Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Εξάμηνο 6ο: Βάσεις Δεδομένων
Διδάσκοντες: Δ. Τσουμάκος, Μ. Κόνιαρης

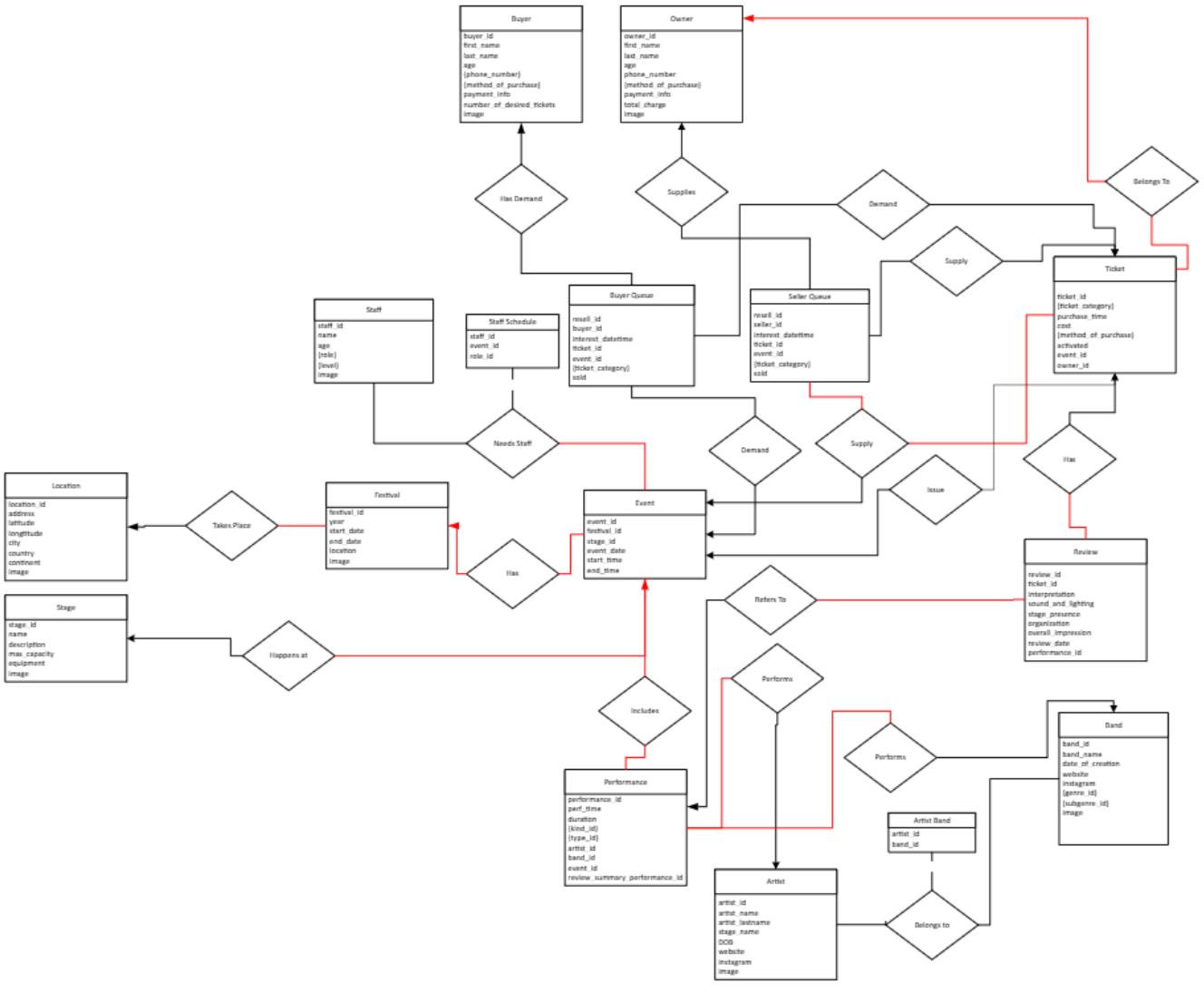
Βάσεις Δεδομένων
Εξαμηνιαία Εργασία

Συγγραφείς	1. Πλιάτσικα Μαγδαληνή-Μαρία (AM:03121220) 2. Σάββα Σοφία (AM:03121189) 3. Πανταζόπουλος Νικόλαος (AM:03121844)
Περιεχόμενα	1. Διάγραμμα Οντοτήτων-Συσχετίσεων (ER) 2. Σχεσιακό Σχήμα (Relational) 3. Ευρετήρια 4. DDL και DML 5. Queries 6. Authentication 7. Οδηγίες Εγκατάστασης

1. Διάγραμμα Οντοτήτων-Συσχετίσεων

Ζητούμενο της εργασίας είναι η σχεδίαση και υλοποίηση βάσης δεδομένων η οποία θα χρησιμοποιηθεί από το διεθνές φεστιβάλ μουσικής Pulse University για την διοργάνωση μελλοντικών φεστιβάλ και την εξόρυξη δεδομένων για τα προηγούμενα.

Ύστερα από προσεκτική ανάγνωση της δοθείσας εκφώνησης έγινε συλλογή των απαραίτητων πληροφοριών και κατανόηση των απαιτήσεων του φεστιβάλ σύμφωνα με τις οποίες σχεδιάστηκε το παρακάτω διάγραμμα ER. Ιδιαίτερη σημασία δόθηκε στις κύριες οντότητες που παρουσιάστηκαν γύρω από τις οποίες σχεδιάστηκαν συμπληρωματικοί πίνακες για την ευκολότερη πρόσβαση στις πληροφορίες τους. Οι οντότητες αυτές είναι το φεστιβάλ, η παράσταση, η εμφάνιση,το εισιτήριο κ.α. Παρακάτω φαίνεται μια αφαιρετική αλλά πλήρης περιγραφή της δομής που αποσκοπούμε να χειριστούμε, σε μορφή διαγράμματος οντοτήτων.



Σημειώνεται ότι οι κόκκινες γραμμές αντιπροσωπεύουν τη διπλή γραμμή του total participation.

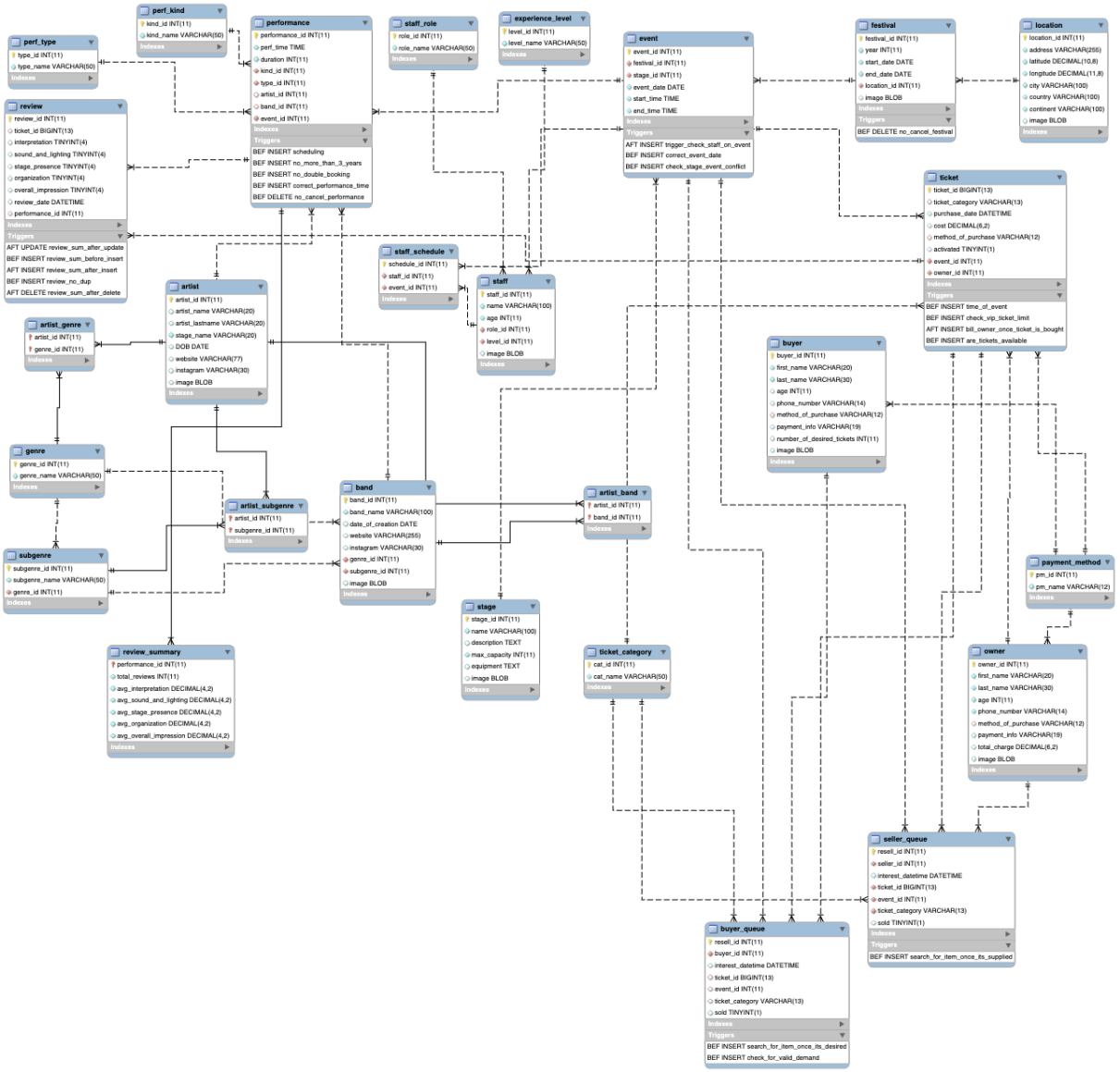
Μια σύντομη περιγραφή των σχέσεων και αλληλοεπιδράσεων που παρατηρούνται από το ER.

Το φεστιβάλ λαμβάνει χώρα κάθε χρόνο και αποτελείται από διαφορετικά events στα οποία εμφανίζονται warm-up, headline και surprise guest acts. Το κάθε event διαδραματίζεται σε ένα stage το οποίο στελεχώνεται με το κατάλληλο προσωπικό και αποτελείται από performances solo καλλιτεχνών και bands. Τα εισιτήρια πωλούνται ανά παράσταση και ανήκουν σε διαφορετικές κατηγορίες ενώ όταν ξεπουληθούν ενεργοποιείται η ουρά μεταπώλησης για την παράσταση που αφορούν όπου οι αγοραστές μπορούν να γίνουν οι νέοι κάτοχοί τους. Τέλος, οι επισκέπτες μπορούν να αξιολογήσουν σύμφωνα με διάφορα κριτήρια τα events που παρακολούθησαν.

Το παραπάνω ER διάγραμμα έχει φτιαχτεί έτσι ώστε να απεικονίζει με όσο το δυνατόν πιο ακριβή και ταυτόχρονα απλό τρόπο τις βασικές λειτουργίες που καλούμαστε να υλοποιήσουμε. Είναι εμφανές ότι σύνθετες σχέσεις δεν απεικονίζονται καθώς δεν αφορούν τον χρήστη της βάσης αλλά τον δημιουργό και διαχειριστή της. Οι άκρως απαραίτητες για την κατανόηση σχέσεις φαίνονται στα relations ενώ οι υπόλοιπες έχουν αποκρυφτεί από το χρήστη και σε πολλές περιπτώσεις (payment_method, ticket category etc) έχουν συμπεριληφθεί στα attributes.

2. Σχεσιακό Μοντέλο

Ζητείται η υλοποίηση του σχεσιακού διαγράμματος που προκύπτει από το E-R μοντέλο. Για την δημιουργία του χρησιμοποιήθηκαν οι μέθοδοι που διδάχτηκαν στο μάθημα καθώς και οι υποδείξεις που δίνονται από την εκφώνηση.
Ακόμη, χρησιμοποιήθηκαν περιορισμοί που εξασφαλίζουν την ορθή λειτουργία της βάσης τόσο στα πεδία τιμών (π.χ. το AEN έχει υποχρεωτικά 13 ψηφία) όσο και στην μορφή triggers.
Το σχεσιακό μοντέλο που προέκυψε είναι το ακόλουθο:



Οι περιορισμοί που σχετίζονται με την ελάχιστη και μέγιστη πληθικότητα των οντοτήτων υλοποιήθηκαν σε επίπεδο εισαγωγής δεδομένων, δηλαδή με triggers, events, stored procedures.

3. Ευρετήρια

Τα ευρετήρια (indexes) δημιουργούνται αυτόματα λόγω των primary και foreign keys. Πιο συγκεκριμένα δημιουργούνται τα εξής:

1. artist.PRIMARY, artist.stage_name δηλαδή το index για τους καλλιτέχνες δημιουργήθηκε αυτόματα λόγω του κλειδιού artist_id. Ακόμη, υπάρχει ευρετήριο στη στήλη stage_name του πίνακα artist για ευκολότερη αναζήτηση με βάση το καλλιτεχνικό ψευδώνυμο του.
2. artist_band.PRIMARY, artist_band.artist_id δηλαδή δημιουργούνται 2 indexes για το entity artist_band (ένα με το primary key (artist_id, band_id) και ένα μόνο με το artist_id για να διευκολύνεται η αναζήτηση με βάση το όνομα του καλλιτέχνη).
3. artist_genre.PRIMARY, artist_genre.genre_id (και αντίστοιχα artist_subgenre.PRIMARY, artist_subgenre.subgenre_id) για αναζήτηση με βάση το primary key (artist_id, genre_id) (ή (artist_id, subgenre_id)) αλλά και με βάση το είδος/ υποείδος.
4. band.PRIMARY, band.band_name, band.genre_id, band.subgenre_id δηλαδή υπάρχει δυνατότητα αναζήτησης βάσει του band_id (primary key), band_name (καλλιτεχνικό ψευδώνυμο του συγκροτήματος) το είδος και το υποείδος.
5. buyer.PRIMARY, buyer.method_of_purchase δηλαδή πέρα από ευρετήριο για το primary key buyer_id δημιουργείται και ένα βάσει του τρόπου πληρωμής.
6. buyer_queue.PRIMARY, buyer_queue.buyer_id, buyer_queue.event_id, buyer_queue.ticket_category, buyer_queue.ticket_id δηλαδή για την buyer_queue υπάρχουν indexes για τα εξής πεδία: στο resell_id (primary key) για ταυτοποίηση κάθε εγγραφής, buyer_id για εύρεση των αιτημάτων ενός συγκεκριμένου αγοραστή, event_id και ticket_category για γρήγορη αναζήτηση βάσει εκδήλωσης και κατηγορίας εισιτηρίου, ticket_id για εντοπισμό συγκεκριμένων εισιτηρίων.
7. event.PRIMARY, event.festival_id, event.stage_id δηλαδή δημιουργούνται 3 indexes για την παράσταση (ένα με το primary key event_id και δύο με foreign keys τα festival_id και stage_id αντίστοιχα).
8. experience_level.PRIMARY, experience_level.level_name δηλαδή δημιουργούνται ευρετήρια στο primary key level_id αλλά και στο level_name για ευκολότερη αναζήτηση βάσει ονόματος.
9. festival.PRIMARY, festival.location_id.
10. genre.PRIMARY, genre.genre_name και subgenre.PRIMARY, subgenre.subgenre_name.
11. location.PRIMARY.
12. owner.PRIMARY, owner.critical_info, owner.method_of_purchase.

13. payment_method.PRIMARY, payment_method.pm_name.
14. performance.PRIMARY, performance.artist_id, performance.band_id, performance.event_id, performance.kind_id, performance.type_id.
15. perf_kind.PRIMARY, perf_kind.kind_name.
16. perf_type.PRIMARY, perf_type.type_name.
17. review.PRIMARY.
18. review_summary.PRIMARY.
19. seller_queue.PRIMARY, seller_queue.event_id, seller_queue.seller_id, seller_queue.ticket_category, seller_queue.ticket_id.
20. staff.PRIMARY, staff.role_id, staff.level_id.
21. staff_role.PRIMARY, staff_role.role_name.
22. staff_schedule.PRIMARY, staff_schedule.staff_id, staff_schedule.event_id.
23. stage.PRIMARY.
24. ticket.PRIMARY, ticket.event_id, ticket.method_of_purchase, ticket.no_double_tickets_per_event, ticket.ticket_category.
25. ticket_category.PRIMARY, ticket_category.cat_name.

Από εμάς δημιουργήθηκε ακόμη το index performance.perf_time για ευκολότερη αναζήτηση βάσει της ώρας του performance.

4. DDL και DML

Έχοντας έτοιμο το διάγραμμα οντοτήτων , αξιοποιήσαμε το εργαλείο MySQL Workbench για την ανάπτυξη του κώδικα της εφαρμογής σε MySQL.

Κάποια σημαντικά σημεία της υλοποίησής μας είναι τα παρακάτω:
Στην υλοποίηση μας δεν υπάρχει resell_queue αλλά seller και buyer queue που αντιπροσωπεύουν την προσφορά και τη ζήτηση. Ο ίδιος ο seller δεν είναι οντότητα από μόνος του αλλά ανήκει στους owners μέχρι να πωληθεί το εισιτήριο που έχει διαθέσει προς πώληση. Αντίστοιχα, ο owner θεωρείται visitor όταν ενεργοποιηθεί το εισιτήριό του.

Εκτός από τους ορισμούς των tables και των αντίστοιχων κλειδιών και ευρετηρίων τους, που θα περιγράψουμε πολύ αναλυτικά στην συνέχεια, δημιουργήσαμε τα διάφορα triggers που ελέγχουν τους περιορισμούς των δεδομένων της βάσης ή υπολογίζουν δυναμικά άλλα δεδομένα.

1. Επιβάλλουμε στο φεστιβάλ ότι κάθε σκηνή μπορεί να φιλοξενεί μια παράσταση την ίδια στιγμή με το trigger check_stage_event_conflict.

2. Ελέγχουμε το προσωπικό ανά παράσταση και το αναθέτουμε σε σκηνές για την ομαλή διεξαγωγή του φεστιβάλ με τα procedures assign_security_to_event, assign_support_to_event και assign_tech_to_event. Σημειώνεται, ότι οι συναρτήσεις αυτές καλούνται εντός του trigger check_staff_on_event και αναθέτουν εξ ολοκλήρου ή συμπληρώνουν (εάν δεν πληρούνται οι ελάχιστες προϋποθέσεις) το προσωπικό που απαιτείται. Στην υλοποίηση της βάσης μας για έλεγχο της ορθότητας των συναρτήσεων αυτών δεν εισάγονται δεδομένα στο staff_schedule αλλά αφήνονται να τοποθετηθούν αυτόματα μέσω του trigger. Για λόγους πληρότητας έχουν υλοποιηθεί οι κατάλληλες συναρτήσεις που τοποθετούν δεδομένα και στο staff_schedule.
3. Καθορίζουμε την σειρά των εμφανίσεων σε μια παράσταση, τα διαλείμματα και τη διάρκεια τους στο trigger scheduling.
4. Αποτρέπουμε τα φεστιβάλ και τις εμφανίσεις από το να ακυρωθούν στα triggers no_cancel_festival και no_cancel_performance.
5. Ελέγχουμε ότι υπάρχουν σωστές ημερομηνίες και timestamps στα events και performances με τα triggers correct_event_date και correct_performace_time.
6. Ελέγχουμε ότι ένας καλλιτέχνης (συγκρότημα) δεν μπορεί να εμφανίζεται ταυτόχρονα σε δύο σκηνές με το trigger no_double_booking και ότι ένας καλλιτέχνης δεν επιτρέπεται να συμμετέχει πάνω από 3 συνεχή έτη στο no_more_than_3_years.
7. Όσον αφορά τα εισιτήρια ελέγχουμε αν η ημερομηνία στο εισιτήριο είναι ορθή με το trigger time_of_event, αν το event είναι sold out με το are_tickets_available και χρεώνουμε τους επισκέπτες κατά την αγορά τους με το trigger bill_owner_once_ticket_is_bought.
8. Ελέγχουμε ότι τα VIP εισιτήρια δεν ξεπερνούν το 10% της μέγιστης χωρητικότητας της σκηνής με το trigger check_vip_ticket_limit προτού εισαχθεί ένα νέο εισιτήριο.
9. Ελέγχουμε τους περιορισμούς που σχετίζονται με την ζήτηση στην ουρά μεταπώλησης (π.χ. ότι δεν ενεργοποιείται πριν την εξάντληση των εισιτηρίων) με το trigger check_for_valid_demand.
10. Στην περίπτωση που ζητείται συγκεκριμένα ένα εισιτήριο ελέγχουμε εάν αυτό υπάρχει στο seller_queue. Εάν είναι έγκυρο το αίτημα του buyer τότε αναζητείται στην ουρά που περιέχει την προσφορά (seller queue) εισιτήριο που πληροί τις προϋποθέσεις. Ύστερα, εάν βρεθεί αλλάζει ο κάτοχος του εισιτηρίου αυτού, χρεώνεται ο νέος buyer και αποζημιώνεται ο παλιός του owner. Οι λειτουργίες αυτές γίνονται με το trigger search_for_item_once_its_desired
11. Στην περίπτωση που επιχειρήσει ένας owner να διαθέσει προς πώληση ήδη ενεργοποιημένο εισιτήριο, αποτυγχάνει η ενέργεια αυτή. Εάν είναι έγκυρο το εισιτήριο προς πώληση αναζητείται entry στην ουρά ζήτησης (buyer queue) που μπορεί να ικανοποιήσει και όταν βρεθεί γίνεται η αλλαγή της κατοχής του

και η απαραίτητη χρέωση/πίστωση. Οι λειτουργίες αυτές γίνονται με το trigger search_for_item_once_its_supplied

12. Η βάση δεδομένων έχει δομηθεί με τέτοιο τρόπο έτσι ώστε να περιορίζονται όσο το δυνατό περισσότερο τα λάθη των καθημερινών χρηστών της. Η ουρά μεταπώλησης σαν οντότητα εμπλέκει τον χρήστη περισσότερο από κάθε άλλη επομένως για την είσοδο δεδομένων σε αυτή έχει δημιουργηθεί μια συνάρτηση που υποχρεώνει τον owner να διαθέσει ένα δικό του εισιτήριο για ένα event που μπορεί να πωληθεί , δηλαδή δεν έχει περάσει και είναι sold out. Οι λειτουργίες αυτές γίνονται με το procedure insert_into_seller_queue
13. Καθαρίζουμε τις ουρές με τα events clear_owners_buyers και clear_queues.
14. Με το event clear_old_resell καθαρίζουμε seller_queue και buyer_queue μετά τη λήξη του φεστιβάλ για εξοικονόμηση χώρου.
15. Επιτρέπουμε σε κάθε επισκέπτη να κάνει review για ένα performance μόνο μια φορά με το trigger review_no_dup.
16. Χειριζόμαστε τους πίνακες review, review_summary έτσι ώστε να έχουμε μαζεμένα τα reviews για ένα καλλιτέχνη και να μπορεί κάθε επισκέπτης να αλλάξει το review του ή και να το διαγράψει με τα triggers review_sum_before_insert,review_sum_after_insert,review_sum_after_update, review_sum_after_delete.
17. Τέλος με το trigger check_review_event_match ελέγχουμε ότι το performance_id στο review table είναι έγκυρο και είναι για ticket του οποίου το event περιέχει αυτό το performance.

Τα παραπάνω triggers λειτουργούν και έχουν προστεθεί δεδομένα που επιβεβαιώνουν τη σωστή χρήση τους. Εμφανίζονται δηλαδή errors κατά την εισαγωγή δεδομένων, ωστόσο αυτά είναι εσκεμμένα με σκοπό να δείξουμε τη λειτουργικότητα των triggers και τον ρεαλισμό της εφαρμογής μας.

Τα δεδομένα στο DML έχουν προστεθεί από την python αντλώντας την πληροφορία από τα csvs και κάνοντας την dump σε append mode σε μορφή sql insert στο αρχείο sql/load.sql. Έτσι ανοίγοντας το βλέπουμε όλα τα insert με την σειρά που πραγματοποιήθηκαν. Προφανώς, αφού έχουμε βάλει δεδομένα για να χτυπάνε τα triggers αυτό το αρχείο δεν μπορεί να τρέχει μόνο του επειδή όταν χτυπήσει το πρώτο trigger, το mysql workbench θα σταματήσει το population της βάσης μας. Συνεπώς, αυτό το αρχείο το έχουμε για καθαρά λόγους πληρότητας και χρησιμοποιούμε τις συναρτήσεις της python για να γεμίσουμε τη βάση. Προφανώς αν θέλαμε να τρέχουμε το DML για να γεμίσουμε τη βάση θα έπρεπε απλά να αφαιρέσουμε τα δεδομένα που καθίστανται μη έγκυρα από τα triggers και procedures μας.
Σημειώνεται ότι οι εικόνες έχουν τη μορφή dummy url όπως υποδείχθηκε σε σχετική ερώτηση από τους διδάσκοντες και ότι αφορούν μόνο τις οντότητες για τις οποίες θα

υπήρχαν φωτογραφίες σε μια πραγματική βάση δεδομένων (παραδείγματος χάριν δεν υπάρχουν φωτογραφίες για το ticket).
Ακόμη, για λόγους απλότητας και ρεαλισμού έχουν χρησιμοποιηθεί μικρά max_capacity στα stages ώστε τα εισιτήρια να επαρκούν. Σε αντίθετη περίπτωση είτε θα χρειάζονταν πολύ περισσότερα εισιτήρια (στην υλοποίηση της βάσης μας εισάγονται 2000) για να καλύψουν τις ανάγκες των σκηνών είτε θα προέκυπταν μη ρεαλιστικά αποτελέσματα όπως " σε μια σκηνή μέγιστης χωρητικότητας 1000 ατόμων τα εισιτήρια ήταν 10".

5. Queries

1. Στο Query 1 βλέπουμε μια σύνοψη των εσόδων του φεστιβάλ και παρουσιάζουμε ανά έτος τα έσοδα που προέκυψαν με bank_account/ credit/ debit όσο και το total revenue.
2. Στο Query 2 βλέπουμε με είσοδο το έτος και το είδος του performance όλους τους καλλιτέχνες που ανήκουν σε αυτό το είδος και το αν συμμετείχαν (τραγουδώντας το είδος αυτό) ή όχι στο φεστιβάλ του συγκεκριμένου έτους.
3. Στο Query 3 βλέπουμε τους καλλιτέχνες που έχουν εμφανιστεί ως warm up πάνω από 2 φορές στο ίδιο φεστιβάλ. Πειράζουμε ανάλογα τα δεδομένα για να έχουμε ικανοποιητικά αποτελέσματα.
4. Στο Query 5 βλέπουμε τον μέγιστο αριθμό των performances σε φεστιβάλ από νέους καλλιτέχνες (ηλικία < 30). Αν έχουμε ισοβαθμία τότε θα έχουμε πολλαπλά rows στην απάντηση του query, αλλιώς πρέπει να έχουμε ένα. Πειράζουμε ανάλογα τα δεδομένα για να δούμε τις διαφορές αυτές.
5. Στο Query 7 βλέπουμε το φεστιβάλ με το χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού.
6. Στο Query 8 βλέπουμε το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία που δίνουμε σαν όρισμα. Στο output έχουν συμπεριληφθεί συμπληρωματικοί πίνακες για την επιβεβαίωση της ορθότητας του αποτελέσματος.
7. Στο Query 9 βλέπουμε τους επισκέπτες που έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με περισσότερες από 3 παρακολουθήσεις.
8. Στο Query 10 βλέπουμε τα ζεύγη είδος-υποείδος που εμφανίστηκαν τις περισσότερες φορές σε φεστιβάλ.
9. Στο Query 11 γίνεται εύρεση όλων των καλλιτεχνών που έχουν συμμετάσχει σε performances τουλάχιστον 5 φορές λιγότερες από τον καλλιτέχνη με τις περισσότερες εμφανίσεις. Αρχικά, υπολογίζεται ο αριθμός εμφανίσεων για κάθε καλλιτέχνη, εντοπίζεται το μέγιστο πλήθος εμφανίσεων και τελικά

επιστρέφονται όσοι έχουν εμφανιστεί τουλάχιστον 5 φορές λιγότερες από αυτόν.

10. Στο Query 12 παρουσιάζεται η κατανομή του προσωπικού ανά ρόλο και ημερομηνία εκδήλωσης. Συγκεκριμένα, για κάθε event date και για κάθε ρόλο προσωπικού, υπολογίζεται πόσα μέλη προσωπικού εργάζονται εκείνη τη μέρα, επιτρέποντας έτσι την παρακολούθηση της στελέχωσης ανά ρόλο και ημέρα φεστιβάλ.
11. Στο Query 13 εντοπίζονται οι καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ που έχουν διεξαχθεί σε τουλάχιστον 3 διαφορετικές ηπείρους. Για κάθε καλλιτέχνη υπολογίζεται ο αριθμός ηπείρων στις οποίες έχει εμφανιστεί, βασιζόμενο στο location των φεστιβάλ όπου συμμετείχε.
12. Στο Query 14 εντοπίζονται τα μουσικά είδη (genres) που είχαν τουλάχιστον 3 εμφανίσεις καλλιτεχνών σε δύο συνεχόμενες χρονιές, με τον ίδιο αριθμό εμφανίσεων και στις δύο χρονιές. Πρώτα υπολογίζονται οι ετήσιες εμφανίσεις ανά είδος, και έπειτα εντοπίζονται τα είδη που είχαν τον ίδιο αριθμό εμφανίσεων σε διαδοχικά έτη.
13. Στο Query 15 παρουσιάζονται οι πέντε κορυφαίοι θεατές (owners) που έχουν δώσει τη μεγαλύτερη συνολική βαθμολογία σε εμφανίσεις συγκεκριμένων καλλιτεχνών. Ο συνολικός βαθμός προκύπτει από το άθροισμα των επιμέρους αξιολογήσεων (ερμηνεία, ήχος & φωτισμός, σκηνική παρουσία, οργάνωση, γενική εντύπωση) για κάθε performance. Το αποτέλεσμα ομαδοποιείται ανά θεατή και καλλιτέχνη.

Παρακάτω θα δούμε αναλυτικά τα αποτελέσματα για τα query 4 και 6 όπως ζητούνται στην εκφώνηση:

Query 4

Στην απλή μορφή του 4ου query παίρνουμε ως όρισμα τα reviews για έναν καλλιτέχνη από το review summary (κάνοντας join τον πίνακα performance) και παρουσιάζουμε τα αποτελέσματα για δύο καλλιτέχνες όσο και το πως προέκυψαν αυτά δείχνοντας traces από τους πίνακες performance και review_summary. Ξεκινάμε με nested loop join (default) και βλέπουμε τα παρακάτω:

- Χωρις force index:

```
| {
  "query_block": {
    "select_id": 1,
    "table": {
      "table_name": "artist",
      "access_type": "const",
      "possible_keys": ["PRIMARY"],
      "key": "PRIMARY",
      "key_length": "4",
      "used_key_parts": ["artist_id"],
      "ref": ["const"],
      "rows": 1,
      "filtered": 100
    },
    "table": {
      "table_name": "review_summary",
      "access_type": "ALL",
      "possible_keys": ["PRIMARY"],
      "rows": 7,
      "filtered": 100
    },
    "table": {
      "table_name": "performance",
      "access_type": "eq_ref",
      "possible_keys": ["PRIMARY", "artist_id"],
      "key": "PRIMARY",
      "key_length": "4",
      "used_key_parts": ["performance_id"],
      "ref": ["pulse_university.review_summary.performance_id"],
      "rows": 1,
      "filtered": 100,
      "attached_condition": "performance.artist_id = 9"
    }
  }
} |
```


- Mε force index:


```
| {
  "query_block": {
    "select_id": 1,
    "table": {
      "table_name": "artist",
      "access_type": "const",
      "possible_keys": ["PRIMARY"],
      "key": "PRIMARY",
      "key_length": "4",
      "used_key_parts": ["artist_id"],
      "ref": ["const"],
      "rows": 1,
      "filtered": 100
    },
    "table": {
      "table_name": "performance",
      "access_type": "ref",
      "possible_keys": ["artist_id"],
      "key": "artist_id",
      "key_length": "5",
      "used_key_parts": ["artist_id"],
      "ref": ["const"],
      "rows": 3,
      "filtered": 100,
      "using_index": true
    },
    "table": {
      "table_name": "review_summary",
      "access_type": "eq_ref",
      "possible_keys": ["PRIMARY"],
      "key": "PRIMARY",
      "key_length": "4",
      "used_key_parts": ["performance_id"],
      "ref": ["pulse_university.performance.performance_id"],
      "rows": 1,
      "filtered": 100
    }
  }
} |
```

Παρατηρούμε ότι χωρίς force index (artist_id) στο performance ο optimizer επιλέγει λανθασμένα το index performance(PRIMARY) και ενώνει πρώτα performance → review_summary κάνοντας συνολικά 9 σαρώσεις. Όταν κάνουμε force index (artist_id) στο performance αναγκάζουμε τον optimizer να μην χρησιμοποιεί το μη σκόπιμο index performance(PRIMARY) με αποτέλεσμα να μη γίνεται full scan του πίνακα review_summary κάνοντας συνολικά 5 σαρώσεις. Το force index λοιπόν εδώ βελτιώνει την απόδοση του query κατά λίγο αφού ο πίνακας review_summary δεν είναι πολύ μεγάλος. Και στα δύο έχουμε nested loop join, ωστόσο στο δεύτερο έχουμε διαφορετική σειρά των join με αποτέλεσμα την παραπάνω επιτάχυνση. Προφανώς σε κάποιο άλλο παράδειγμα θα μπορούσαμε να είχαμε επιβράδυνση των join και αργότερο lookup στους πίνακες μας.

Query 6

Στο Query 6 βλέπουμε το μέσο όρο αξιολόγησης των παραστάσεων που έχει παρακολουθήσει και αξιολογήσει ο δοσμένος ως input owner. Στο αρχείο Q06 έχουν συμπεριληφθεί βοηθητικά queries έτσι ώστε να χρησιμοποιηθούν για τον έλεγχό του owners που έχουν παρακολουθήσει πολλές εμφανίσεις σε διαφορετικές παραστάσεις.

- Χωρίς forced index:


```
{
  "query_block": {
    "select_id": 1,
    "read_sorted_file": {
      "filesort": {
        "sort_key": "reviewed_performances.event_id",
        "table": {
          "table_name": "<derived2>",
          "access_type": "ALL",
          "rows": 2,
          "filtered": 100,
          "attached_condition": "reviewed_performances.performance_id is not null",
          "materialized": {
            "query_block": {
              "select_id": 2,
              "filesort": {
                "sort_key": "review.performance_id",
                "temporary_table": {
                  "table": {
                    "table_name": "ticket",
                    "access_type": "ref",
                    "possible_keys": [
                      "PRIMARY",
                      "no_double_tickets_per_event",
                      "tick_id",
                      "own_id"
                    ],
                    "key": "no_double_tickets_per_event",
                    "key_length": "4",
                    "used_key_parts": ["owner_id"],
                    "ref": ["const"],
                    "rows": 2,
                    "filtered": 100,
                    "using_index": true
                  },
                  "table": {
                    "table_name": "review",
                    "access_type": "ref",
                    "possible_keys": ["performance_id", "ticket_id"],
                    "key": "ticket_id",
                    "key_length": "9",
                    "used_key_parts": ["ticket_id"],
                    "ref": ["pulse_university.ticket.ticket_id"],
                    "rows": 1,
                    "filtered": 100
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
        }
      }
    },
    "table": {
      "table_name": "performance",
      "access_type": "eq_ref",
      "possible_keys": ["PRIMARY"],
      "key": "PRIMARY",
      "key_length": "4",
      "used_key_parts": ["performance_id"],
      "ref": ["reviewed_performances.performance_id"],
      "rows": 1,
      "filtered": 100
    }
  }
}
```


- Με forced index:


```
"query_block": {
  "select_id": 1,
  "read_sorted_file": {
    "filesort": {
      "sort_key": "reviewed_performances.event_id",
      "table": {
        "table_name": "<derived2>",
        "access_type": "ALL",
        "rows": 10,
        "filtered": 100,
        "attached_condition": "reviewed_performances.performance_id is not null",
        "materialized": {
          "query_block": {
            "select_id": 2,
            "filesort": {
              "sort_key": "review.performance_id",
              "temporary_table": {
                "table": {
                  "table_name": "review",
                  "access_type": "ALL",
                  "rows": 10,
                  "filtered": 100,
                  "attached_condition": "review.ticket_id is not null"
                },
                "table": {
                  "table_name": "ticket",
                  "access_type": "eq_ref",
                  "possible_keys": ["PRIMARY"],
                  "key": "PRIMARY",
                  "key_length": "8",
                  "used_key_parts": ["ticket_id"],
                  "ref": ["pulse_university.review.ticket_id"],
                  "rows": 1,
                  "filtered": 100,
                  "attached_condition": "ticket.owner_id = @searched_for_owner"
                }
              }
            }
          }
        }
      }
    }
  },
  "table": {
    "table_name": "performance",
    "access_type": "eq_ref",
    "possible_keys": ["PRIMARY"],
    "key": "PRIMARY",
    "key_length": "4",
    "used_key_parts": ["performance_id"],
    "ref": ["reviewed_performances.performance_id"],
    "rows": 1,
```

Σε αντίθεση με την προηγούμενη άσκηση παρατηρούμε ότι στο συγκεκριμένο παράδειγμα η επιβολή χρήσης του index μπορεί να προκαλέσει μεγάλη αύξηση της καθυστέρησης και να οδηγήσει σε σπατάλη χρόνου και πόρων. Επομένως τα indexes που δημιουργήθηκαν αυτόματα κρίνονται ακατάλληλα για το συγκεκριμένο ερώτημα.

Όπως εξηγείται και αναλυτικότερα πιο κάτω παρόλο που η πολυπλοκότητα O(NxM) δεν είναι στις περισσότερες περιπτώσεις βέλτιστη η επιβολή indexes μπορεί να οδηγήσει σε περαιτέρω σπατάλη "οδηγώντας" λανθασμένα την αναζήτηση, στην περίπτωσή μας στο PRIMARY index του review και όχι στο performance column που μας ενδιαφέρει.

Γενικά

Το nested loop join έχει πολυπλοκότητα O(NxM), χωρίς καθόλου index αφού πρέπει να σαρώσει για κάθε γραμμή του πρώτου πίνακα όλες τις γραμμές του δεύτερου. Το nested loop join με index (όπου και χρησιμοποιούμε εδω) έχει πολυπλοκότητα O(NxlogM) αφού κάνει για κάθε γραμμή του πρώτου πίνακα lookup μέσω ευρετηρίου στον δεύτερο πίνακα. Εαν το mariadb υποστήριζε hash join θα βλέπαμε ότι φτιάχνουμε hash tables μνήμης O(min(N,M)) και το lookup γίνεται σε χρόνο O(N+M) που είναι σαφώς πολύ μικρότερος από αυτόν του nested loop join. Εαν το mariadb υποστήριζε merge join θα είχαμε πάλι πολυπλοκότητα O(N+M) αν οι πίνακες ήταν ταξινομημένοι, αλλιώς θα έπρεπε πρώτα να κάνουμε sorting τους πίνακες που θα καθιστούσε τον αλγόριθμο πολύ πιο αργό από αυτόν του nested loop join. Αν είχαμε ταξινομημένα δεδομένα τότε το merge join θα είχε την καλύτερη επίδοση και από τους δύο παραπάνω αλγορίθμους.

6. Authentication

Πρόσβαση στην βάση δεδομένων πρέπει να έχουν τόσο οι διαχειριστές του διαγωνισμού όσο και οι διαγωνιζόμενοι. Δημιουργούμε έναν χρήση "admin" στον οποίο δίνουμε όλα τα δικαιώματα επί της βάσης, όπως τροποποίηση όλων των δεδομένων, δημιουργία αντιγράφου ασφαλείας και επαναφορά του συστήματος. Η είσοδος στην βάση γίνεται μετά την εισαγωγή του συνθηματικό "secure_password".

7. Οδηγίες Εγκατάστασης

1. Κάνετε clone το Github Repository της εργασίας: https://github.com/Nickp03/Databases-Ntua-Pulse_University_Music_Festival
2. Ανοίξτε τον MySQL server και τον Apache web server (XAMPP).
3. Ανοίγουμε το MySQL Workbench ή το DBeaver.

4. Τρέχουμε το /sql/install.sql σε έναν από τους Graphical clients της επιλογής μας.
5. Τρέχουμε το /code/database_fill.py κάνουμε populate τη βάση και γεμίζουμε το DML.
6. Τρέχουμε το /sql/users.sql σε έναν από τους Graphical clients της επιλογής μας.
7. Η βάση μας είναι έτοιμη και μπορούμε να τη χρησιμοποιήσουμε κάνοντας ερωτήματα και βάζοντας δεδομένα.

Να σημειωθεί ότι δεν επισυνάπτουμε τον κώδικα του DDL ή του DML, καθώς πρόκειται για αρχεία των 1000+ γραμμών και των 10000+ γραμμών αντίστοιχα.