**Εθνικό Μετσόβιο Πολυτεχνείο**

***Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών***

Εξάμηνο 6ο : Βάσεις Δεδομένων

Ομάδα 15
Παπαϊωάννου Ελένη (03121041)
Σκαρμούτσος Σπυρίδων (03121644)

# Αναφορά Εξαμηνιαίας Εργασίας ΒΔ

## Σχολιασμός Βάσης Δεδομένων

Η Βάση Δεδομένων που υλοποιήσαμε σχετίζεται με ένα σύστημα αποθήκευσης και διαχείρισης πληροφοριών για την οργάνωση του διεθνούς festival μουσικής Pulse University . Κληθήκαμε να υλοποιήσουμε αυτή την βάση η οποία αποτελείται από τις εξής κύριες οντότητες :

1. Festival
   Αυτή είναι η κύρια οντότητα της βάσης μας και περιέχει πληροφορίες όπως την ημερομηνία διεξαγωγής , την τοποθεσία (μέσω του location entity ) ,την περιγραφή του.

2. Location
   Αυτή η οντότητα περιλαμβάνει πληροφορίες για την τοποθεσία διεξαγωγής των festival όπως διεύθυνση, συντεταγμένες , πόλη, χώρα και ήπειρο.

3. Stages
   Αυτή η οντότητα για το κάθε festival (fk festival_id) περιλαμβάνει περιγραφή , χωρητικότητα, εξοπλισμό καθώς και το πλήθος των εργαζομένων .

4. Staff
   Αυτή η οντότητα περιλαμβάνει το όνομα, την ηλικία, περιγραφή, τον ρόλο και το επίπεδο εμπειρίας του κάθε εργαζομένου . Με έναν βοηθητικό πίνακα ( StaffAssigment) συσχετίζουμε τις οντότητες 2 και 3.

5. StageStaff
   Ο συγκεκριμένος πίνακας αντιστοιχίζει εργαζόμενους σε σκηνές και περιλαμβάνει πληροφορίες όπως stage_id, staff_id (ως foreign keys) καθώς και την ημερομηνία ανάθεσης του αργαζομένου σε κάποια σκηνή.

6. Artists
   Αυτή η οντότητα αποτελείται από το όνομα , περιγραφή και διάφορες πληροφορίες για τον κάθε καλλιτέχνη (social media pages, εικόνες).

7. ArtistGenre
   Περιλαμβάνει τα είδη και τα υποείδη που αφορούν τον κάθε καλλιτέχνη ( fk artist_id).

8. Bands
   Το κάθε συγκρότημα αποτελείται από όνομα , ημερομηνία σχηματισμού , περιγραφή και πληροφορίες για social media, website.

9. BandMembers
   Αυτός ο βοηθητικός πίνακας αντιστοιχίζει τους artists στα bands που ανήκουν. (partial participation of artists.

10. Events
    Περιλαμβάνει όνομα, περιγραφή, το festival στο οποίο διεξάγεται (fk festival_id), την σκήνη στην οποία διεξάγεται ( fk stage_id).

11. Performances
    Περιλαμβάνουν τύπο performance (“Warm-up” , “Encore” etc) , ώρα έναρξης , διάρκεια, περιγραφή καθώς και το event στο οποίο διεξάγεται (fk event_id) τους καλλιτέχνες η συγκροτήματα που θα εμφανιστούν σε κάθε performance (fk band_id or artist_id).

12. Visitors
    Περιέχουν πληροφορίες όπως το όνομα και επώνυμο, την ηλικία και πληροφορίες επικοινωνίας για το κάθε επισκέπτη.

13. Tickets
    Αυτή η οντότητα περιγράφει το είδος του εισιτηρίου (General,VIP,Backstage), την ημέρα αγοράς, την τιμή, τον τρόπο πληρωμής, τον κώδικα ean13, τις μέρες για τις οποίες είναι σε ισχύ το εισιτήριο και το Boolean value για το αν χρησιμοποιείται το εισιτήριο ή όχι. Τέλος μέσω των fk event_id, visitor_id συσχετίζεται με τις οντότητες 9 και 11 αντίστοιχα.

14. Ratings
    Αυτή η οντότητα αξιολογεί κάθε performance (fk performance_id) από κάτοχο ενεργοποιημένου εισιτηρίου (fk ticket_id). Η αξιολόγηση γίνεται με βάση 5 κριτήρια: Ερμηνεία καλλιτεχνών, Ήχος και φωτισμός, Σκηνική παρουσία, Οργάνωση, Συνολική εντύπωση, και βαθμολογείται με βάση τη κλίμακα Likert (το “1” σημαίνει strongly disagree και το “5” strongly agree).

15. ResaleQueue
    Αυτός ο πίνακας λειτουργεί ως η βάση για την υλοποίηση της ουράς FIFO σε συνδυασμό με τα triggers που έχουμε ορίσει. Περιλαμβάνει το status πώλησης του κάθε εισιτηρίου στην ουρά, την ημερομηνία καταχώρησης του κάθε εισιτηρίου και την αντίστοιχη ημερομηνία πώλησης του αν υπάρχει. Συνδέεται με 3 οντότητες (Tickets, Visitors, Events) μέσω των 4 fk (ticket_id, seller_id, buyer_id και event_id), να σημειωθεί ότι οι sellers και buyers είναι και οι δύο οντότητες Visitors.

16. BuyRequests
    Αυτή η οντότητα λειτουργεί σε συνδυασμό με την ResaleQueue για την υλοποίηση της ουράς FIFO, καθώς χειρίζεται τα αιτήματα αγοράς από τους ενδιαφερόμενους επισκέπτες-buyers. Έχει πληροφορίες για την ημερομηνία δημιουργίας του εκάστοτε αιτήματος και το Boolean value εάν το αίτημα έχει ικανοποιηθεί. Τέλος χρησιμοποιεί τα fk buyer_id και event_id.

## Ευρετήρια

Τα ευρετήρια αυξάνουν την απόδοση της βάσης δεδομένων. Δημιουργήσαμε ευρετήρια για τα ερωτήματα 2.4 και 2.6 όπου κληθήκαμε να αναζητήσουμε στην βάση μας με index για την βελτιστοποίηση των query. Στα ίδια ερωτήματα χρησιμοποιήθηκαν και FORCE INDEXES. Στην πρώτη περίπτωση τα indexes χρησιμοποιούνται από την βάση οπότε κρίνεται απαραίτητο ενώ στην δεύτερη περίπτωση ορίζουμε πότε θα χρησιμοποιηθεί ποιο index. Τα ευρετήρια δημιουργήθηκαν με τον εξής τρόπο :

CREATE INDEX idx_rating_performance_id ON Ratings(performance_id); -- query 4

CREATE INDEX idx_performance_artist_id ON Performances(artist_id); -- query 4

CREATE INDEX idx_artist_id ON Artists(artist_id); -- query 4

CREATE INDEX visitor_id_idx ON Tickets(visitor_id); -- query 6

CREATE INDEX ticket_id_idx ON Ratings(ticket_id); -- query 6

## Queries 4 & 6

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

Υλοποιήσαμε το παραπάνω query με τη χρήση 4 διαφορετικών στρατηγικών για τα Join:

- Default Join: Ο optimizer επιλέγει τον τρόπο που θα γίνουν τα joins
- Nested Loop Join: Μέσω της εντολής STRAIGHT_JOIN υποχρεώνουμε το MySQL να κάνει τα joins με τη σειρά που δηλώνονται.

- Hash Join (BNL): Προσομοιώνουμε το hash join με την αγνόηση indexes, δηλαδή με την χρήση της εντολής IGNORE INDEX. Αν δεν υπάρχουν διαθέσιμα indexes, ή αγνοηθούν με αυτόν τον τρόπο, η βάση δεδομένων πιθανότατα θα χρησιμοποιήσει **Block Nested Loop Join**, που λειτουργεί παρόμοια με Hash Join, έτσι φορτώνονται "μπλοκ" δεδομένων στη μνήμη (hash buffer) και γίνεται σύγκριση χωρίς ευρετήρια.
- Merge Join: Χρησιμοποιώντας FORCE INDEX ενημερώνουμε τον MySQL optimizer ότι θέλουμε να βασιστεί σε συγκεκριμένα indexes και ταξινομούμε τους πίνακες. Κατά αυτό το τρόπο ο optimizer εκτελεί τα joins με συγχώνευση.

**Explain:**

Default

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | a | const | PRIMARY,idx_artist_id | PRIMARY | 4 | const | 1 | |
| 1 | SIMPLE | r | ALL | idx_rating_performance_id | NULL | NULL | NULL | 1 | |
| 1 | SIMPLE | p | eq_ref | PRIMARY,idx_performance_artist_id | PRIMARY | 4 | pulseuniversityfestival.r.performance_id | 1 | Using where |

Nested Loop Join

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | a | const | PRIMARY,idx_artist_id | PRIMARY | 4 | const | 1 | |
| 1 | SIMPLE | p | ALL | PRIMARY | NULL | NULL | NULL | 105 | Using where |
| 1 | SIMPLE | r | ALL | NULL | NULL | NULL | NULL | 1 | Using where; Using join buffer (flat, BNL join) |

Hash Join

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | a | const | PRIMARY,idx_artist_id | PRIMARY | 4 | const | 1 | |
| 1 | SIMPLE | r | ALL | NULL | NULL | NULL | NULL | 1 | |
| 1 | SIMPLE | p | ALL | NULL | NULL | NULL | NULL | 105 | Using where; Using join buffer (flat, BNL join) |

Merge Join

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | a | const | PRIMARY | PRIMARY | 4 | const | 1 | |
| 1 | SIMPLE | r | ALL | idx_rating_performance_id | NULL | NULL | NULL | 1 | |
| 1 | SIMPLE | p | eq_ref | idx_performance_artist_id | idx_performance_artist_id | 9 | const,pulseuniversityfestival.r.performance_id | 1 | Using index |

**Execution time comparison between the join strategies:**

| Join Strategy | Execution Time (ms) |
|---|---|
| MERGE JOIN | 25.048000000000000000000000000000000000... |
| DEFAULT JOIN | 26.378000000000000000000000000000000000... |
| HASH JOIN (BNL) | 26.397000000000000000000000000000000000... |
| NESTED LOOP JOIN | 28.682000000000000000000000000000000000... |

**Συμπεράσματα:**

- Παρατηρούμε ότι η στρατηγική των merge Join έχει την καλύτερη απόδοση, καθώς εκμεταλλεύεται τα indexes: idx_performance_artist_id, idx_rating_performance_id, τα joins είναι eq_ref (δηλαδή υπάρχει μοναδική αντιστοιχία ανά γραμμή), και στη στήλη

extra που αναφέρει using index, δείχνει ότι επιτυγχάνεται ακόμη καλύτερη απόδοση γιατί διαβάζει μόνο από index χωρίς να διαβάζει ολόκληρο τον πίνακα.

- Στη συνέχεια βλέπουμε ότι ως προς τον χρόνο εκτέλεσης ακολουθεί η default στρατηγική join που ακολουθεί ο optimizer, όπου δεν χρησιμοποίησε πλήρως τα διαθέσιμα indexes στο Ratings και εκτελεί ALL scan στο πίνακα Ratings.
- Το hash join αποτελεί μια ελαφρώς πιο αργή επιλογή, καθώς για τη δική μας βάση δεδομένων από ότι φαίνεται η χρήση προσωρινής μνήμη RAM για την προσομοίωση Block Nested Loop αντί της χρήσης indexes δίνει μία μέτρια απόδοση.
- Τέλος, το nested loop join φαίνεται να είναι το πιο αργό, πρόκειται για τη κλασική στρατηγική που σαρώνει τον 1ο πίνακα και για κάθε row αναζητά στον 2ο, που για πολλά δεδομένα λόγω των πολλών επαναλήψεων έχει χειρότερη απόδοση.

Επομένως, το **Merge Join** αποδείχτηκε το πιο αποδοτικό σε χρόνο και χρήση index, και είναι η **προτεινόμενη στρατηγική** για αυτό το query.

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

Υλοποιήσαμε το παραπάνω query με τη χρήση 4 διαφορετικών στρατηγικών για τα Join όπως και προηγουμένως:

- Default Join
- Nested Loop Join
- Hash Join (BNL)
- Merge Join

**Explain:**

Default

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | t | ref | PRIMARY,visitor_id_jdx | visitor_id_jdx | 4 | const | 3 | Using index; Using temporary; Using filesort |
| 1 | SIMPLE | r | ref | ticket_id,idx_rating_performance_id,ticket_id_idx | ticket_id | 4 | pulseuniversityfestival.t.ticket_id | 1 | |
| 1 | SIMPLE | p | eq_ref | PRIMARY | PRIMARY | 4 | pulseuniversityfestival.r.performance_id | 1 | |

Nested Loop Join

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | t | ALL | PRIMARY | NULL | NULL | NULL | 206 | Using where; Using temporary; Using filesort |
| 1 | SIMPLE | r | ALL | NULL | NULL | NULL | NULL | 78 | Using where; Using join buffer (flat, BNL join) |
| 1 | SIMPLE | p | eq_ref | PRIMARY | PRIMARY | 4 | pulseuniversityfestival.r.performance_id | 1 | |

Hash Join

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | r | ALL | ticket_id,idx_rating_performance_id | NULL | NULL | NULL | 78 | Using temporary; Using filesort |
| 1 | SIMPLE | t | eq_ref | PRIMARY,visitor_id_idx | PRIMARY | 4 | pulseuniversityfestival.r.ticket_id | 1 | Using where |
| 1 | SIMPLE | p | ALL | NULL | NULL | NULL | NULL | 105 | Using where; Using join buffer (flat, BNL join) |

Merge Join

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | t | ref | visitor_id_idx | visitor_id_idx | 4 | const | 3 | Using index; Using temporary; Using filesort |
| 1 | SIMPLE | r | ref | ticket_id_idx | ticket_id_idx | 4 | pulseuniversityfestival.t.ticket_id | 1 | |
| 1 | SIMPLE | p | eq_ref | PRIMARY | PRIMARY | 4 | pulseuniversityfestival.r.performance_id | 1 | |

**Execution time comparison between the join strategies:**

| Strategy | ExecutionTimeMs |
|---|---|
| Default Join | 32.615000000000000000000000000000000000... |
| Nested Loop Join | 36.156000000000000000000000000000000000... |
| Hash Join (BNL) | 34.200000000000000000000000000000000000... |
| Merge Join | 31.619000000000000000000000000000000000... |

**Συμπεράσματα:**

- Παρατηρούμε ότι η στρατηγική των merge Join έχει την καλύτερη απόδοση, καθώς αξιοποιεί σωστά τα index σε Tickets, Ratings και Performances, τα joins είναι ref/eq_ref (δηλαδή αποδοτικές μορφές join σε foreign keys, primary keys,unique not null ), και στη στήλη extra που αναφέρει using index, δείχνει ότι επιτυγχάνεται ακόμη καλύτερη απόδοση γιατί διαβάζει μόνο από index χωρίς να διαβάζει ολόκληρο τον πίνακα.
- Στη συνέχεια βλέπουμε ότι ως προς τον χρόνο εκτέλεσης πολύ κοντά στη στρατηγική merge join βρίσκεται και η default στρατηγική join που ακολουθεί ο optimizer, ο οποίος χρησιμοποιεί κάποια indexes αλλά περιλαμβάνει Using temporary; Using filesort, που δείχνει πιθανή χρήση buffer.
- Το hash join αποτελεί μια ελαφρώς πιο αργή επιλογή, καθώς χρησιμοποιεί join buffer και αγνοεί τα index, και γενικότερα είναι καλή στρατηγική όταν δεν υπάρχουν index ,όχι όμως η πιο αποδοτική αν υπάρχουν.
- Τέλος, το nested loop join φαίνεται να είναι η πιο αργή στρατηγική, αφού στους περισσότερους πίνακες περιλαμβάνει πλήρεις(ALL) σαρώσεις, και παράλληλα χρησιμοποιεί join buffer, γεγονός που δεν βοηθάει ιδιαιτέρως καθώς μεγαλώνει το κόστος όταν αυξάνονται τα δεδομένα.

Επομένως, το **Merge Join** αποδείχτηκε το πιο αποδοτικό σε χρόνο και χρήση index, και είναι η **προτεινόμενη στρατηγική** και για αυτό το query.

## Triggers

Μέσω των triggers ορίσαμε τους περιορισμούς της υλοποίησης της ΒΔ.

1. enforce_staff_requirements_before_insert
2. enforce_staff_requirements_before_delete
3. enforce_performance_schedule
4. prevent_artist_stage_conflict
5. prevent_artist_too_many_years
6. prevent_duplicate_tickets
7. enforce_stage_capacity
8. enforce_vip_ticket_limit
9. prevent_reusing_ticket
10. auto_match_resale
11. auto_match_buy_request
12. prevent_used_ticket_resale
13. remove_unfulfilled_request_after_ticket_purchase
14. prevent_rating_without_used_ticket
15. prevent_duplicate_ratings
16. prevent_rating_other_events

> Τα περισσότερα triggers έχουν προφανή ρόλο , ωστόσο σημειώνουμε ότι το auto_match_resale και το auto_match_buy_request είναι triggers που υλοποιούν την FIFO και αυτοματοποιούν τις διαδικασίες αντιστοίχισης μεταξύ αιτημάτων αγοράς και πώλησης.

## DDL SCRIPT

Το DDL Script είναι γραμμένο σε SQL και περιέχει τα tables, indexes και triggers της βάσης δεδομένων μας.

## DML SCRIPT

Το DML Script κάνει populate την βάση μας με τα δεδομένα. Εδώ έχουμε κάνει την παραδοχή ότι δεν υπάρχουν περιττά δεδομένα πχ όλοι οι artists θα παίξουν σε κάποιο event.

## Σχολιασμός ER

Σε αυτό μέρος της αναφοράς θα εξηγηθούν λίγο παραπάνω οι πιο περίπλοκες συνδέσεις του ER διαγράμματος.

- Events - Performances ( 1 - M ): Κάθε Event έχει πολλά performances και έχουμε total participation και των 2 entities.
- Events – Tickets ( 1 – M ): Κάθε εισιτήριο δίνει access σε 1 event αλλά πολλά διαφορετικά εισιτήρια μπορούν να δώσουν access στο ίδιο event.
- Tickets – Visitors ( M – 1 ): Ένας επισκέπτης μπορεί να αγοράσει εισιτήρια για πολλά event.
- Tickets-ResaleQueue-Events ( M – 1 ): Πολλά εισιτήρια μπορούν να διατεθούν στην ουρά για διάφορα events. Έχουμε partial participation των tickets καθώς δεν διατίθενται όλα στην ουρά μεταπώλησης.
- Visitors – BuyRequets – Tickets – Events ( 1 – M ): Ένας επισκέπτης μπορεί να κάνει διαφορετικά αιτήματα για διαφορετικά εισιτήρια για τα εκάστοτε events.
- Performance – Bands, Performance – Artists (M – 1, M – 1): Κάθε performance περιλαμβάνει είτε μία μπάντα είτε έναν καλλιτέχνη, και κάθε καλλιτέχνης ή μπάντα μπορεί να παίξει σε διάφορα performances (π.χ performances σε διαφορετικές χρονιές).

## Σχόλια

Για την παραγωγή των πληροφορίων της βάσης καθώς και την διόρθωση κάποιων σφαλμάτων του κώδικα χρησιμοποιήθηκαν LLMs, όπως Chat-GPT και Claude.

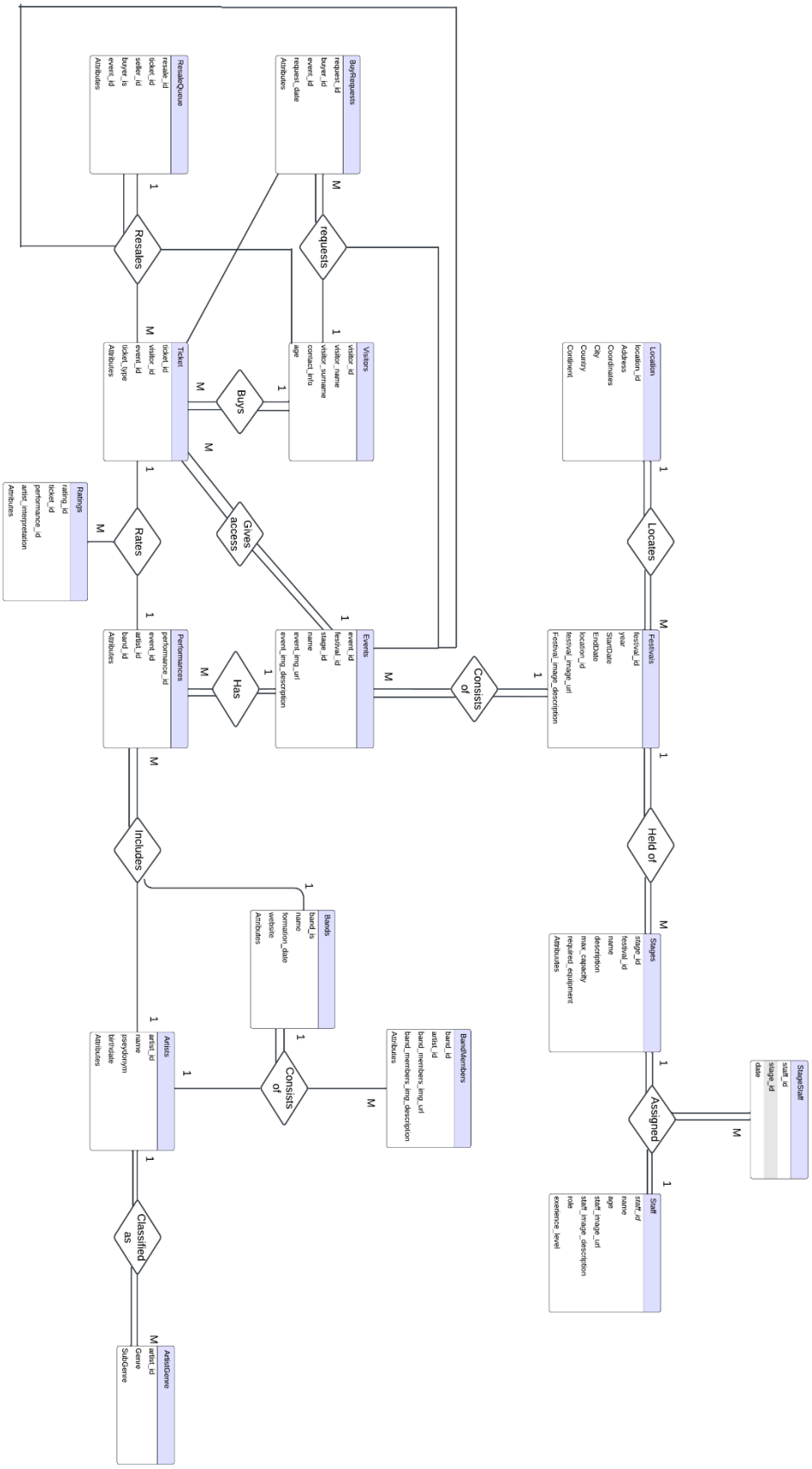
Κατασκευάστηκε ένας πίνακας Trigger Log (με prompt από το Chat-GPT) ώστε να μπορούμε να τεστάρουμε τα 2 triggers που αφορούν το matching του ResaleQueue και του BuyRequets ώστε να λειτουργεί αποτελεσματικά και ορθά η FIFO.

Έχουν προστεθεί 2 αρχεία. Αν εκτελεστούν σειριακά τα SQL statements αποτελούν έλεγχο για τα 2 σκέλη της FIFO (BuyRequestsTest.sql και ResaleQueueTest.sql).

Στο query Q09 έγινε η παραδοχή πως οι παρακολουθήσεις είναι περισσότερες ή ίσες του 3. Προφανώς, το query με την αλλαγή του COUNT(*) > 2 σε COUNT(*) > 3 θα ήταν ορθό αλλά δεν θα παρουσίαζε καθόλου δεδομένα εξόδου λόγω των δεδομένων της βάσης μας. Γι’ αυτό το λόγο προτιμήσαμε να κάνουμε την παραδοχή του “COUNT(*) > 2”, ώστε να φαίνεται ορθά η λειτουργικότητα του συγκεκριμένου query.

Τέλος, για τα query 4 και 6 έχουμε αποθηκεύσει τα outputs των πειραματισμών με τις διαφορετικές στρατηγικές (προφανώς ως πανομοιότυπα στη λογική query βγάζουν το ίδιο αποτέλεσμα) και τα αποτελέσματα σύγκρισης των χρόνων εκτέλεσης. Τα αποτελέσματα των EXPLAIN βρίσκονται ως screenshot στην αναφορά.

# ER Diagram


Location
location_id
Address
Coordinates
City
Country
Continent
Locates
Festivals
festival_id
year
StartDate
EndDate
location_id
festival_image_url
Festival_image_description
Held of
Stages
stage_id
festival_id
name
description
max_capacity
required_equipment
Attributes
Assigned
Staff
staff_id
name
age
staff_image_url
staff_image_description
role
experience_level
StageStaff
staff_id
stage_id
date
Consists of
Events
event_id
festival_id
stage_id
name
event_img_url
event_img_description
Has
Performances
performance_id
event_id
artist_id
band_id
Attributes
Includes
Bands
band_id
name
formation_date
website
Attributes
Consists of
BandMembers
band_id
artist_id
band_members_img_url
band_members_img_description
Attributes
Artists
artist_id
name
pseydonym
birthdate
Attributes
Classified as
ArtistGenre
artist_id
Genre
SubGenre
requests
BuyRequests
request_id
buyer_id
event_id
request_date
Attributes
Visitors
visitor_id
visitor_name
visitor_surname
contact_info
age
Buys
Gives access
Ticket
ticket_id
visitor_id
event_id
ticket_type
Attributes
Resales
ResaleQueue
resale_id
ticket_id
seller_id
buyer_is
event_id
Attributes
Rates
Ratings
rating_id
ticket_id
performance_id
artist_interpretation
Attributes

# Relational Schema