται αντίστοιχο αριθμό φορών αυτή η χρονολογία στον πίνακα καθώς αυτό βοηθά στην εύρεση ερωτημάτων.
Visitor	Περιλαμβάνει πληροφορίες σχετικά με τους επισκέπτες του φεστιβάλ.	Δεν είναι αναγκαστικό να έχουν αγοράσει εισιτήριο.
Ticket	Συνδέεται με μοναδικό event_id και visitor_id. Περιέχει την τιμή activated για το αν έχει ενεργοποιηθεί το εισιτήριο, το οποίο χρησιμοποιούμε σαν έλεγχο στην ουρά μεταπώλησης και τα reviews.	Θεωρούμε πως οι επισκέπτες μπορούν να έχουν αγοράσει εισιτήριο για παραστάσεις που επικαλύπτονται χρονικά.
Ticket_Class	Πίνακας που περιέχει τις 4 κλάσεις εισιτηρίων general,premium,vip, backstage.	
PayedWith	Πίνακας που περιέχει τους τρόπους πληρωμής όπως credit,debit,paypal και cash.	

Resale_queue	Η ουρά μεταπώλησης περιέχει το EAN του εισιτηρίου που διατίθεται προς πώληση και την ημερομηνία εκδήλωσης ενδιαφέροντος πώλησης.	Το πεδίο sold παίρνει τιμές 0 ή 1 και δείχνει αν έχει πωληθεί ένα συγκεκριμένο εισιτήριο.
Buyer	Περιέχει είτε τιμές σχετικά με το event_id και το ticket_class είτε την τιμή ενός συγκεκριμένου EAN που επιθυμεί ο αγοραστής.	Θεωρούμε πως κάποιος ήδη υπάρχων επισκέπτης μπορεί να γίνει buyer μεταπουλημένου εισιτηρίου. Η επιτυχημένη πωληση σε αγοραστή ενημερώνει την τιμή του πεδίου satisfied από 0 σε 1.
Review	Αυτός ο πίνακας αντιστοιχεί συγκεκριμένη τούπλα (εισιτήριο, ερμηνεία) με τα 5 πεδία βαθμολόγησης τα οποία είναι voice, light_sound, stage_presence, organisation, overall.	
Likert	Πίνακας με πεδίο που παίρνει τιμές 1-5 ώστε να μην έχουμε redundancies.	Με check ελέγχουμε να είναι μέσα στο όριο τιμών.

3. Δημιουργία των Views

Δεδομένων των queries που θα χρειαστεί να υλοποιήσουμε (όπως άμεση απεικόνιση ανάμεσα σε έναν performer και των genre μουσικής που παίζει) και με βάση τη νοηματική σχέση μεταξύ κάποιων οντοτήτων (όπως η πιθανή ανάγκη να δούμε τη σχέση του επισκέπτη και των εισιτηρίων που έχει αγοράσει), αποφασίστηκε να κάνουμε μερικά views.
Στην αναφορά θα παρουσιαστούν τα ονόματα τους και τι προβάλλουν.

A/A	Views	Purpose
1	PerformerGenre	To make a virtual table that connects the performer to their genres
2	EventStaff	To make a virtual table that connects an event to the staff that work there
3	PerformerPerformance	To make a virtual table that connects the performer to the performances and events he plays in.
4	VisitorTicket	To make a virtual table that connects a visitor to the tickets that he has bought over the years.
5	FestivalPerformer	To make a virtual table that shows a festival and all the performers that have performed in that.

4. Δημιουργία των Triggers

Για την ορθή λειτουργικότητα της βάσης, ήταν σημαντική η δημιουργία των triggers. Αυτά χρειαζόντουσαν είτε για την νοηματική συνοχή των δεδομένων της βάσης (όπως το ένας άνθρωπος να μην μπορεί να βρίσκεται σε δύο μέρη την ίδια στιγμή), είτε για την εφαρμογή κάποιων περιορισμών (όπως το ένας επισκέπτης να μην μπορεί να αγοράσει πάνω από ένα εισιτήριο για την ίδια παράσταση).

Παρακάτω παρατίθεται ένας πίνακας με τα ονόματα των triggers της βάσης μας καθώς και την λειτουργία που εξυπηρετούν.

A/A	Trigger	Function
1	artist_or_band	Ελέγχει ένας performer να μην είναι artist και band παράλληλα
2	performance_in_event	Ελέγχει ένα performance να είναι εντός των χρονικών ορίων του event στο οποίο ανήκει.
3	check_stage_overlap	Ελέγχει μία σκηνή να μην στεγάζει δύο events ταυτόχρονα.
4	check_sec_aux_staff	Ελέγχει ένα event να καλύπτει τις προβλεπόμενες ανάγκες του σε προσωπικό ασφαλείας και βοηθητικό προσωπικό. Ο έλεγχος γίνεται πριν την πώληση εισιτηρίων για το αντίστοιχο event.
5	staff_overlap	Ελέγχει το προσωπικό να μην εργάζεται σε δύο σκηνές ταυτόχρονα.
6	break	Ελέγχει να υπάρχει το απαιτούμενο διάλειμμα μεταξύ των performances
7	check_ticket_count	Ελέγχει να μην πωλούνται εισιτήρια αφού έχουν εξαντληθεί.
8	check_vip_tickets	Ελέγχει να μην πωλούνται VIP εισιτήρια αφού έχουν εξαντληθεί.
9	check_activated_review	Ελέγχει ένα review να προέρχεται από ενεργοποιημένο εισιτήριο.
10	delete_performances	Απαγορεύει την διαγραφή προγραμματισμένων performances.
11	check_double_perform	Ελέγχει ένας artist να μην βρίσκεται σε δύο σκηνές ταυτόχρονα, ανεξάρτητα με το αν είναι ο ίδιος performer ή κάποιο συγκρότημα στο οποίο ανήκει.
12	check_consecutive_years	Απαγορεύει σε έναν performer να παίξει πάνω από τις χρονιές συνεχόμενα.
13	update_ids	Ενημερώνει τα artist_id και band_id στον πίνακα Performer αν γίνει insert μόνο με το όνομα του artist ή του band
14	update_years	Ενημερώνει τον πίνακα με τις χρονιές στις οποίες υπάρχει προγραμματισμένο φεστιβαλ.

15	update_perf_years	Ενημερώνει τον βοηθητικό πίνακα PerformerYears όταν προγραμματίζεται εμφάνιση για κάποιον performer.
16	update_part	Ενημερώνει τις συμμετοχές κάθε artist όταν προγραμματίζεται ένα performance στο οποίο συμμετέχει.
17	check_event_sold_out	Ελέγχει να μην μπορεί να πωληθεί ένα εισιτήρια αν το performance δεν έχει γίνει ακόμα sold out.
18	remove_from_resale_if_activated	Διαγράφει ένα εισιτήριο από το resale queue αν ο αρχικός αγοραστής το εξαργυρώσει.
19	match_on_resale	Κάνει την αντιστοίχιση και μεταφορά εισιτηρίου όταν υπάρχει υπάρχει ενδιαφέρον για ένα εισιτήριο που μόλις μπήκε στο resale queue.
20	match_on_buyer	Κάνει την αντιστοίχιση και μεταφορά εισιτηρίου όταν υπάρχει διαθέσιμο εισιτήριο μόλις εξέφρασε κάποιος ενδιαφέρον γι' αυτό.
21	review_attended_performance	Δεν επιτρέπει σε επισκέπτη να αξιολογήσει performance στο οποίο δεν έχει πάει.
22	future_activated_insert	Δεν επιτρέπει την δημιουργία επικυρωμένων εισιτηρίων για μελλοντικά events.
23	future_activated_update	Δεν επιτρέπει την επικύρωση εισιτηρίων για μελλοντικά events.
24	two_tickets_update	Απαγορεύει σε έναν επισκέπτη να αγοράσει πάνω από ένα εισιτήριο για ένα event μέσω του resale queue.
25	two_tickets_insert	Απαγορεύει σε έναν επισκέπτη να αγοράσει πάνω από ένα εισιτήριο για ένα event.

5. Δημιουργία των Indexes

Τα indexes είναι σημαντικά για την αποδοτικό ψάξιμο στα tables και για το optimization των queries Q04 και Q06. Τα indexes που δημιουργήσαμε βασίστηκαν κυρίως στις απαιτήσεις των queries και τις ανάγκες των triggers.

A/A	Index	Function
1	idx_artist_participations ON Artist(participations)	Χρησιμοποιείται στο optimization του query 11 καθώς από 90 γραμμές γίνεται scan μόνο των 66 με χρήση του index. Αυτό έγινε επειδή εντοπίζονται πιο γρήγορα οι γραμμές που ικανοποιούν τους περιορισμούς και επιστρέφονται ήδη ordered με βάση τα participations (Q11).
2	idx_artist_birthDate ON Artist(birthDate)	Χρησιμοποιείται για το optimization του Q05.
3	idx_performer_name ON Performer(performer_name)	Χρησιμοποιείται για το optimization του Q04.
4	idx_performance_start_end_time ON Performance(start_time, end_time)	Χρησιμοποιείται σε διάφορα triggers όπως το performance_in_event
5	idx_ticket_opt ON Ticket(visitor_id, activated, event_id)	Χρησιμοποιείται για το optimization του Q06

Φυσικά, πέρα από τα παραπάνω indices υπάρχουν και αυτά που δημιουργούνται αυτόματα μέσω των foreign keys τα οποία δεν ονομάσαμε συγκεκριμένα αλλά ενεργοποιούνται κατά το τρέξιμο των queries όπως θα δούμε και στην ανάλυση των Q04 και Q06 παρακάτω.

6. Δημιουργία των queries

- **Q01 - Έσοδα του φεστιβάλ, ανά έτος από την πώληση εισιτηρίων, λαμβάνοντας υπόψη όλες τις κατηγορίες εισιτηρίων και παρέχοντας ανάλυση ανά είδος πληρωμής.**

  Χρησιμοποιήθηκε το aggregate function sum ώστε να βρεθεί το συνολικό ποσό και με επιπλέον χρήση της CASE WHEN ... THEN έγινε η ανάλυση ανα είδος. Λόγω του aggregate function ήταν απαραίτητο το group by year ώστε να ξέρει η MySQL ως προς τι να προσθέσει τις τιμές των εισιτηρίων. Τέλος για να είναι πιο ευανάγνωστο το αποτέλεσμα έγινε order by year.

- **Q02 - Οι καλλιτέχνες που ανήκουν σε ένα συγκεκριμένο μουσικό είδος με ένδειξη αν συμμετείχαν σε εκδηλώσεις του φεστιβάλ για το συγκεκριμένο έτος.**

  Στο συγκεκριμένο query θεωρούμε ως καλλιτέχνες τους performers.
  Χρησιμοποιήθηκε ο βοηθητικός πίνακας PerformerYears ώστε να ελέγξουμε με χρήση της IF(py.years_id, 'YES', 'NO') αν ο καλλιτέχνης έχει συμμετάσχει στο έτος που προσδιορίζουμε manually. Είναι αναγκαία η χρήση Left Join για το PerformerYears επειδή χωρίς αυτό δεν θα εμφανίζονταν οι καλλιτέχνες που δεν έχουν συμμετέχει το συγκεκριμένο έτος.

- **Q03 - Οι καλλιτέχνες που έχουν εμφανιστεί ως warm up περισσότερες από 2 φορές στο ίδιο φεστιβάλ.**

  Πάλι θεωρούμε πως καλλιτέχνες είναι οι performers.
  Χρησιμοποιούμε το aggregate function count(*) για να μετρήσουμε τις γραμμές που θα παραχθούν από το view PerformerPerformance συνενωμένο με το event_id το οποίο περιέχει την πληροφορία για το έτος του φεστιβάλ. Όπως και στο Q01 είναι απαραίτητο το group by και το filtering γίνεται με το having.

- **Q04 - Για κάποιο καλλιτέχνη, βρείτε το μέσο όρο αξιολογήσεων (Ερμηνεία καλλιτεχνών) και εμφάνιση (Συνολική εντύπωση)**

  Καλλιτέχνες θεωρούμε τους performers.

Κάνουμε συνένωση του πίνακα Review με το view PerformerPerformance ώστε να αντιστοιχηθεί ο performer_id με το performance_id για το οποίο είναι γραμμένο το review. Για τον μέσο όρο γίνεται χρήση της AVG().

- **Q05 - Οι νέοι καλλιτέχνες (ηλικία < 30 ετών) που έχουν τις περισσότερες συμμετοχές σε φεστιβάλ.**

  Εδώ προσεγγίζουμε τους καλλιτέχνες ως artists.
  Χρησιμοποιήθηκε η εντολή DATE_SUB(CURDATE(), INTERVAL 30 YEAR) καθώς έτσι γίνεται χρήση του idx_artist_birthDate και αντί για full table scan γίνονται scanned μόνο 8 γραμμές. Αν χρησιμοποιούταν η συνάρτηση YEAR() για την εύρεση της ηλικίας θα ήταν αναγκαίος ο διαπερασμός όλου του πίνακα.

- **Q06 - Οι παραστάσεις που έχει παρακολουθήσει κάποιος επισκέπτης και ο μέσος όρος της αξιολόγησης του, ανά παράσταση.**

  Για να βρεθούν οι παραστάσεις που έχει παρακολουθήσει ο visitor_id=1 που επιλέξαμε, διατρέξαμε όλα τα εισητήρια του που ήταν activated και συνενώσαμε με τα reviews που είχαν το ίδιο EAN. Το left join είναι αναγκαίο καθώς έτσι εμφανίζονται στο αποτέλεσμα NULL τιμές σε περίπτωση που έχει παρακολουθήσει κάποια παράσταση αλλά δεν την έχει αξιολογήσει.

- **Q07 - Το φεστιβάλ με τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού**

  Αρχικά βρίσκουμε τον μέσο όρο εμπειρίας για κάθε φεστιβάλ με ένα subquery και αυτόν τον πίνακα, τον κάνουμε sort με το average_xp και παίρνουμε την πρώτη γραμμή. Για την έυρεση του μέσου όρου εμπειρίας αρκεί να βρούμε ποιοι δουλεύουν ως τεχνικό προσωπικό σε κάθε event του φεστιβάλ μέσω του view EventStaff. Η εντολή group by festival εδώ είναι κρίσιμης σημασίας καθώς έτσι αναγνωρίζει η MySQL που ακριβώς αντιστοιχεί η τιμή AVG(xp).

- **Q08 - Το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία**

  Αρχικά κάνουμε JOIN το Staff με το Role μέσω του Foreign Key(role_id) ώστε να μπορούμε να ψάχνουμε με το όνομα του ρόλου και όχι το id του στο WHERE clause. Έπειτα κάνουμε αυτό το αποτέλεσμα LEFT JOIN με ένα subquery που επιλέγει το το id του προσωπικού και του event μιας συγκεκριμένης ημερομηνίας από έναν πίνακα που προκύπτει από την συνένωση του Event_P με το Employment. Έτσι καταλήγουμε με το έναν πίνακα με το βοηθητικό προσωπικό αριστερά και δεξιά το employment_id αν εργάζονται εκείνη την ημερομηνία ή NULL αν δεν εργάζονται εκείνη την μερά. Επομένως, Με την συνθήκη αν το αριστερό μέρος είναι NULL, μπορούμε να βρούμε ποιοι δεν έχουν προγραμματισμένη εργασία.

- **Q09 - Επισκέπτες έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με περισσότερες από 3 παρακολουθήσεις**

  Χρησιμοποιούμε ένα CTE ως Event_counts που έχει ως πεδία το festival_id, το visitor_id και το πλήθος παραστάσεων που έχει δει ο κάθε επισκέπτης σε κάθε φεστιβάλ μόνο όταν έχουν παρακολουθήσει πάνω από τρεις παραστάσεις, το οποίο προκύπτει από την συνένωση του Ticket και του Event_P. Μετά κάνουμε self join για να μπορούμε να συγκρίνουμε αν υπάρχει για κάθε επισκέπτη και κάποιος άλλος ο οποίος έχει δει τον ίδιο αριθμό παραστάσεων.

- **Q10 - Τα 3 κορυφαία (top-3) ζεύγη που εμφανίστηκαν σε φεστιβάλ**

  Για να βρούμε τα ζεύγη ειδών μουσικής, κάνουμε self join τον πίνακα PerformerGenre με βάση το performer_id ο οποίος περιέχει τους performers και τα μουσικά είδη που παίζουν με κατάλληλη συνθήκη ώστε κάθε ζεύγος να εμφανίζεται μία μοναδική φορά. Από εκεί, παρουσιάζουμε τις πρώτες καταχωρήσεις με το μεγαλύτερο COUNT().

- **Q11 - Όλους τους καλλιτέχνες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον καλλιτέχνη με τις περισσότερες συμμετοχές σε φεστιβάλ**

  Για την εύρεση των καλλιτεχνών με την ερμηνεία Artists που έχουν 5 λιγότερες συμμετοχές από αυτόν με τις περισσότερες, φτάνει να συγκρίνουμε στο WHERE clause τα participations κάθε artist με ένα

subquery που επιστρέφει το max() των participations από τον πίνακα Artist μείον 5.

- **Q12 - Το προσωπικό που απαιτείται για κάθε ημέρα του φεστιβάλ, παρέχοντας ανάλυση ανά κατηγορία (τεχνικό προσωπικό ασφαλείας, βοηθητικό προσωπικό)**

  Ομαδοποιούμε τα entries του view EventStaff ανά φεστιβάλ και ημέρα. Μετράμε τον αριθμό των εργαζομένων ανά κατηγορία με εκφράσεις της μορφής:
  COUNT(CASE WHEN role = 2 THEN 1 END) AS Technician

- **Q13 - Εύρεση καλλιτεχνών που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους**

  Καλλιτέχνες θεωρούμε τους artists και σε αυτό το query.
  Συνενώνουμε το view PerformerPerformance και τα tables Event_P, Festival, Location, Continent. Ομαδοποιούμε τα αποτελέσματα με βάση το performer_id, ώστε να μπορέσουμε να μετρήσουμε τον αριθμό των διακριτών ηπείρων που έχει επισκεφτεί στα πλαίσια του φεστιβάλ. Επιλέγουμε μόνο τους καλλιτέχνες για τους οποίους μετρήσαμε πάνω από δύο ηπείρους και ταξινομούμε τα αποτελέσματα με φθίνοντα αριθμό ηπείρων.

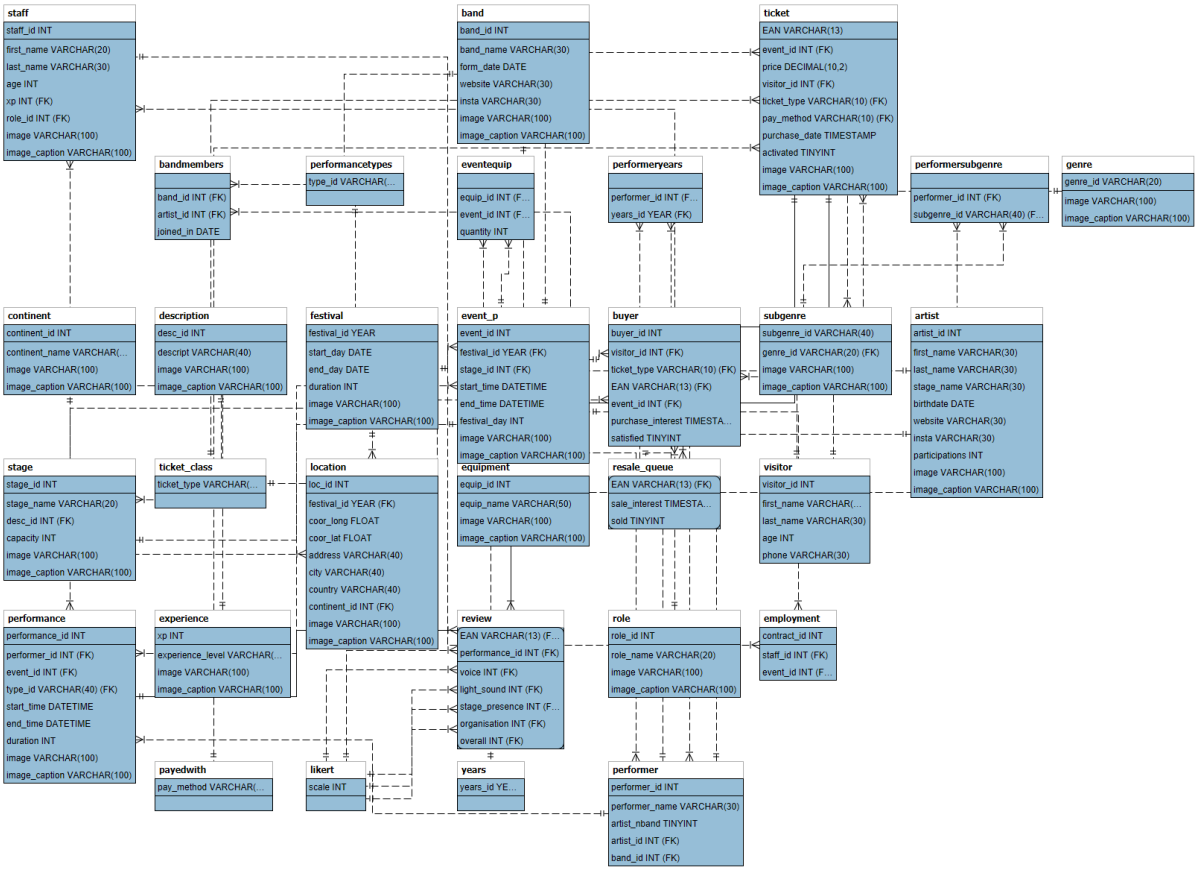
- **Q14 - Εύρεση μουσικών ειδών με ίσο και μεγαλύτερο του 2 αριθμό εμφανίσεων σε δύο συνεχόμενα έτη**
  Δημιουργούμε Common Table Expression fgp1, το οποίο περιέχει τον αριθμό των εμφανίσεων που αντιστοιχούν σε κάθε μουσικό είδος που εμφανίστηκε σε κάθε φεστιβάλ, μόνο σε περίπτωση που μετρήθηκαν πάνω από 3 εμφανίσεις.
  Για τη δημιουργία του fgp1, δημιουργούμε ένα προσωρινό table fgp, το οποίο συνδυάζει φεστιβάλ, εμφάνιση και μουσικό είδος, μετά από συνένωση των πινάκων Event και Performance με το view PerformerGenre.
  Εάν μια παράσταση ανήκει σε πολλά είδη, θα υπάρχει αντίστοιχο entry για κάθε είδος. Ο fgp1 προκύπτει από τον fgp, με ομαδοποίηση του περιεχομένου του με βάση το φεστιβάλ και το είδος, ώστε να μπορούν να μετρηθούν οι εμφανίσεις. Κρατάμε μόνο τα entries όπου ο αριθμός των εμφανίσεων είναι μεγαλύτερος του 2.
  Συνενώσαμε το fpg1 f1 με instance του εαυτό του, με όνομα f2, υπό τη συνθήκη ότι f2.festival_id = f1.festival_id +1, δηλαδή τα έτη είναι διαδοχικά και ο αριθμός εμφανίσεων για κάποιο είδος μουσικής είναι κοινός.

- **Q15 - Εύρεση των 5 ζευγων καλλιτέχνη-επισκέπτη, για τα οποία συναντάμε την υψηλότερη αθροιστική βαθμολόγηση, στα πεδία overall και voice.**

  Για το τελευταίο query, καλλιτέχνη θεωρούμε τον performer.
  Εκτελούμε JOIN στους πίνακες Review, Ticket, Visitor και στο view PerformerPerformance. Ομαδοποιούμε τα αποτελέσματα με βάση το visitor_id και το performer_id. Τα ταξινομούμε με φθίνον total_score, το οποίο για κάθε ομάδα υπολογίζεται ως SUM(voice+overall). Κρατάμε τα 5 πρώτα αποτελέσματα.

7. Δημιουργία σχεσιακού διαγράμματος

Το σχεσιακό διάγραμμα έγινε αυτόματα μετά την ολοκλήρωση της βάσης μέσω της εφαρμογής MySQL Workbench[2]. Παρατηρούμε πως χρειάζονται αισθητά περισσότερα tables για να καλύψουν τις ανάγκες των σχέσεων, καθώς και για λόγους κανονικοποίησης.



[2] MySQL Workbench

8. Ερωτήματα 4 και 6

1. Ερώτημα 4

```
SELECT performer_name, AVG(voice) as avg_voice, AVG(overall) as avg_overall
FROM REVIEW r
JOIN Performance p ON p.performance_id = r.performance_id
JOIN Performer per ON per.performer_id = p.performer_id
WHERE per.performer_name='Taylor Swift';
```

Η εκτέλεση του query με το αρχικό query plan δίνει τα παρακάτω αποτελέσματα:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	per	ref	PRIMARY,idx_performer_name	idx_performer_name	122	const	1	Using where; Using index
1	SIMPLE	p	ref	PRIMARY,performer_id	performer_id	4	pulse_music_festival.per.performer_id	1	Using index
1	SIMPLE	r	ref	performance_id	performance_id	4	pulse_music_festival.p.performance_id	1	

και μέσω του profiling του phpMyAdmin βλέπουμε τους παρακάτω χρόνους για τις διάφορες εργασίες που γίνονται κατά την εκτέλεση του query.

Βλέπουμε ότι η εκτέλεση του query γίνεται ως εξής:

State	Total Time	% Time	Calls	ø Time
Starting	189 µs	18.98%	1	189 µs
Updating Status	106 µs	10.64%	1	106 µs
Statistics	105 µs	10.54%	1	105 µs
Init	67 µs	6.73%	1	67 µs
Preparing	67 µs	6.73%	1	67 µs
Optimizing	64 µs	6.43%	1	64 µs
Opening Tables	58 µs	5.82%	1	58 µs
Executing	58 µs	5.82%	1	58 µs
Table Lock	41 µs	4.12%	1	41 µs
Checking Permissions	31 µs	3.11%	1	31 µs
Closing Tables	30 µs	3.01%	2	15 µs
Freeing Items	25 µs	2.51%	1	25 µs
Query End	23 µs	2.31%	1	23 µs
After Opening Tables	21 µs	2.11%	1	21 µs
Commit	19 µs	1.91%	1	19 µs
Reset For Next Command	18 µs	1.81%	1	18 µs
System Lock	17 µs	1.71%	1	17 µs
Unlocking Tables	16 µs	1.61%	1	16 µs
Starting Cleanup	15 µs	1.51%	1	15 µs

1. Γίνεται scan όλου του Review Table (61 rows) το οποίο φαίνεται από το type ALL
2. Γίνεται αναζήτηση μέσω του primary key με το type eq_ref το οποίο είναι το αποδοτικότερο join, καθώς μία γραμμή αντιστοιχίζεται σε κάθε γραμμή του πίνακα review χρησιμοποιώντας το primary key performance_id
3. Ομοίως με το 2 έχουμε αντιστοίχιση με το primary key το οποίο είναι ο γρηγορότερος τρόπος αναζήτησης.

Ως μέθοδο optimization δοκιμάζουμε την προσθήκη force index (idx_review_performance_id) στον πίνακα Review. Παρατηρούμε ότι δεν εφαρμόζεται από τον optimizer της MySQL επειδή το full scan table μάλλον είναι λιγότερο κοστοβόρο από την δημιουργία indexes.

Δοκιμάζουμε να κάνουμε force index (idx_performance_performer_id).

```
explain SELECT performer_name, AVG(voice) as avg_voice, AVG(overall) as avg_overall
FROM REVIEW r
JOIN Performance p force index (idx_performance_performer_id) ON p.performance_id = r.performance_id
JOIN Performer per ON per.performer_id = p.performer_id
WHERE per.performer_name='Taylor Swift';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	per	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	67	Using where
1	SIMPLE	p	ref	idx_performance_performer_id	idx_performance_performer_id	4	pulse_music_festival.per.performer_id	1	Using index
1	SIMPLE	r	ALL	idx_review_performance_id	*NULL*	*NULL*	*NULL*	8	Using where; Using join buffer (flat, BNL join)

Summary by state

State	Total Time	% Time	Calls	ø Time
Starting	99 µs	19.64%	1	99 µs
Checking Permissions	14 µs	2.78%	1	14 µs
Opening Tables	29 µs	5.75%	1	29 µs
After Opening Tables	9 µs	1.79%	1	9 µs
System Lock	8 µs	1.59%	1	8 µs
Table Lock	20 µs	3.97%	1	20 µs
Init	37 µs	7.34%	1	37 µs
Optimizing	33 µs	6.55%	1	33 µs
Statistics	53 µs	10.52%	1	53 µs
Preparing	34 µs	6.75%	1	34 µs
Executing	27 µs	5.36%	1	27 µs
Query End	11 µs	2.18%	1	11 µs
Commit	9 µs	1.79%	1	9 µs
Closing Tables	13 µs	2.58%	2	6.5 µs
Unlocking Tables	7 µs	1.39%	1	7 µs
Starting Cleanup	8 µs	1.59%	1	8 µs
Freeing Items	20 µs	3.97%	1	20 µs
Updating Status	53 µs	10.52%	1	53 µs
Reset For Next Command	8 µs	1.59%	1	8 µs

Βλέπουμε ότι βελτιώνεται ο χρόνος εκτέλεσης του query από 58 µs σε 27 µs , το starting time από 189 µs σε 99 µs και το updating status από 106 µs σε 53 µs.

Τέλος δημιουργούμε επίσης το index idx_performer_name το οποίο αποτρέπει το full table scan του table performer. Βλέπουμε ότι έχουμε την καλύτερη επίδοση από όλες τις περιπτώσεις.

```
explain SELECT performer_name, AVG(voice) as avg_voice, AVG(overall) as avg_overall
FROM Performer per FORCE INDEX (idx_performer_name)
JOIN Performance p FORCE INDEX (idx_performance_performer_id) ON per.performer_id = p.performer_id
JOIN REVIEW r FORCE INDEX (idx_review_performance_id) ON p.performance_id = r.performance_id
WHERE per.performer_name = 'Taylor Swift';
```

Summary by state

State	Total Time	% Time	Calls	ø Time
Starting	88 µs	20.28%	1	88 µs
Checking Permissions	10 µs	2.30%	1	10 µs
Opening Tables	27 µs	6.22%	1	27 µs
After Opening Tables	7 µs	1.61%	1	7 µs
System Lock	6 µs	1.38%	1	6 µs
Table Lock	15 µs	3.46%	1	15 µs
Init	30 µs	6.91%	1	30 µs
Optimizing	26 µs	5.99%	1	26 µs
Statistics	61 µs	14.06%	1	61 µs
Preparing	27 µs	6.22%	1	27 µs
Executing	21 µs	4.84%	1	21 µs

Παρατηρούμε ότι το executing time κατέβηκε στα 21 µs, το starting στα 88 µs και το updating status στα 52µs.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	per	ref	idx_performer_name	idx_performer_name	122	const	1	Using where; Using index
1	SIMPLE	p	ref	idx_performance_performer_id	idx_performance_performer_id	4	pulse_music_festival.per.performer_id	1	Using index
1	SIMPLE	r	ref	idx_review_performance_id	idx_review_performance_id	4	pulse_music_festival.p.performance_id	1	

Για να δοκιμάσουμε διαφορετικές στρατηγικές Join αρχικά απενεργοποιούμε την χρήση των indexes και παίρνουμε το παρακάτω query plan, το οποίο χρησιμοποιεί block nested loop join.

```
SELECT per.performer_name, AVG(voice) AS avg_voice, AVG(overall) AS avg_overall FROM REVIEW r IGNORE INDEX (performance_id) JOIN Performance p IGNORE INDEX (PRIMARY,performer_id) ON p.performance_id = r.performance_id JOIN Performer per IGNORE INDEX (PRIMARY,idx_performer_name) ON per.performer_id = p.performer_id WHERE per.performer_name = 'Taylor Swift';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	r	ALL	*NULL*	*NULL*	*NULL*	*NULL*	61	
1	SIMPLE	per	ALL	*NULL*	*NULL*	*NULL*	*NULL*	67	Using where; Using join buffer (flat, BNL join)
1	SIMPLE	p	ALL	*NULL*	*NULL*	*NULL*	*NULL*	156	Using where; Using join buffer (incremental, BNL join)

Starting	280 µs	18.65%	1	280 µs
Updating Status	165 µs	10.99%	1	165 µs
Statistics	144 µs	9.59%	1	144 µs
Preparing	126 µs	8.39%	1	126 µs
Optimizing	123 µs	8.19%	1	123 µs
Init	102 µs	6.80%	1	102 µs
Opening Tables	82 µs	5.46%	1	82 µs
Executing	76 µs	5.06%	1	76 µs
Table Lock	64 µs	4.26%	1	64 µs
Commit	41 µs	2.73%	1	41 µs
System Lock	38 µs	2.53%	1	38 µs
Checking Permissions	37 µs	2.47%	1	37 µs
Query End	33 µs	2.20%	1	33 µs
Freeing Items	32 µs	2.13%	1	32 µs
After Opening Tables	29 µs	1.93%	1	29 µs
Reset For Next Command	26 µs	1.73%	1	26 µs
Closing Tables	23 µs	1.53%	2	11.5 µs
Unlocking Tables	22 µs	1.47%	1	22 µs
Starting Cleanup	21 µs	1.40%	1	21 µs

Με τις εντολές

```
SET SESSION join_cache_level = 8; -- Enables hash join algorithm
SET SESSION optimizer_switch = 'join_cache_hashed=on';
```

προσπαθούμε να ενεργοποιήσουμε το hash join. Παίρνουμε το παρακάτω αποτέλεσμα, που βλέπουμε ότι χρησιμοποιείται το Batched Key Access Hash Join.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	r	ALL	*NULL*	*NULL*	*NULL*	*NULL*	61	
1	SIMPLE	per	ALL	*NULL*	*NULL*	*NULL*	*NULL*	67	Using where; Using join buffer (flat, BNL join)
1	SIMPLE	p	hash_ALL	*NULL*	#hash#$hj	8	pulse_music_festival.r.performance_id,pulse_music_festival.per.performer_id	156	Using where; Using join buffer (incremental, BNLH join)

State	Total Time	% Time	Calls	ø Time
Starting	268 µs	17.97%	1	268 µs
Updating Status	175 µs	11.74%	1	175 µs
Statistics	146 µs	9.79%	1	146 µs
Preparing	142 µs	9.52%	1	142 µs
Executing	105 µs	7.04%	1	105 µs
Init	99 µs	6.64%	1	99 µs
Optimizing	99 µs	6.64%	1	99 µs
Opening Tables	83 µs	5.57%	1	83 µs
Table Lock	58 µs	3.89%	1	58 µs
Checking Permissions	37 µs	2.48%	1	37 µs
Query End	35 µs	2.35%	1	35 µs
Freeing Items	34 µs	2.28%	1	34 µs
After Opening Tables	30 µs	2.01%	1	30 µs
Commit	26 µs	1.74%	1	26 µs
Reset For Next Command	26 µs	1.74%	1	26 µs
System Lock	24 µs	1.61%	1	24 µs
Closing Tables	23 µs	1.54%	2	11.5 µs
Starting Cleanup	23 µs	1.54%	1	23 µs
Unlocking Tables	22 µs	1.48%	1	22 µs

Παρατηρούμε ότι το hash join χειροτέρευσε τον χρόνο executing , καλυτέρευσε τον χρόνο optimizing όμως τα περισσότερα πεδία παρέμειναν ίδια.

Τέλος, για την υλοποίηση του merge join χρησιμοποιούμε τις παρακάτω εντολές

```
SET SESSION join_cache_level = 4; -- Enables merge join algorithm
SET SESSION optimizer_switch = 'join_cache_hashed=off';
SET SESSION optimizer_switch = 'use_index_extensions=off,
index_merge=off, index_merge_union=off, index_merge_sort_union=off,
index_merge_intersection=off';
```

2. Ερώτημα 6

```
SELECT t.event_id AS Event, (AVG(r.voice) + AVG(r.light_sound) +
AVG(r.stage_presence) + AVG(r.organisation) + AVG(r.overall))/5 AS Score FROM
Ticket t
LEFT JOIN Review r ON r.EAN = t.EAN
WHERE t.visitor_id = 1 AND t.activated = 1
GROUP BY t.event_id;
```

Η εκτέλεση του query με το αρχικό query plan δίνει τα παρακάτω αποτελέσματα:

State	Total Time	% Time	Calls	ø Time
Statistics	101 µs	16.32%	1	101 µs
Starting	96 µs	15.51%	1	96 µs
Init	67 µs	10.82%	1	67 µs
Updating Status	59 µs	9.53%	1	59 µs
Preparing	36 µs	5.82%	1	36 µs
Creating Tmp Table	30 µs	4.85%	1	30 µs
Table Lock	28 µs	4.52%	1	28 µs
Opening Tables	27 µs	4.36%	1	27 µs
Optimizing	26 µs	4.20%	1	26 µs
Executing	22 µs	3.55%	1	22 µs
Checking Permissions	14 µs	2.26%	1	14 µs
Sorting Result	13 µs	2.10%	1	13 µs
Freeing Items	13 µs	2.10%	1	13 µs
Reset For Next Command	10 µs	1.62%	1	10 µs
After Opening Tables	9 µs	1.45%	1	9 µs
Query End	9 µs	1.45%	2	4.5 µs
System Lock	8 µs	1.29%	1	8 µs
Commit	8 µs	1.29%	1	8 µs
Removing Tmp Table	7 µs	1.13%	1	7 µs
Closing Tables	7 µs	1.13%	2	3.5 µs
Unlocking Tables	6 µs	0.97%	1	6 µs
Starting Cleanup	6 µs	0.97%	1	6 µs

Κοιτώντας τους χρόνους για το opening tables, optimizing και executing βλέπουμε ότι το καθένα παίρνει γύρω στα 22-27µs.
Μέσω του EXPLAIN βλέπουμε ότι ο SQL Optimizer χρησιμοποιεί το index που εχουμε φτιάξει για την σύνδεση visitor με ticket.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ref	idx_ticket_visitor_id	idx_ticket_visitor_id	4	const	4	Using where; Using temporary; Using filesort
1	SIMPLE	r	ref	PRIMARY,idx_review_EAN	PRIMARY	54	pulse_music_festival.t.EAN	1	

Σαν πρώτη βελτίωση χρησιμοποιούμε τον idx_review_EAN και παρατηρούμε μια μικρή βελτίωση στον χρόνο εκτέλεσης κατα 4-7µs. Ακόμη, όμως, δεν έχουν βελτιωθεί οι καθυστερήσεις από τα using temporary και using filesort που γίνονται στο table Ticket. Αυτός ο προσωρινός πίνακας και η ανάγκη για filesort δημιουργούνται από την εντολή group by event_id. Έτσι προσθέτουμε έναν composite index με την εντολή

State	Total Time	% Time	Calls	ø Time
Starting	86 µs	19.59%	1	86 µs
Statistics	52 µs	11.85%	1	52 µs
Updating Status	41 µs	9.34%	1	41 µs
Init	35 µs	7.97%	1	35 µs
Preparing	25 µs	5.69%	1	25 µs
Creating Tmp Table	22 µs	5.01%	1	22 µs
Opening Tables	21 µs	4.78%	1	21 µs
Table Lock	21 µs	4.78%	1	21 µs
Optimizing	20 µs	4.56%	1	20 µs
Executing	16 µs	3.64%	1	16 µs
Checking Permissions	12 µs	2.73%	1	12 µs
Freeing Items	10 µs	2.28%	1	10 µs
Sorting Result	9 µs	2.05%	1	9 µs
Unlocking Tables	9 µs	2.05%	1	9 µs
After Opening Tables	7 µs	1.59%	1	7 µs
Reset For Next Command	7 µs	1.59%	1	7 µs
System Lock	6 µs	1.37%	1	6 µs
Query End	6 µs	1.37%	2	3 µs
Commit	6 µs	1.37%	1	6 µs
Removing Tmp Table	5 µs	1.14%	1	5 µs
Closing Tables	5 µs	1.14%	2	2.5 µs
Starting Cleanup	5 µs	1.14%	1	5 µs

CREATE INDEX idx_ticket_opt ON Ticket(visitor_id, activated, event_id);

και παίρνουμε τα εξής αποτελέσματα:

State	Total Time	% Time	Calls	ø Time
Starting	82 µs	14.67%	1	82 µs
Checking Permissions	10 µs	1.79%	1	10 µs
Opening Tables	19 µs	3.40%	2	9.5 µs
After Opening Tables	7 µs	1.25%	2	3.5 µs
System Lock	6 µs	1.07%	2	3 µs
Table Lock	7 µs	1.25%	2	3.5 µs
Closing Tables	6 µs	1.07%	4	1.5 µs
Unlocking Tables	5 µs	0.89%	2	2.5 µs
Init	35 µs	6.26%	1	35 µs
Optimizing	31 µs	5.55%	1	31 µs
Statistics	67 µs	11.99%	1	67 µs
Preparing	28 µs	5.01%	1	28 µs
Sorting Result	10 µs	1.79%	1	10 µs
Executing	17 µs	3.04%	1	17 µs
Query End	7 µs	1.25%	1	7 µs
Commit	6 µs	1.07%	1	6 µs
Starting Cleanup	5 µs	0.89%	1	5 µs
Freeing Items	36 µs	6.44%	1	36 µs
Updating Status	64 µs	11.45%	1	64 µs
Reset For Next Command	10 µs	1.79%	1	10 µs

Παρατηρούμε ότι παρ' ότι αυξήθηκε ο χρόνος της βελτιστοποίησης (optimizing), μηδενίστηκε ο χρόνος creating tmp table που έπαιρνε 22µs και ο χρόνος table lock από 21 µs σε 7 µs.

Το index δηλαδή μέσω των ορισμάτων visitor_id και activated μας επέτρεψε να κάνουμε γρήγορα το filtering του where και με το event_id το group by στο τέλος.

Παρατηρούμε ακόμη ότι μειώθηκαν οι γραμμές αναζήτησης από 5 σε 3.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ref	idx_ticket_visitor_id,idx_ticket_opt	idx_ticket_opt	5	const,const	2	Using where; Using index
1	SIMPLE	r	ref	idx_review_EAN	idx_review_EAN	54	pulse_music_festival.t.EAN	1	

Τώρα θα προσπαθήσουμε να φτιάξουμε το query plan με hash join, χρησιμοποιώντας τις ίδιες εντολές με παραπάνω και παίρνουμε τα παρακάτω αποτελέσματα.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ALL	*NULL*	*NULL*	*NULL*	*NULL*	414	Using where; Using temporary; Using filesort
1	SIMPLE	r	hash_ALL	*NULL*	#hash#$hj	54	pulse_music_festival.t.EAN	61	Using where; Using join buffer (flat, BNLH join)

State	Total Time	% Time	Calls	ø Time
Starting	155 µs	19.33%	1	155 µs
Updating Status	97 µs	12.09%	1	97 µs
Init	64 µs	7.98%	1	64 µs
Statistics	59 µs	7.36%	1	59 µs
Creating Tmp Table	45 µs	5.61%	1	45 µs
Preparing	40 µs	4.99%	1	40 µs
Opening Tables	38 µs	4.74%	1	38 µs
Optimizing	38 µs	4.74%	1	38 µs
Executing	33 µs	4.11%	1	33 µs
Table Lock	28 µs	3.49%	1	28 µs
Sorting Result	27 µs	3.37%	1	27 µs
Checking Permissions	19 µs	2.37%	1	19 µs
Freeing Items	19 µs	2.37%	1	19 µs
After Opening Tables	14 µs	1.75%	1	14 µs
Query End	14 µs	1.75%	2	7 µs
Removing Tmp Table	14 µs	1.75%	1	14 µs
Reset For Next Command	14 µs	1.75%	1	14 µs
Commit	13 µs	1.62%	1	13 µs
System Lock	12 µs	1.50%	1	12 µs
Closing Tables	11 µs	1.37%	2	5.5 µs
Unlocking Tables	10 µs	1.25%	1	10 µs
Starting Cleanup	10 µs	1.25%	1	10 µs

Παρατηρούμε ότι ο χρόνος εκτέλεσης διπλασιάστηκε και γίνεται scan όλων των γραμμών των δύο πινάκων.

Συμπεραίνουμε, λοιπόν, πως η δημιουργία των κατάλληλων indexes είναι κρίσιμη για το optimization των queries, καθώς επιτρέπει στον βελτιστοποιητή να επιλέγει πιο αποδοτικές στρατηγικές εκτέλεσης. Οι διαφορετικές στρατηγικές υλοποίησης πέρα από BNL, όπως οι merge joins και hash joins, μπορούν να προσφέρουν καλύτερη απόδοση κυρίως σε βάσεις δεδομένων με μεγάλα μεγέθη και σωστά indexed columns. Ωστόσο, η αποτελεσματικότητά τους εξαρτάται από τη δομή των δεδομένων, την ύπαρξη κατάλληλων indexes και τις εκτιμήσεις του optimizer. Στην περίπτωσή μας, η χρήση hash join δεν οδήγησε σε αισθητή βελτίωση, γεγονός που υποδεικνύει ότι για μικρού μεγέθους πίνακες ο optimizer ενδέχεται να προτιμήσει την απλούστερη στρατηγική BNL.