

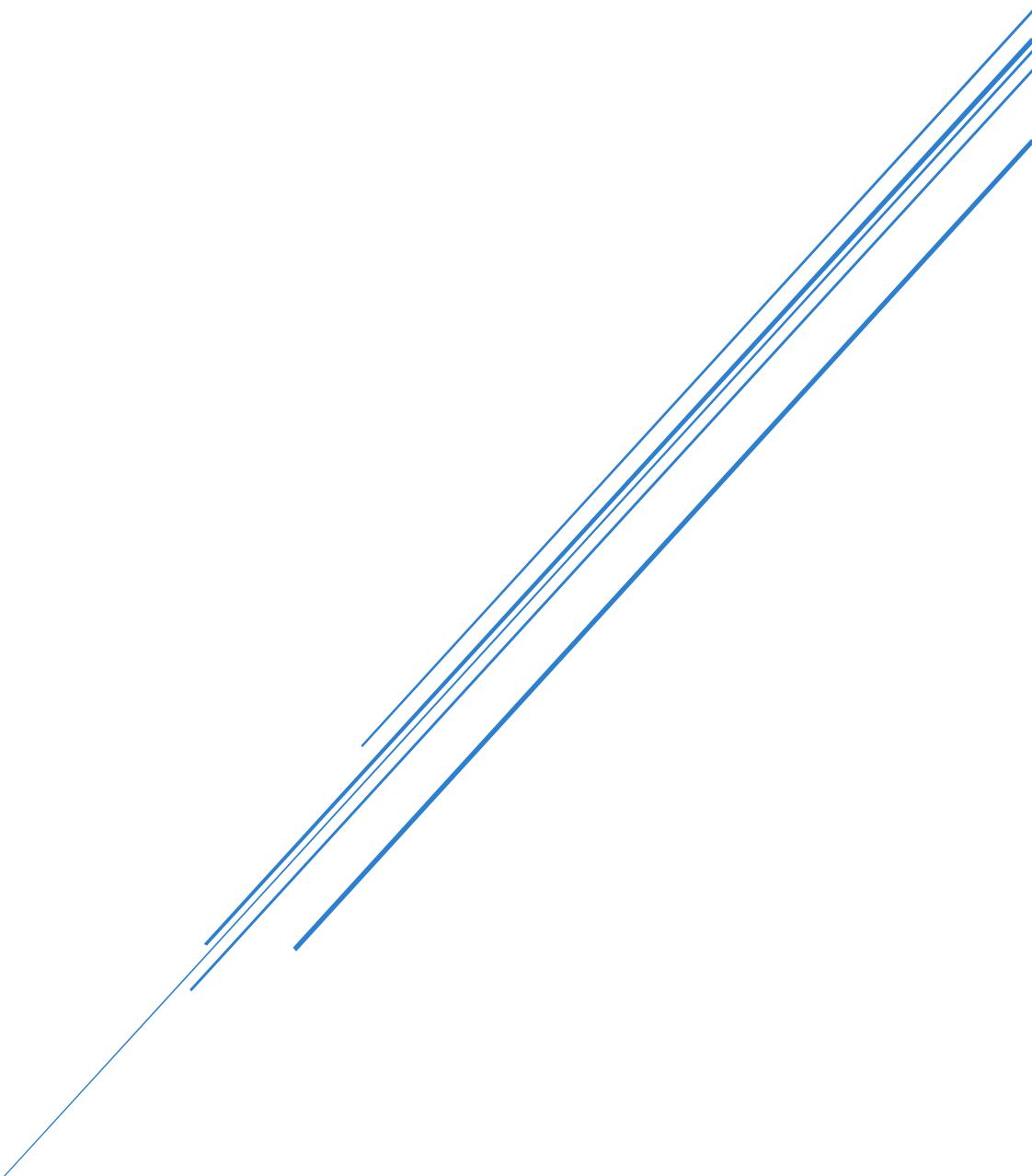
ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ PULSE UNIVERSITY

Σταύρος Αμπανάβας 03122202

Βιολέτα Ζαγγογιάννη 03122213

Σταμάτης-Μάριος Χαντζόπουλος 03122858



Ακ. Έτος: 2024-2025
Σχολή ΗΜΜΥ

Περιεχόμενα

Εισαγωγή	2
Απαιτήσεις Βάσης Δεδομένων	3
Κύριες Οντότητες	3
Σχέσεις	3
Κανόνες και περιορισμοί	3
Υποθέσεις και παραδοχές	4
Σχεδιασμός Βάσης Δεδομένων	6
Relational Diagram	6
ER Diagram	7
Tables	8
Indexes	16
Triggers	17
Procedures	23
Data	30
Demo	34
Queries	39
Βιβλιογραφία και χρήσιμες Ιστοσελίδες	62

Εισαγωγή

Στα πλαίσια του μαθήματος «Βάσεις Δεδομένων» του 6^{ου} Εξαμήνου της Σχολής HMMY, κληθήκαμε να αναπτύξουμε μια βάση δεδομένων για το διεθνές φεστιβάλ μουσικής Pulse University. Αντικείμενο της εργασίας αποτελεί η μελέτη, ο σχεδιασμός και η υλοποίηση μιας βάσης που θα εξυπηρετεί το φεστιβάλ μουσικής, αφού θα αποθηκεύει και θα διαχειρίζεται πληροφορίες, απαραίτητες για την οργάνωση και την διεξαγωγή του.

Το φεστιβάλ αυτό είναι ετήσιο, περιλαμβάνει παραστάσεις τόσο από καλλιτέχνες, όσο και από μπάντες, σε διαφορετική τοποθεσία ανά έτος. Σκοπός της βάσης είναι η πλήρης παρακολούθηση της διοργάνωσης, από την συνολική οργάνωση του φεστιβάλ και των παραστάσεων του, μέχρι και την πώληση των εισιτηρίων και την αξιολόγηση των παραστάσεων από το κοινό.

Η εργασία υλοποιήθηκε χρησιμοποιώντας SQL, σε σύστημα MySQL για τη δημιουργία του σχήματος, της εισαγωγής και της διαχείρισης των δεδομένων, όπως και για την απάντηση ερωτημάτων σχετικά με τη βάση (queries). Επίσης, χρησιμοποιήθηκε python για την αυτοματοποίηση δημιουργίας δεδομένων.

Απαιτήσεις Βάσης Δεδομένων

Αρχικά, για την υλοποίηση ήταν απαραίτητη η καταγραφή και η κατανόηση των απαιτήσεων του συστήματος και βασισμένοι στην εκφώνηση της εργασίας καταλήγουμε στα εξής:

Κύριες Οντότητες

Για την κατασκευή της βάσης δεδομένων θεωρούμε ως κύριες οντότητες τις εξής:

- ❖ *festival*: Τα φεστιβάλ που πραγματοποιούνται
- ❖ *location*: Τοποθεσία διεξαγωγής του φεστιβάλ
- ❖ *events*: Παραστάσεις που γίνονται σε κάθε φεστιβάλ
- ❖ *staff*: Προσωπικό που υποστηρίζει κάθε σκηνή σε κάθε παράσταση
- ❖ *performance*: Σειριακές εμφανίσεις αποτελούν κάθε παράσταση
- ❖ *performers*: Αυτοί που συμμετέχουν στο performance χωρίς διαχωρισμό σε καλλιτέχνη ή μπάντα
- ❖ *visitor*: Επισκέπτες των φεστιβάλ
- ❖ *ticket*: Εισιτήρια παραστάσεων
- ❖ *resaleQueue*: Αυτόματη ουρά μεταπώλησης που λειτουργεί με FIFO
- ❖ *sellers*: Πωλητές εισιτηρίων
- ❖ *buyers*: Αγοραστές εισιτηρίων sold-out παραστάσεων
- ❖ *reviews*: Αξιολογήσεις κατόχων ενεργοποιημένων εισιτηρίων, που μπορούν να αξιολογήσουν με βάση την κλίμακα Likert.

Σχέσεις

- Ένα φεστιβάλ αποτελείται από πολλές παραστάσεις και γίνεται σε συγκεκριμένη τοποθεσία, διαφορετική κάθε χρονιά.
- Κάθε παράσταση πραγματοποιείται σε μία σκηνή και περιλαμβάνει πολλές εμφανίσεις.
- Κάθε εμφάνιση γίνεται από έναν καλλιτέχνη ή μπάντα.
- Κάθε σκηνή απαιτεί συγκεκριμένο πλήθος προσωπικού, διαφορετικό ανά είδος προσωπικού και ανά παράσταση.
- Κάθε εισιτήριο αφορά έναν επισκέπτη και μία παράσταση.
- Οι επισκέπτες μπορούν να κάνουν αξιολογήσεις για εμφανίσεις στις οποίες παραβρέθηκαν.

Κανόνες και περιορισμοί

Βάσει της εκφώνησης προκύπτουν οι εξής κανόνες και περιορισμοί.

- Τα φεστιβάλ είναι ετήσια και πραγματοποιούνται σε συναπτά έτη.
- Το φεστιβάλ, αν είναι πολυύμερο, είναι σε συνεχόμενες ημέρες.
- Κάθε σκηνή μπορεί να φιλοξενεί μόνο μια παράσταση κάθε χρονική στιγμή.
- Το προσωπικό χαρακτηρίζεται από ένα επίπεδο εμπειρίας.
- Το 5% της χωρητικότητας κάθε σκηνής πρέπει να είναι μικρότερο από το πλήθος του security προσωπικού, και το 2% του βοηθητικού προσωπικού.

- Οι εμφανίσεις των καλλιτεχνών είναι σειριακές και ανάμεσα σε δύο διαδοχικές το διάλειμμα έχει διάρκεια 5 έως 30 λεπτών. Μέγιστη διάρκεια των εμφανίσεων είναι 3 ώρες.
- Ως καλλιτέχνες ορίζονται είτε σόλο καλλιτέχνες είτε μέλη συγκροτημάτων.
- Ένας καλλιτέχνης μπορεί να ανήκει σε παραπάνω από ένα συγκροτήματα.
- Ένας καλλιτέχνης δεν μπορεί να εμφανίζεται σε 3 συναπτά φεστιβάλ.
- Ένας επισκέπτης μπορεί να αγοράσει εισιτήρια για όσες μέρες του κάθε φεστιβάλ θέλει και το καθένα αναφέρεται σε συγκεκριμένη παράσταση. Δεν μπορεί, όμως να αγοράσει εισιτήρια για περισσότερες από μια παραστάσεις την ίδια ημέρα του φεστιβάλ, ούτε προφανώς για την ίδια παράσταση.
- Οι τρόποι αγοράς των εισιτηρίων είναι είτε με χρεωστική κάρτα ή με πιστωτική ή με τραπεζικό λογαριασμό.
- Κάθε εισιτήριο μπορεί να χρησιμοποιηθεί μόνο μια φορά, αφού γίνεται επικύρωση με την είσοδο του κάθε επισκέπτη στον χώρο της παράστασης.
- Το κόστος των εισιτηρίων διαφέρει ανά ημέρα του φεστιβάλ.
- Τα εισιτήρια VIP μπορούν να αποτελούν το πολύ το 10% της χωρητικότητας κάθε σκηνής.
- Η ουρά μεταπώλησης ενεργοποιείται αυτόματα όταν γίνεται sold-out ένα event.
- Ένας ενδιαφερόμενος αγοραστής μπορεί να αναζητήσει εισιτήριο για συγκεκριμένη παράσταση και κατηγορία εισιτηρίου, ή για συγκεκριμένο εισιτήριο.
- Η πώληση εισιτηρίου πραγματοποιείται αυτόματα.
- Η ουρά υλοποιείται με FIFO.
- Οι χρηματικές συναλλαγές πραγματοποιούνται αυτόματα.
- Μόνο οι κάτοχοι εισιτηρίων για μια συγκεκριμένη παράσταση μπορούν να αξιολογήσουν την παράσταση αυτή, και μόνο μία φορά.
- Δεν υπάρχουν ακυρώσεις σε φεστιβάλ/παραστάσεις/εισιτήρια.

Υποθέσεις και παραδοχές

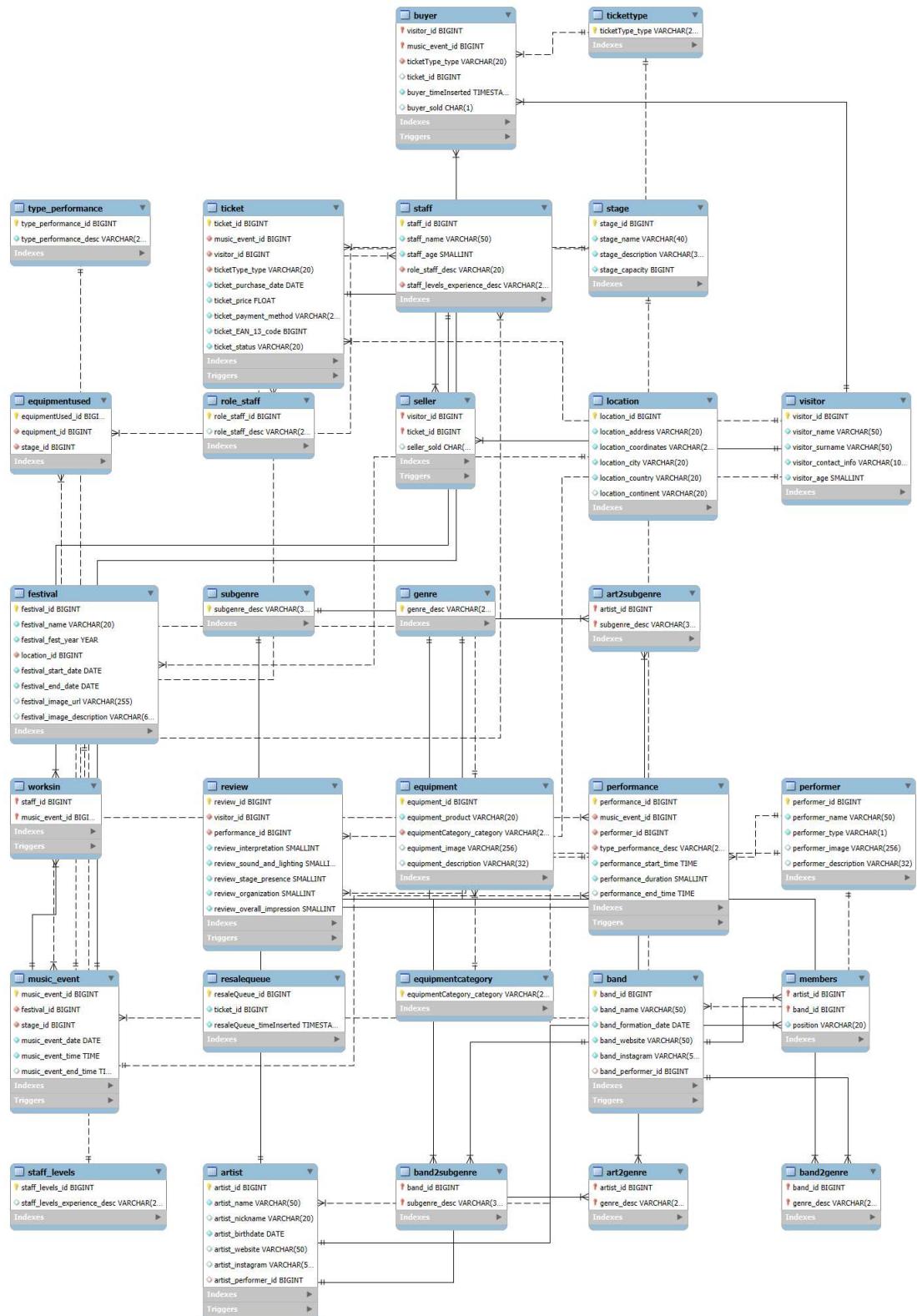
Για την υλοποίηση της Βάσης Δεδομένων που μας ζητήθηκε, κάναμε τις εξής παραδοχές:

- ✓ Όλες οι ημέρες ενός φεστιβάλ πρέπει να ανήκουν στο ίδιο έτος. Δηλαδή, ένα φεστιβάλ δεν μπορεί να ξεκινάει τον Δεκέμβριο και να συνεχίζεται μέχρι τον Ιανουάριο.
- ✓ Δεν χρειάζεται να υπάρχει event κάθε ημέρα του φεστιβάλ.
- ✓ Ένα event πραγματοποιείται μόνο μία φορά.
- ✓ Ένα performance πραγματοποιείται μόνο μία φορά σε ένα συγκεκριμένο event.
- ✓ Τα performances δεν μπορούν να ξεκινάνε μετά τις 23:59. Δηλαδή, κάθε performance πρέπει να ξεκινάει την ίδια ημέρα με το αντίστοιχο event.
- ✓ Τα performances ενός event πρέπει να γίνονται insert με την σωστή σειρά. Αυτό συμβαίνει για να μπορεί η Βάση αυτόματα να υπολογίζει το διάλειμμα μεταξύ δύο διαδοχικών performances και να βρίσκει αν είναι εντός των ορίων που θέλουμε.
- ✓ Δεν μπορούν να αλλάξουν οι ημερομηνίες και οι ώρες ενός φεστιβάλ/ event/ performance από την στιγμή που αυτό εισαχθεί στην βάση.
- ✓ Η Βάση δεν θα ελέγχει αυτόματα αν σε κάθε event έχει ανατεθεί ικανοποιητικό πλήθος προσωπικού. Ωστόσο, υπάρχει το Procedure *checkEnoughStaff* το οποίο θα το καλεί ο χρήστης για το event που τον ενδιαφέρει, ώστε να ελέγχει την πληρότητα του staff.
- ✓ Η Βάση δεν υποθέτει ότι όλα τα εισιτήρια, ίδιου τύπου, για το ίδιο event έχουν την ίδια τιμή ή ότι ένα είδος εισιτηρίου πρέπει πάντα να έχει μεγαλύτερη τιμή από ένα άλλο, πχ το VIP να είναι πιο ακριβό από τα άλλα.

- ✓ Οι ηλικίες του προσωπικού και των επισκεπτών του φεστιβάλ θεωρούνται ότι είναι με βάση την τρέχουσα χρονική στιγμή και δεν θα γίνεται αναδρομικός έλεγχος για προηγούμενα φεστιβάλ. Θεωρούμε ότι ο χρήστης έχει δώσει σωστά δεδομένα.
- ✓ Δεν ελέγχεται αυτόματα αν κάποια σκηνή βρίσκεται πράγματι στην τοποθεσία που διεξάγεται το φεστιβάλ. Θεωρούμε ότι τα δεδομένα είναι σωστά.
- ✓ Δεν ελέγχεται αυτόματα αν η ημερομηνία αγοράς του εισιτηρίου είναι σωστή. Θεωρούμε ότι τα δεδομένα είναι σωστά.
- ✓ Οι φωτογραφίες και οι περιγραφές τους θα έχουν μία default τιμή και θα είναι ευθύνη του διαχειριστή της ιστοσελίδας του φεστιβάλ να βάλει κατάλληλα δεδομένα.
- ✓ Οι φωτογραφίες δίνονται ως URL.
- ✓ Τα μουσικά υποείδη είναι ανεξάρτητα από τα μουσικά είδη. Ένας καλλιτέχνης μπορεί να ανήκει σε περισσότερα από ένα.
- ✓ Ένας επισκέπτης αν πουλήσει το εισιτήριό του και το ξαναγοράσει, δεν μπορεί να το διαθέσει ξανά για πώληση.

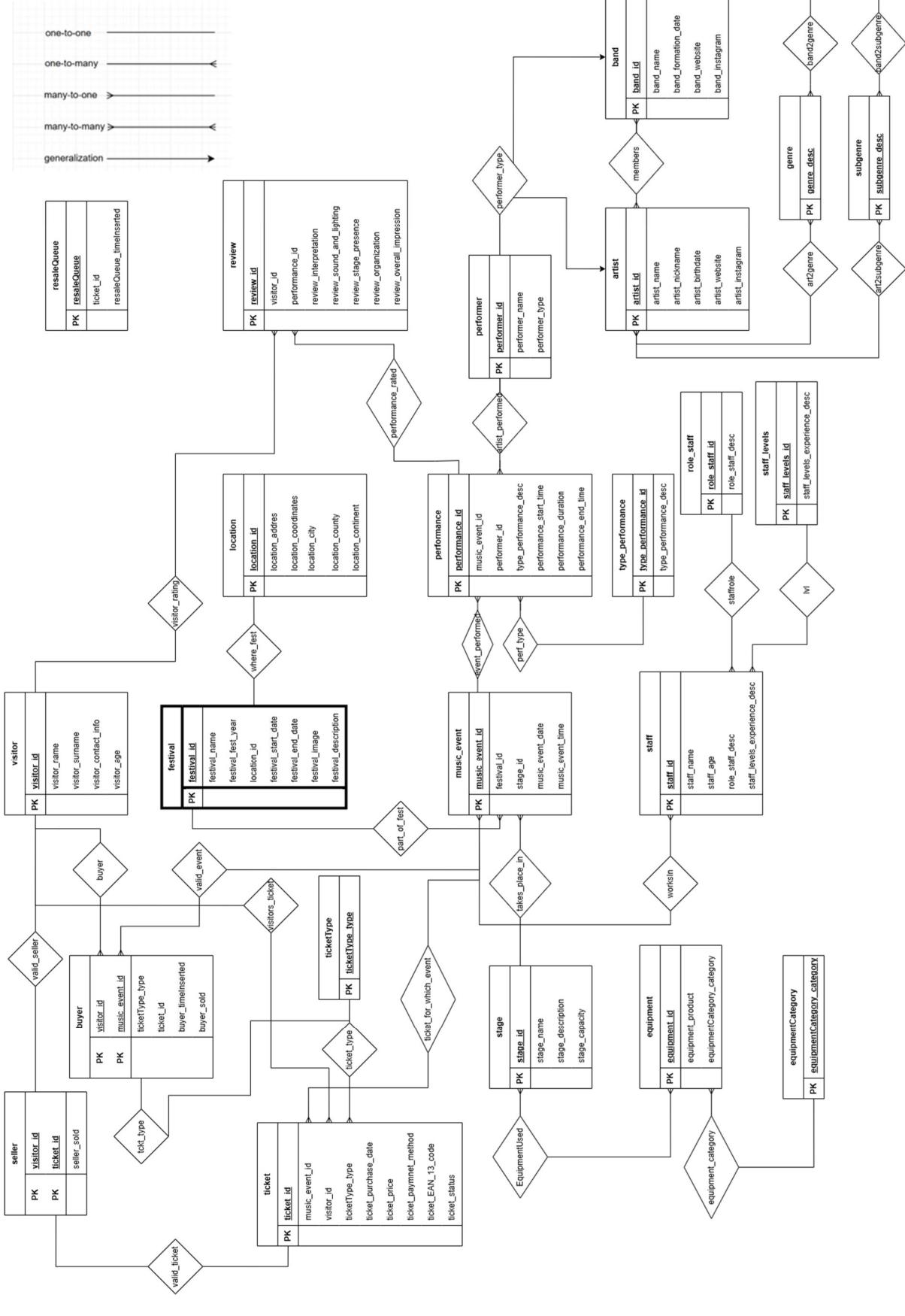
Σχεδιασμός Βάσης Δεδομένων

Relational Diagram



ER Diagram

Υπόμνημα για το ER Diagram:



Tables

Για τον σχεδιασμό του Schema για το διεθνές φεστιβάλ μουσικής Pulse University υλοποιήσαμε τα παρακάτω Tables.

➤ *festival*

Το συγκεκριμένο table αναφέρεται στα φεστιβάλ τα οποία πραγματοποιούνται και περιέχει τα εξής attributes:

- *festival_id*
- *festival_name* (όνομα φεστιβάλ)
- *festival_fest_year* (έτος πραγματοποίησης φεστιβάλ)
- *location_id* (τοποθεσία φεστιβάλ)
- *festival_start_date* (ημερομηνία έναρξης φεστιβάλ)
- *festival_end_date* (ημερομηνία λήξης φεστιβάλ)
- *festival_image_url*
- *festival_image_description*

To *festival_id* είναι μοναδικό και χαρακτηρίζει κάθε φεστιβάλ, όπως και το *festival_fest_year*, αφού τα φεστιβάλ είναι ετήσια. Ως primary key χρησιμοποιείται το *festival_id*, και δημιουργείται σχέση για το *location_id* με το αντίστοιχο table *location*. Επιπλέον, ελέγχεται ότι το φεστιβάλ έχει έγκυρη διάρκεια (*festival_start_date < festival_end_date*) και ότι οι ημερομηνίες αυτές αντιστοιχούν στο έτος που πραγματοποιείται το εκάστοτε φεστιβάλ.

➤ *location*

Το *location* είναι ένα table που περιέχει όλες τις πληροφορίες για την τοποθεσία του φεστιβάλ.

Attributes

- *location_id*
- *location_address*
- *location_coordinates*
- *location_city*
- *location_country*
- *location_continent*

Primary key αποτελεί το *location_id* και γίνεται έλεγχος ότι το *location_continent* είναι όντως μια υπαρκτή ήπειρος.

➤ *stage*

Το *stage* αποτελείται από όλες τις δυνατές σκηνές που μπορεί να πραγματοποιηθεί μία παράσταση (*event*).

Attributes

- *stage_id*
- *stage_name* (Όνομα σκηνής)
- *stage_description* (Περιγραφή σκηνής)
- *stage_capacity* (Χωρητικότητα σκηνής)

Primary key του table είναι το *stage_id*.

➤ *music_event*

Κάθε φεστιβάλ περιλαμβάνει παραστάσεις/*events*.

Attributes

- *music_event_id*
- *festival_id*
- *stage_id*
- *music_event_date* (Ημερομηνία event)
- *music_event_time* (Ωρα έναρξης event)

To *music_event_id* αποτελεί το primary key του table. Το event αντιστοιχίζεται στο *festival* μέσω των *id* τους και λαμβάνει χώρα στο *stage* που αποτελεί ένα επιπλέον foreign key.

➤ *staff*

Κάθε σκηνή για κάθε παράσταση χρειάζεται συγκεκριμένο προσωπικό/ staff.

Attributes

- *staff_id*
- *staff_name*
- *staff_age*
- *role_staff_desc* (Περιγραφή ρόλου του προσωπικού)
- *staff_levels_experience_desc* (Επίπεδο εμπειρίας προσωπικού)

To primary key του table *staff* είναι το *staff_id*. Κάθε μέλος του προσωπικού ανήκει σε μία κατηγορία, όπως τεχνικό προσωπικό, προσωπικό ασφαλείας και βοηθητικό προσωπικό (technical, security, support). Η σχέση αυτή είναι many-to-one, συνεπώς για την υλοποίηση αυτής της σχέσης κατασκευάστηκε το table *role_staff*, που συνδέεται με foreign key με το table *staff*. Όμοια επειδή το προσωπικό χαρακτηρίζεται σε 5 επίπεδα εμπειρίας, το οποίο αποτελεί σχέση many-to-many, δημιουργήθηκε ο πίνακας *staff_levels*. Επίσης, απαιτούμε όλα τα μέλη του *staff* να είναι ενήλικα.

➤ *role_staff*

To table για την περιγραφή του προσωπικού ως προς την κατηγορία στην οποία ανήκει.

Attributes

- *role_staff_id*
- *role_staff_desc*

Primary key αποτελεί το *role_staff_id* και το *role_staff_desc* περιλαμβάνει τον χαρακτηρισμό ως technical, security ή support.

➤ *staff_levels*

To συγκεκριμένο table αξιοποιείται για τη many-to-one σχέση μεταξύ προσωπικού και χαρακτηρισμού εμπειρίας του.

Attributes

- *staff_levels_id*
- *staff_levels_experience_desc*

Ως primary key χρησιμοποιείται το *staff_levels_id* και το *staff_levels_experience_desc* ορίζει τα επίπεδα εμπειρίας ως ειδικευόμενος, αρχάριος, μέσος, έμπειρος, πολύ έμπειρος.

➤ *worksIn*

Για την ανάθεση εργασίας στο προσωπικό χρησιμοποιείται ο παρών πίνακας. Κάθε μέλος του προσωπικού μπορεί να δουλεύει σε συγκεκριμένο event. Η σχέση αυτή είναι many-to-many.

Attributes

- staff_id
- music_event_id

Ως primary key χρησιμοποιείται το (*staff_id, music_event_id*). Το primary key είναι ένα σύνθετο πρωτεύον κλειδί το οποίο εξυπηρετεί τη many-to-many σχέση μεταξύ προσωπικού και event. Το σύνθετο primary key αποτελείται από δύο στήλες, και ο συνδυασμός των δύο αυτών πεδίων είναι μοναδικός για κάθε εγγραφή στον πίνακα, αφού ένα μέλος του προσωπικού δεν μπορεί να ανατεθεί πολλαπλές φορές στο ίδιο event. Ως foreign keys χρησιμοποιούνται και τα δύο attributes για να συνδέσουν το table με τα tables *staff* και *music_event*.

➤ *equipment*

Ο πίνακας αυτός αφορά τον εξοπλισμό που είναι διαθέσιμος για τα φεστιβάλ.

Attributes

- equipment_id
- equipment_product (Όνομα τεμαχίου εξοπλισμού)
- equipmentCategory_category (Κατηγορία τεμαχίου εξοπλισμού)
- equipment_image
- equipment_description

To *equipment_id* αποτελεί το primary key του table και το *equipmentCategory_category* σχετίζεται ως foreign key με το table *equipmentCategory*.

➤ *equipmentCategory*

Ο πίνακας αυτός χρησιμοποιείται για τον διαχωρισμό του εξοπλισμού σε κατηγορίες ανάλογα με τη χρήση τους.

Attribute

- equipmentCategory_category

To attribute αυτό αποτελεί και το primary key.

➤ *equipmentUsed*

To table αυτό χρειάζεται ώστε να συσχετίσει τον εξοπλισμό με το αντίστοιχο stage.

Attributes

- equipment_id (Τεμάχιο εξοπλισμού)
- stage_id (Σκηνή)

Ως primary key λειτουργεί το ζευγάρι (*equipment_id, stage_id*) αφού εξ ορισμού είναι μοναδικό.

➤ *performance*

Κάθε παράσταση/event αποτελείται από εμφανίσεις/ performances καλλιτεχνών.

Attributes

- performance_id
- music_event_id (Event που πραγματοποιείται το performance)
- performer_id (Εμφανιζόμενος καλλιτέχνης)

- *type_performance_desc* (Είδος εμφάνισης)
- *performance_start_time* (Ωρα έναρξης)
- *performance_duration* (Διάρκεια)
- *performance_end_time* (Τέλος performance)

Το primary key του table είναι το *performance_id*. Το table αυτό αντιστοιχίζεται, μέσω foreign keys στα tables *event*, *performer* και *type_performance*. Επίσης, ελέγχεται ότι η διάρκεια της παράστασης είναι το πολύ 3 ώρες και η ώρα λήξης υπολογίζεται αυτόματα από trigger.

➤ *type_performance*

Χαρακτηρισμός είδους performance ως Warm-up, Headline ή Special Guest.

Attributes

- *type_performance_id*
- *type_performance_desc* (Χαρακτηρισμός ως Warm-up, Headline ή Special Guest)

Το *type_performance_id* συνιστά το primary key του table.

➤ *performer*

Σε κάθε performance συμμετέχει είτε ένας καλλιτέχνης είτε μια μπάντα, τα οποία ενοποιούμε σε ένα table *performer*.

Attributes

- *performer_id*
- *performer_name* (Όνομα καλλιτέχνη είτε μπάντας)
- *performer_type* (Τύπος καλλιτέχνη: A για artist, B για band)
- *performer_image*
- *performer_description*

Το πεδίο *performer_id* χρησιμεύει ως το primary κλειδί του πίνακα και γίνεται έλεγχος ότι το *performer_type* είναι είτε A είτε B.

➤ *artist*

Πίνακας που περιέχει τους καλλιτέχνες και τα στοιχεία τους.

Attributes

- *artist_id*
- *artist_name* (Όνομα)
- *artist_nickname* (Ψευδώνυμο)
- *artist_birthdate* (Ημερομηνία γέννησης)
- *artist_website* (Ιστοσελίδα)
- *artist_instagram* (Instagram profile)
- *artist_performer_id* (Σύνδεση με performer)

Πρωτεύον κλειδί του κάθε καλλιτέχνη αποτελεί το *artist_id* και foreign το *artist_performer_id*.

➤ *band*

Οι μπάντες και στα στοιχεία τους περιέχονται στον πίνακα.

Attributes

- *band_id*
- *band_name* (Όνομα μπάντας)
- *band_formation_date* (Ημερομηνία δημιουργίας μπάντας)

- *band_website* (Ιστοσελίδα)
- *band_instagram* (Instagram profile)
- *band_performer_id* (Σύνδεση με performer)

Κάθε τιμή του *band_id* χρησιμεύει ως primary key, και όπως και στους καλλιτέχνες συνδέεται με το performer μέσω foreign key.

➤ *members*

Η κάθε μπάντα αποτελείται από τα μέλη της, τα οποία είναι καταγεγραμμένα ως *artists*.

Attributes

- *artist_id*
- *band_id*
- *position* (Θέση μέλους του συγκροτήματος πχ *Lead Singer, Drummer*)

Ως primary key χρησιμοποιείται το (*artist_id, band_id*) ως ένα σύνθετο κλειδί που συνδέει τον πίνακα των *artist* με αυτόν των *bands* και επιτρέπει τόσο σε μια μπάντα να έχει πολλούς καλλιτέχνες όσο και σε έναν καλλιτέχνη να ανήκει σε πολλές μπάντες.

➤ *genre*

Ο πίνακας με τα είδη μουσικής.

Attribute

- *genre_desc* είδος μουσικής

To attribute του πίνακα αποτελεί και το primary key του.

➤ *subgenre*

Υποείδη μουσικής.

Attribute

- *subgenre_desc*

To attribute του πίνακα αποτελεί και το primary key του.

➤ *art2subgenre*

Σύνδεση υποειδών μουσικής με καλλιτέχνη

Attributes

- *artist_id*
- *subgenre_desc*

To primary key του πίνακα είναι ο συνδυασμός (*artist_id, subgenre_desc*), με foreign keys τις επιμέρους συνιστώσες.

➤ *band2subgenre*

Σύνδεση υποειδών μουσικής με καλλιτέχνη

Attributes

- *artist_id*
- *subgenre_desc*

Όμοια με το προηγούμενο table, primary key ορίζεται το (*band_id, subgenre_desc*), και foreign keys τα *subgenre_desc* και *band_id*.

➤ *art2genre*

Πίνακας που εξυπηρετεί τη σχέση many-to-many μεταξύ *artist* και *genre*.

Attributes

- *artist id*
- *genre desc*

Για το συγκεκριμένο table χρησιμοποιούμε το σύνθετο primary key (*artist_id, genre_desc*), και καθεμία από τις συνιστώσες του primary κλειδιού συνδέεται ως foreign key με τον αντίστοιχο πίνακα.

➤ *band2genre*

Σύνδεση είδους μουσικής με μπάντα, όμοια όπως με καλλιτέχνη.

Attributes

- *band id*
- *genre desc*

Το primary key αυτού του table είναι σύνθετο και αποτελείται από τα attributes του πίνακα που είναι και foreign keys.

➤ *ticket*

Ο πίνακας αυτός περιλαμβάνει τα εισιτήρια για τα event.

Attributes

- *ticket id*
- *music_event_id* (Event για το οποίο είναι το εισιτήριο)
- *visitor_id* (Ποιος επισκέπτης έχει το εκάστοτε εισιτήριο)
- *ticketType_type* (Είδος εισιτηρίου VIP, BACKSTAGE, REGULAR, STUDENT)
- *ticket_purchase_date* (Ημέρα αγοράς εισιτηρίου)
- *ticket_price* (Τιμή εισιτηρίου)
- *ticket_payment_* (Τρόπος πληρωμής DEBIT, CREDIT ή BANK_DEPOSIT)
- *ticket_EAN_13_code* (Κώδικας EAN-13 σε αριθμητική μορφή)
- *ticket_status* (Κατάσταση εισιτηρίου USED, NOT USED ή FOR SALE)

Το primary key του πίνακα είναι το *ticket_id* και ο πίνακας συνδέεται με του πίνακες *music_event*, *visitor* και *ticket_type* με foreign keys. Ελέγχουμε ότι η τιμή του εισιτηρίου είναι θετική και ότι ο κωδικός EAN-13 είναι μοναδικός.

➤ *ticketType*

Table για τους τύπους των εισιτηρίων.

Attribute

- *ticketType type*

Το μοναδικό attribute του πίνακα αποτελεί και το primary key του.

➤ *visitor*

Πίνακας για τους επισκέπτες των φεστιβάλ.

Attributes

- *visitor id*
- *visitor_name*
- *visitor_surname*
- *visitor_contact_info*
- *visitor_age*

Ος πρωτεύον κλειδί του πίνακα ορίζεται το *visitor_id* και γίνεται έλεγχος ότι δίνεται valid ηλικία για τον επισκέπτη.

➤ *seller*

Ο πίνακας περιλαμβάνει visitors που έχουν αγοράσει εισιτήριο και θέλουν να το πουλήσουν.

Attributes

- *visitor_id*
- *ticket_id*
- *seller_sold* ('Y' αν το εισιτήριο πουλήθηκε, αλλιώς 'N')

Ος σύνθετο primary key λειτουργεί ο συνδυασμός (*visitor_id, ticket_id*), το καθένα από τα οποία είναι και foreign key.

➤ *buyer*

Ο πίνακας αυτός περιέχει τους visitors που θέλουν να αγοράσουν εισιτήριο.

Attributes

- *visitor_id*
- *music_event_id* (Event για το οποίο ψάχνει ο buyer εισιτήριο)
- *ticketType_type* (Τι τύπο εισιτηρίου ψάχνει ο buyer)
- *ticket_id* (Αν ψάχνει ο buyer συγκεκριμένο εισιτήριο)
- *buyer_timeInserted* (Ωρα που μπήκε ο buyer στο resaleQueue)
- *buyer_sold* ('Y' αν ο buyer πήρε εισιτήριο, αλλιώς 'N')

To primary key του table αυτού είναι ένα σύνθετο κλειδί που ορίζεται ως εξής (*visitor_id, music_event_id*). Ο buyer αντιστοιχίζεται μέσω foreign keys με τα visitor, *music_event* και *ticket_type*.

➤ *resaleQueue*

Το table υλοποιεί την ουρά μεταπώλησης εισιτηρίων για τα sold-out φεστιβάλ. Δηλαδή, αποθηκεύονται όσα εισιτήρια είναι διαθέσιμα για μεταπώληση.

Attributes

- *resaleQueue_id*
- *ticket_id*
- *resaleQueue_timeInserted*

Primary key του table αποτελεί το *resaleQueue_id*.

➤ *review*

Table για τις αξιολογήσεις των επισκεπτών.

- *visitor_id*
- *performance_id*
- *review_interpretation*
- *review_sound_and_lighting*
- *review_stage_presence*
- *review_organization*
- *review_overall_impression*

Ο συνδυασμός (*visitor_id, performance_id*) είναι το primary key του πίνακα, με τα *visitor_id* και *performance_id* να είναι foreign keys. Γίνεται έλεγχος ότι οι αξιολογήσεις πληρούν την κλίμα Likert.

Παράδειγμα υλοποίησης table

Επιλέγουμε το table festival για την επίδειξη της υλοποίησης ενός table.

```
DROP TABLE IF EXISTS festival;
CREATE TABLE festival (
    festival_id BIGINT NOT NULL AUTO_INCREMENT,
    festival_name VARCHAR(20) NOT NULL,
    festival_fest_year YEAR NOT NULL UNIQUE,
    location_id BIGINT NOT NULL UNIQUE,
    festival_start_date DATE NOT NULL UNIQUE,
    festival_end_date DATE NOT NULL UNIQUE,
    festival_image_url VARCHAR(255) DEFAULT 'https://images.unsplash.com/photo-1533174072545-7a4b6ad7a6c3?q=80&w=2070&auto=format&fit='
    festival_image_description VARCHAR(64) DEFAULT 'This is a festival!',
    PRIMARY KEY(festival_id),
    CONSTRAINT where_fest FOREIGN KEY (location_id) REFERENCES location(location_id),
    CONSTRAINT festival_date CHECK (festival_start_date < festival_end_date),
    CONSTRAINT festival_year CHECK (YEAR(festival_start_date) = festival_fest_year AND YEAR(festival_end_date) = festival_fest_year)
```

Ο παραπάνω κώδικας αναλύεται στα: Το *DROP TABLE IF EXISTS* διαγράφει τον πίνακα μόνο αν υπάρχει ήδη, χωρίς να δημιουργεί σφάλμα αν δεν υπάρχει. Στη συνέχεια μέσω του *CREATE TABLE* δημιουργούμε τον πίνακα και ύστερα ορίζουμε τις στήλες και τους περιορισμούς του πίνακα. Αναλυτικά, με την εντολή *festival_id BIGINT NOT NULL AUTO_INCREMENT*, ορίζουμε το όνομα της πρώτης στήλης ως *festival_id*, τον τύπο δεδομένων ως *BIGINT*, που είναι ένας ακέραιος μεγάλης κλίμακας, συγκεκριμένα χωρίς πρόσημο εκφράζει αριθμούς στο εύρος 0 έως 18446744073709551615. Το *festival_id* είναι μη μηδενικό λόγω του *NOT NULL* και το *AUTO_INCREMENT* επιτρέπει να δημιουργηθεί ένας μοναδικός αριθμός αυτόματα μόλις γίνει *INSERT* στο Table. Οι άλλοι τύποι μεταβλητών που χρησιμοποιούνται είναι το *VARCHAR(size)* που αποτελεί ένα string μέγιστου μήκους *size*, το *YEAR* που είναι ένα έτος σε μορφή τεσσάρων ψηφίων και το *DATE* που είναι ημερομηνία στο format YYYY-MM-DD. Παράλληλα χρησιμοποιούνται και constraints. Το *UNIQUE* είναι ένα constraint που διασφαλίζει ότι όλες οι τιμές σε μία στήλη είναι διαφορετικές και το *DEFAULT* είναι ένα constraint που παρέχει μια προκαθορισμένη (default) τιμή για μια στήλη. Το *Primary Key* ξεχωρίζει μοναδικά κάθε γραμμή του table. Γενικά τα primary keys πρέπει να περιέχουν μοναδικές μη μηδενικές τιμές (*UNIQUE, NOT NULL*). Το *Foreign Key* είναι ένα πεδίο σε ένα table που αναφέρεται σε μία στήλη ενός άλλου table με στόχο να αποτρέψει ενέργειες που θα κατέστρεφαν τον σύνδεσμο μεταξύ δύο tables. Το *CHECK* περιορίζει την τιμή που μπορεί να εισαχθεί σε μια στήλη.

Indexes

Τα indexes είναι ευρετήρια που φτιάχνονται με βάση μία συγκεκριμένη στήλη. Αυτό που κάνουν είναι να κρατάνε σε κάποια δομή δεδομένων τα entry κάθε σχέσης ταξινομημένα με βάση την τιμή αυτής της στήλης. Έτσι, αν σε κάποιο query ψάχνουμε να βρούμε tuples με βάση την στήλη στην οποία έχουμε βάλει index αντί να ψάξουμε όλη την σχέση θα ανατρέξουμε στο αντίστοιχο ευρετήριο.

Για να τρέχουν πιο γρήγορα τα Queries χρησιμοποιούμε indexes. Η MySQL φτιάχνει αυτόματα indexes στα primary keys κάθε πίνακα, καθώς και σε όσες στήλες χρησιμοποιούν foreign key σε κάποιο primary key ενός άλλου table ή σε στήλη με τον χαρακτηρισμό UNIQUE. Για παράδειγμα στον πίνακα festival έχουν δημιουργηθεί τα κάτωθι indexes:

Result Grid Filter Rows: Export: Wrap Cell Content: <input type="checkbox"/>																
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	E:		
festival	0	PRIMARY	1	festival_id	A	10	NULL	NULL	BTREE				YES	NULL		
festival	0	festival_fest_year	1	festival_fest_year	A	10	NULL	NULL	BTREE				YES	NULL		
festival	0	location_id	1	location_id	A	10	NULL	NULL	BTREE				YES	NULL		
festival	0	festival_start_date	1	festival_start_date	A	10	NULL	NULL	BTREE				YES	NULL		
festival	0	festival_end_date	1	festival_end_date	A	10	NULL	NULL	BTREE				YES	NULL		

Επίσης, βάλαμε index στην εξής στήλη:

- στον τρόπο με τον οποίο αγοράστηκε το εισιτήριο. Για το πρώτο query πρέπει να βρούμε πόσα εισιτήρια αγοράστηκαν με κάποιον συγκεκριμένο τρόπο. Οπότε αντί να ψάχνουμε όλη την σχέση, κοιτάμε απλά το αντίστοιχο ευρετήριο.

```
CREATE INDEX ticket_buy_method ON ticket(ticket_payment_method);
```

Triggers

Για να γίνεται έλεγχος του business logic χρησιμοποιήσαμε διάφορα Triggers. Σε μερικά από αυτά χρησιμοποιούνται και κάποια Procedures. Θα αναλύσουμε πρώτα τα Trigger και μετά τα Procedures.

enoughStaffUpd/enoughStaffDel

Αυτά τα triggers φροντίζουν ότι όταν πάμε να προσθέσουμε ή να αφαιρέσουμε κάποιον εργαζόμενο από κάποιο event, ο νέος αριθμός εργαζομένων δεν θα είναι μικρότερος από τους περιορισμούς που θέτει η εταιρεία που διοργανώνει το φεστιβάλ. Το μόνο που χρειάζεται είναι τα Trigger να καλέσουν την Procedure *checkEnoughStaff*. Για να ελέγχουμε το καινούριο πλήθος των εργαζομένων μετά το update/delete τα trigger πρέπει να είναι *AFTER UPDATE/DELETE*.

```
DELIMITER //
CREATE TRIGGER enoughStaffUpd AFTER UPDATE ON worksIn FOR EACH ROW
BEGIN
    CALL checkEnoughStaff(NEW.music_event_id);
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER enoughStaffDel AFTER DELETE ON worksIn FOR EACH ROW
BEGIN
    CALL checkEnoughStaff(OLD.music_event_id);
END;
//
DELIMITER ;
```

staffInsert

Κάθε φορά, πριν προσθέσουμε έναν εργαζόμενο σε κάποιο event, θέλουμε να δούμε αν τον έχουμε αναθέσει κάπου αλλού την ίδια στιγμή. Επομένως σε ένα BEFORE INSERT Trigger καλούμε το Procedure *checkStaffOverlap*.

```
DELIMITER //
CREATE TRIGGER staffInsert BEFORE INSERT ON worksIn FOR EACH ROW
BEGIN
    CALL checkStaffOverlap(NEW.staff_id, NEW.music_event_id);
END;
//
DELIMITER ;
```

event date

Αυτό το Trigger φροντίζει η ημερομηνία του event που βάζουμε να ανήκει στις ημέρες διεξαγωγής του αντίστοιχου φεστιβάλ. Αν δεν ανήκει θα σταματάει και θα τυπώνει αντίστοιχο μήνυμα. Εδώ το Trigger είναι *BEFORE INSERT*.

```

DELIMITER //
CREATE TRIGGER event_date BEFORE INSERT ON music_event FOR EACH ROW
BEGIN
    IF NOT EXISTS
        (SELECT * FROM festival WHERE festival_id = NEW.festival_id
        AND NEW.music_event_date BETWEEN festival_start_date AND festival_end_date) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Event Date Error';
    END IF;
END;
//
DELIMITER ;

```

checkPerformances

Κάθε φορά που προσθέτουμε ένα performance πρέπει να ελέγχονται τα εξής πράγματα:

- A) Είναι το διάλειμμα μεταξύ 5 και 30 λεπτών;
 - B) Χρησιμοποιείται το ίδιο stage την ίδια στιγμή;
 - Γ) Ο/οι καλλιτέχνης/καλλιτέχνες εμφανίζονται αλλού την ίδια στιγμή;
- Καθένα από αυτά ελέγχεται με αντίστοιχο Procedure που θα εξηγηθεί παρακάτω.
Επίσης με βάση την ώρα έναρξης και την διάρκεια της εμφάνισης, υπολογίζεται αυτόματα η ώρα λήξης της. Το trigger είναι *BEFORE INSERT*.

```

DELIMITER //
CREATE TRIGGER checkPerformances BEFORE INSERT ON performance FOR EACH ROW
BEGIN
    DECLARE perfEndTime BIGINT;

    SET NEW.performance_end_time = ADDTIME(NEW.performance_start_time, SEC_TO_TIME(NEW.performance_duration * 60));

    CALL checkBreak(NEW.music_event_id, NEW.performance_start_time);
    CALL checkStageOverlap(NEW.music_event_id, NEW.performance_start_time, NEW.performance_end_time);
    CALL checkArtistOverlap(NEW.performer_id, NEW.music_event_id, NEW.performance_start_time, NEW.performance_end_time);
END;
//
DELIMITER ;

```

checkArtistYears

Αφού προστεθεί το performance ελέγχουμε ότι ο καλλιτέχνης δεν παίρνει μέρος σε 3 συνεχόμενα φεστιβάλ, χρησιμοποιώντας το Procedure *checkArtistStreak*. Για να γίνει πιο εύκολος ο έλεγχος, καλούμε το procedure αφού έχουμε κάνει insert το performance. Δηλαδή, χρησιμοποιούμε εδώ *AFTER INSERT*.

```

DELIMITER //
CREATE TRIGGER checkArtistYears AFTER INSERT ON performance FOR EACH ROW
BEGIN
    CALL checkArtistStreak();
END;
//
DELIMITER ;

```

createPerformerA

Κάθε φορά που προσθέτουμε έναν καλλιτέχνη, θέλουμε να φτιάξουμε έναν performer με το ίδιο όνομα. Έτσι, με αυτό το Trigger, όταν προσθέτουμε έναν artist αυτόματα φτιάχνεται ένας performer με το ίδιο όνομα και το *performer_id* που μόλις δημιουργήθηκε επιστρέφεται και χρησιμοποιείται σαν attribute στον artist.

```

DELIMITER //
CREATE TRIGGER createPerformerA BEFORE INSERT ON artist FOR EACH ROW
BEGIN
    DECLARE idCreated BIGINT;

    INSERT INTO performer(performer_name, performer_type) VALUES (NEW.artist_name, 'A');
    SELECT performer_id INTO idCreated FROM performer WHERE performer_name = NEW.artist_name;

    SET NEW.artist_performer_id = idCreated;
END;
//
DELIMITER ;

```

createPerformerB

To ίδιο με το από πάνω, αλλά για μπάντα.

```

DELIMITER //
CREATE TRIGGER createPerformerB BEFORE INSERT ON band FOR EACH ROW
BEGIN
    DECLARE idCreated BIGINT;

    INSERT INTO performer(performer_name, performer_type) VALUES (NEW.band_name, 'B');
    SELECT performer_id INTO idCreated FROM performer WHERE performer_name = NEW.band_name;

    SET NEW.band_performer_id = idCreated;
END;
//
DELIMITER ;

```

checkTickets

Πριν προσθέσουμε ένα εισιτήριο πρέπει να εκλεχθούν τα εξής:

- A) Ο κωδικός *EAN* – 13 είναι έγκυρος.
 - B) Ο επισκέπτης δεν έχει ήδη εισιτήριο για την ίδια παράσταση.
 - Γ) Ο επισκέπτης δεν έχει εισιτήριο την ίδια ημέρα.
 - Δ) Η παράσταση δεν είναι ήδη sold-out.
 - Ε) Τα VIP εισιτήρια δεν ξεπερνάνε το 10% της συνολικής χωρητικότητας της αίθουσας.
- To trigger ελέγχει τα από πάνω με αυτήν την σειρά. Επίσης, φροντίζει ένα εισιτήριο που μόλις προστέθηκε στην βάση να μην είναι ήδη προς πώληση *FOR SALE*, αλλά να είναι είτε *USED* είτε *NOT USED*.

```

DELIMITER //
CREATE TRIGGER checkTickets BEFORE INSERT ON ticket FOR EACH ROW
BEGIN
    DECLARE eventDate DATE;
    DECLARE capacity BIGINT;

    SELECT music_event_date INTO eventDate FROM music_event WHERE NEW.music_event_id = music_event_id;
    SELECT stage_capacity INTO capacity FROM music_event NATURAL JOIN stage WHERE NEW.music_event_id = music_event_id LIMIT 1;

    CALL checkEAN(NEW.ticket_EAN_13_code);

    IF NEW.ticket_status = 'FOR SALE' THEN
        SET NEW.ticket_status = 'NOT USED';
    END IF;

    IF EXISTS (SELECT * FROM ticket WHERE visitor_id = NEW.visitor_id AND music_event_id = NEW.music_event_id) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Visitor Already Owns Ticket';
    END IF;

    IF EXISTS (SELECT * FROM ticket NATURAL JOIN music_event
               WHERE visitor_id = NEW.visitor_id AND music_event_date = eventDate) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Visitor Already Owns Ticket';
    END IF;

    IF (SELECT COUNT(*) + 1 FROM
        ticket NATURAL JOIN music_event WHERE music_event_id = NEW.music_event_id) > capacity THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Too Many Tickets';
    END IF;

    IF NEW.ticketType_type = 'VIP' THEN
        SET capacity = CEILING(capacity * 0.1);
        IF (SELECT COUNT(*) + 1 FROM
            ticket WHERE music_event_id = NEW.music_event_id AND ticketType_type = 'VIP') > capacity THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Too Many VIP';
        END IF;
    END IF;
END;
//  

DELIMITER ;

```

validReview

Εδώ, πριν προστεθεί στην βάση κάποιο review ελέγχουμε αν ο επισκέπτης που κάνει την αξιολόγηση διαθέτει USED εισιτήριο για την παράσταση που θέλει να αξιολογήσει. Απλώς κάνουμε JOIN τους πίνακες *ticket*, *music_event*, *performance* για να βρούμε αν υπάρχει entry με τα απαιτούμενα *visitor_id*, *performance_id*.

```

DELIMITER //
CREATE TRIGGER valid_review BEFORE INSERT ON review FOR EACH ROW
BEGIN
    IF NOT EXISTS
        (SELECT * FROM ticket NATURAL JOIN music_event NATURAL JOIN performance
         WHERE visitor_id = NEW.visitor_id AND performance_id = NEW.performance_id AND ticket_status = 'USED') THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Review Error';
    END IF;
END;
//  

DELIMITER ;

```

validSeller

Πριν το insert κάποιου seller ελέγχονται τα εξής:

A) Αν η παράσταση έχει γίνει sold-out. Αν δεν έχει γίνει, δε μπορεί να μεταπουληθεί ακόμα το συγκεκριμένο εισιτήριο και εμφανίζεται σχετικό μήνυμα.

B) Ο επισκέπτης διαθέτει πράγματι το εισιτήριο που πουλάει και είναι NOT USED.

Για συντομία στον κώδικα στην μεταβλητή eventId αποθηκεύεται το music_event_id που αντιστοιχεί στο συγκεκριμένο εισιτήριο.

Τέλος, εφόσον ισχύουν τα παραπάνω καλείται το Procedure *findBuyer* το οποίο επιστρέφει μέσω της μεταβλητής @result 1 αν βρέθηκε αγοραστής, 0 διαφορετικά. Το πώς λειτουργεί το συγκεκριμένο Procedure εξηγείται στη συνέχεια.

```

DELIMITER //
CREATE TRIGGER valid_seller BEFORE INSERT ON seller FOR EACH ROW
BEGIN
    DECLARE eventId BIGINT;
    SELECT music_event_id INTO eventId
        FROM music_event NATURAL JOIN ticket WHERE NEW.ticket_id = ticket_id;

    CALL checkSoldOut(eventId, @soldout);
    IF @soldout = FALSE THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tickets Not Sold Out';
    END IF;

    IF NOT EXISTS
        (SELECT * FROM ticket WHERE ticket_id = NEW.ticket_id AND visitor_id = NEW.visitor_id AND ticket_status = 'NOT USED') THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Seller Error';
    END IF;

    SET NEW.seller_sold = 'N';

    CALL findBuyer(NEW.ticket_id, @result);

    IF @result = 1 THEN
        SET NEW.seller_sold = 'Y';
    END IF;
END;
//
```

insertOnBuyer

Πάλι ελέγχουμε αν η παράσταση έχει γίνει sold-out μέσω του Procedure *checkSoldOut*.

Κάποιος που ενδιαφέρεται για μία παράσταση, πρέπει να δηλώσει υποχρεωτικά το ποια παράσταση είναι αυτή και τον τύπο εισιτηρίου που τον ενδιαφέρει. Ωστόσο, προαιρετικά, μπορεί να δώσει και ένα συγκεκριμένο ticket_id. Επομένως, οφείλουμε να ελέγχουμε ότι το εισιτήριο αυτό αναφέρεται πράγματι στην παράσταση και είναι του ίδιου τύπου με αυτό που έχει δηλώσει.

Έπειτα, αν ο επισκέπτης ενδιαφέρεται για συγκεκριμένο εισιτήριο, ελέγχουμε αν αυτό είναι προς πώληση (δηλαδή βρίσκεται στον πίνακα resaleQueue).

Τέλος, μέσω του Procedure *findTicket* βλέπουμε αν υπάρχει εισιτήριο που ικανοποιεί τα ζητούμενα κριτήρια. Αν υπάρχει, η αγορά γίνεται αυτόματα (αφού ο επισκέπτης δεν έχει ήδη εισιτήριο για την συγκεκριμένη παράσταση). Το *findTicket*, όπως και όλα τα υπόλοιπα Procedures, θα παρουσιαστούν αναλυτικά στην συνέχεια.

```

DELIMITER //
CREATE TRIGGER insertOnBuyer BEFORE INSERT ON buyer FOR EACH ROW
BEGIN
    DECLARE eventId BIGINT;
    SELECT music_event_id INTO eventId FROM music_event WHERE music_event_id = NEW.music_event_id;

    CALL checkSoldOut(eventId, @soldout);
    IF @soldout = FALSE THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tickets Not Sold Out';
    END IF;

    IF (NEW.ticket_id IS NOT NULL) AND NOT EXISTS (SELECT * FROM ticket
        WHERE NEW.ticket_id = ticket_id AND NEW.music_event_id = music_event_id AND NEW.ticketType_type = ticketType_type) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Ticket Information Incorrect';
    END IF;

    IF NEW.ticket_id IS NOT NULL AND
        NOT EXISTS (SELECT * FROM resaleQueue WHERE ticket_id = NEW.ticket_id) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Ticket Not Available for Resale';
    END IF;

    SET NEW.buyer_timeInserted = CURRENT_TIMESTAMP;
    SET NEW.buyer_sold = 'N';

    CALL findTicket(NEW.visitor_id, NEW.music_event_id, NEW.ticketType_type, @result);

    IF @result = 1 THEN
        IF (SELECT COUNT(*) FROM ticket WHERE visitor_id = NEW.visitor_id AND music_event_id = NEW.music_event_id) > 1 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Already Owns Ticket';
        END IF;

        SET NEW.buyer_sold = 'Y';
    END IF;
END;
//  

DELIMITER ;

```

Procedures

Ακολουθούν τα Procedures που ορίσαμε:

checkEnoughStaff

Δέχεται σαν είσοδο το *music_event_id* και ελέγχει αν το προσωπικό που έχει ανατεθεί μέχρι στιγμής πληροί τα ακόλουθα κριτήρια:

Το προσωπικό ασφαλείας είναι τουλάχιστον το 5% της χωρητικότητας της σκηνής.

Το βοηθητικό προσωπικό είναι τουλάχιστον το 2% της χωρητικότητας της σκηνής.

Λειτουργεί με τον εξής τρόπο:

Πρώτα αποθηκεύεται στην μεταβλητή *capacity* η χωρητικότητα της σκηνής. Για να την βρούμε απλώς κάνουμε *NATURAL JOIN* τους πίνακες *music_event*, *stage* και κρατάμε το *entry* με το *music_event_id* που μας ενδιαφέρει. Έπειτα, υπολογίζουμε το *minSecurity*, *minSupport* που είναι το ελάχιστο προσωπικό ασφαλείας και βοηθητικό προσωπικό που πρέπει να έχει ανατεθεί. Τέλος, στις μεταβλητές *currentSecurity*, *currentSupport* βάζουμε το πλήθος του αντίστοιχου προσωπικού που έχουμε ήδη αναθέσει. Αν είναι λιγότερο από το ελάχιστο, εμφανίζεται σχετικό μήνυμα.

```
DELIMITER //
CREATE PROCEDURE checkEnoughStaff(eventId BIGINT)
BEGIN
    DECLARE capacity BIGINT;
    DECLARE minSecurity, minSupport BIGINT;

    DECLARE currentSecurity, currentSupport BIGINT;
    DECLARE staffType VARCHAR(20);

    SELECT stage_capacity INTO capacity FROM music_event NATURAL JOIN stage WHERE music_event_id = eventId;
    SET minSecurity = CEILING(0.05 * capacity);
    SET minSupport = CEILING(0.02 * capacity);

    SELECT COUNT(*) INTO currentSecurity
        FROM worksIn NATURAL JOIN staff WHERE eventId = music_event_id AND role_staff_desc = 'Security';
    SELECT COUNT(*) INTO currentSupport
        FROM worksIn NATURAL JOIN staff WHERE eventId = music_event_id AND role_staff_desc = 'Support';

    IF currentSecurity < minSecurity THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Not Enough Security';
    END IF;

    IF currentSupport < minSupport THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Not Enough Support Workers';
    END IF;
END;
//
```

checkStaffOverlap

Καλείται κάθε φορά που κάνουμε insert στον πίνακα *worksIn*, δηλαδή μόλις αναθέτουμε κάποιον από το προσωπικό σε κάποιο event. Αν τον έχουμε αναθέσει να δουλεύει την ίδια στιγμή κάπου αλλού εμφανίζεται μήνυμα σφάλματος.

Δέχεται σαν είσοδο, το staff_id και το εκάστοτε event. Πρώτα αποθηκεύει στις μεταβλητές *eventDate*, *eventStart*, *eventEnd* την ημερομηνία και την ώρα έναρξης και λήξης του event. Έπειτα, στον προσωρινό πίνακα *SameDayStaff* αποθηκεύονται όλα τα υπόλοιπα event της ίδιας μέρας στα οποία εργάζεται ο εργαζόμενος που προσθέτουμε. Αν κάποιο από αυτά διαδραματίζεται την ίδια στιγμή με το τρέχον event, δημιουργείται πρόβλημα.

```
DELIMITER //
CREATE PROCEDURE checkStaffOverlap(staffId BIGINT, eventId BIGINT)
BEGIN
    DECLARE eventDate DATE;
    DECLARE eventStart TIME;
    DECLARE eventEnd TIME;

    SELECT music_event_date INTO eventDate
        FROM music_event WHERE music_event_id = eventId;
    SELECT music_event_time INTO eventStart
        FROM music_event WHERE music_event_id = eventId;
    SELECT music_event_end_time INTO eventEnd
        FROM music_event WHERE music_event_id = eventId;

    DROP TEMPORARY TABLE IF EXISTS SameDayStaff;
    CREATE TEMPORARY TABLE SameDayStaff AS
    SELECT music_event_time, music_event_end_time FROM
        worksIn NATURAL JOIN music_event WHERE staff_id = staffId AND music_event_date = eventDate;

    IF EXISTS
        (SELECT * FROM SameDayStaff WHERE
            NOT(eventStart > SameDayStaff.music_event_end_time OR eventEnd < SameDayStaff.music_event_time)) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Staff working at the same time';
    END IF;
END;
//
```

checkBreak

Δέχεται σαν είσοδο το event στο οποίο θέλουμε να προσθέσουμε μία παράσταση και την ώρα έναρξης της παράστασης. Στην μεταβλητή *previousTime* αποθηκεύεται η ώρα λήξης της προηγούμενης παράστασης του ίδιου event. Αν δεν υπάρχει, σημαίνει ότι αυτή είναι η πρώτη παράσταση και άρα πρέπει να ξεκινάει ακριβώς στην ώρα έναρξης του event (μεταβλητή *eventTime*). Άλλως, στην μεταβλητή *break* αποθηκεύεται σε δευτερόλεπτα η διαφορά ώρας μεταξύ της έναρξης της τρέχουσας παράστασης και της λήξης της προηγούμενης. Αν το *break* δεν είναι μεταξύ των ορίων που θέλουμε, εμφανίζεται σχετικό μήνυμα.

```

DELIMITER //
CREATE PROCEDURE checkBreak(event_id BIGINT, perfStart TIME)
BEGIN
    DECLARE previousTime TIME;
    DECLARE eventTime TIME;
    DECLARE break BIGINT;

    SELECT performance_end_time INTO previousTime
        FROM performance NATURAL JOIN music_event WHERE music_event_id = event_id ORDER BY performance_end_time DESC LIMIT 1;

    SELECT music_event_time INTO eventTime
        FROM music_event WHERE music_event_id = event_id;

    IF previousTime IS NULL THEN
        IF perfStart != eventTime THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'First Performance Must Begin At the Start of the Event';
        END IF;
    ELSE
        SET break = TIME_TO_SEC(TIMEDIFF(perfStart, previousTime));

        IF break < 5 * 60 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Break Too Short';
        END IF;
        IF break > 30 * 60 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Break Too Long';
        END IF;
    END IF;
END;

```

checkStageOverlap

Δέχεται σαν είσοδο το event και την ώρα έναρξης και λήξης της παράστασης που μας ενδιαφέρει. Στις μεταβλητές stageId και eventDate αποθηκεύεται η σκηνή και η ημερομηνία διεξαγωγής του event. Μετά, δημιουργούμε έναν προσωρινό πίνακα SameDayPerformances στον οποίο αποθηκεύουμε όλες τις παραστάσεις οι οποίες διαδραματίζονται στην συγκεκριμένη σκηνή την ημερομηνία που μας ενδιαφέρει. Αν οποιοδήποτε entry αυτού του πίνακα έχει παράσταση, η οποία βρίσκεται εντός του χρονικού πλαισίου της παράστασης που πάμε να εισαγάγουμε, δημιουργείται πρόβλημα.

```

DELIMITER //
CREATE PROCEDURE checkStageOverlap(event_id BIGINT, perfStartTime TIME, perfEndTime TIME)
BEGIN
    DECLARE stageId BIGINT;
    DECLARE eventDate DATE;

    SELECT stage_id INTO stageId
        FROM music_event WHERE music_event_id = event_id;
    SELECT music_event_date INTO eventDate
        FROM music_event WHERE music_event_id = event_id;

    DROP TEMPORARY TABLE IF EXISTS SameDayPerformances;
    CREATE TEMPORARY TABLE SameDayPerformances AS
        SELECT * FROM performance NATURAL JOIN music_event
        WHERE stage_id = stageId AND music_event_date = eventDate;

    IF EXISTS
        (SELECT * FROM SameDayPerformances WHERE
            NOT(perfStartTime > SameDayPerformances.performance_end_time OR perfEndTime < SameDayPerformances.performance_start_time)) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stage is in use';
    END IF;
END;
//
```

checkArtistOverlap

Λειτουργία παρόμοια με το προηγούμενο Procedure, αλλά αυτήν τη φορά ελέγχει αν υπάρχει επικάλυψη στους καλλιτέχνες. Πρώτα, δημιουργούμε τον προσωρινό πίνακα *CurrentPerformers* στον οποίο αποθηκεύονται τα *artist_id* των καλλιτεχνών που συμμετέχουν στην παράσταση. Αν τραγουδάει κάποιος solo, αποθηκεύεται το δικό του id, αλλιώς το id όλων των μελών της μπάντας. Για να το υπολογίσουμε αυτό, απλά κοιτάμε στους πίνακες *artist*, *band* το κατάλληλο *performer_id* και στην περίπτωση της μπάντας, κάνουμε *NATURAL JOIN* με *members*.

Έπειτα, υπολογίζεται ο πίνακας *SameDayPerformers*. Βρίσκουμε όλα τα performances που συμβαίνουν την ίδια ημέρα και βρίσκουμε τους καλλιτέχνες που συμμετέχουν σε καθένα από αυτά. Έπειτα, σβήνουμε αυτούς που δεν συμμετέχουν στην παράσταση που θέλουμε να εισάγουμε τώρα.

Επομένως, μας έχουν μείνει μόνο οι καλλιτέχνες οι οποίοι εμφανίζονται την ίδια ημέρα και συμμετέχουν στην καινούρια παράσταση που θα εισαγάγουμε. Το μόνο που μένει να κάνουμε είναι να δούμε ότι για καθέναν από αυτούς τα χρονικά διαστήματα δεν αλληλοκαλύπτονται.

```
DELIMITER //
CREATE PROCEDURE checkArtistOverlap(performerId BIGINT, event_id BIGINT, perfStartTime TIME, perfEndTime TIME)
BEGIN
    DECLARE eventDate DATE;
    DECLARE eventYear YEAR;

    SELECT music_event_date INTO eventDate
        FROM music_event WHERE music_event_id = event_id;
    SET eventYear = YEAR(eventDate);

    DROP TEMPORARY TABLE IF EXISTS CurrentPerformers;
    CREATE TEMPORARY TABLE CurrentPerformers AS
    WITH
        A AS (SELECT artist_id FROM artist WHERE artist_performer_id = performerId),
        B AS (SELECT artist_id FROM band NATURAL JOIN members WHERE band_performer_id = performerId)
    (SELECT * FROM A UNION (SELECT * FROM B));

    DROP TEMPORARY TABLE IF EXISTS SameDayPerformers;
    CREATE TEMPORARY TABLE SameDayPerformers AS
    WITH
        A AS (
            SELECT *
            FROM music_event
            NATURAL JOIN performance
            NATURAL JOIN (performer JOIN artist ON artist_performer_id = performer_id)
            WHERE music_event_date = eventDate),
        B AS (
            SELECT *
            FROM music_event
            NATURAL JOIN performance
            NATURAL JOIN (performer JOIN band ON band_performer_id = performer_id)
            NATURAL JOIN members
            NATURAL JOIN artist
            WHERE music_event_date = eventDate)
    (SELECT artist_id, performance_start_time, performance_end_time FROM A)
    UNION (SELECT artist_id, performance_start_time, performance_end_time FROM B);
```

```

DELETE FROM SameDayPerformers WHERE artist_id <> ALL(SELECT * FROM CurrentPerformers);
IF EXISTS
    (SELECT * FROM SameDayPerformers WHERE
        NOT(perfStartTime > SameDayPerformers.performance_end_time
        OR perfEndTime < SameDayPerformers.performance_start_time))
THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Artist performing at the same time';
END IF;
END;
//
```

checkArtistStreak

Ελέγχουμε ότι κανένας καλλιτέχνης δεν εμφανίζεται σε 3 συνεχόμενα φεστιβάλ. Πρώτα πηγαίνουμε σε όλα τα performances και βάζουμε στον πίνακα *YearsPerformed* όλους τους καλλιτέχνες που έχουν συμμετάσχει ποτέ σε κάποιο φεστιβάλ. Μετά για κάθε artist ταξινομούμε τις εμφανίσεις του ανά έτος και τις αριθμούμε (στήλη *rowNum*). Αν έχει εμφανιστεί συνεχόμενες φορές ξέρουμε ότι η διαφορά (*eventYear - rowNum*) θα είναι σταθερή για κάθε καλλιτέχνη. Επομένως, πηγαίνουμε για κάθε artist και βλέπουμε πόσες φορές στον καθένα εμφανίζεται η ίδια τιμή της διαφοράς (*eventYear - rowNum*). Αν εμφανίζεται πάνω από 3 φορές, ξέρουμε ότι εμφανίζεται σε παραπάνω από 3 συνεχόμενα φεστιβάλ και τυπώνουμε σχετικό μήνυμα.

```

DELIMITER //
CREATE PROCEDURE checkArtistStreak()
BEGIN
    DROP TEMPORARY TABLE IF EXISTS YearsPerformed;
    CREATE TEMPORARY TABLE YearsPerformed AS
    WITH
        A AS (SELECT * FROM music_event NATURAL JOIN (performance JOIN artist ON artist.performer_id = performance.performer_id)),
        B AS (SELECT * FROM music_event NATURAL JOIN (performance JOIN band ON band.performer_id = performance.performer_id) NATURAL JOIN members)
    (SELECT artist_id, YEAR(music_event_date) AS eventYear FROM A) UNION (SELECT artist_id, YEAR(music_event_date) AS eventYear FROM B);

    DROP TEMPORARY TABLE IF EXISTS YearsPerformedNumbered;
    CREATE TEMPORARY TABLE YearsPerformedNumbered AS
    SELECT artist_id, eventYear, ROW_NUMBER() OVER (PARTITION BY artist_id ORDER BY eventYear) AS rowNum
    FROM YearsPerformed;

    IF EXISTS (SELECT artist_id, eventYear - rowNum FROM YearsPerformedNumbered GROUP BY artist_id, eventYear - rowNum HAVING COUNT(*) > 3) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Artist performing for more than 3 years in a row';
    END IF;
END;
//
```

checkEAN

Παίρνουμε σαν είσοδο τον κωδικό *EAN* – 13 ενός εισιτηρίου και αν αυτός δεν είναι έγκυρος εμφανίζεται μήνυμα σφάλματος. Πρώτα, χωρίζουμε τον αριθμό στα ψηφία του και τα αποθηκεύουμε στον πίνακα *Digits*. Στην μεταβλητή *odd* αποθηκεύουμε το άθροισμα των ψηφίων που βρίσκονται στις θέσεις 1,3,...,11 και στην *even* των ψηφίων στις θέσεις 2,4,...,12. Τέλος, ελέγχουμε αν το 13^ο ψηφίο είναι σωστό με βάση το πρότυπο *EAN* – 13, χρησιμοποιώντας τις μεταβλητές *odd* και *even*.

```

DELIMITER //
CREATE PROCEDURE checkEAN(ticket_code BIGINT)
BEGIN
    DECLARE odd BIGINT;
    DECLARE even BIGINT;
    DECLARE errorDigit BIGINT;

    DROP TEMPORARY TABLE IF EXISTS Digits;
    CREATE TEMPORARY TABLE Digits AS
        SELECT num mod POWER(10, 0) div POWER(10, 0) AS d0 ,
               num mod POWER(10, 2) div POWER(10, 1) AS d1 ,
               num mod POWER(10, 3) div POWER(10, 2) AS d2 ,
               num mod POWER(10, 4) div POWER(10, 3) AS d3 ,
               num mod POWER(10, 5) div POWER(10, 4) AS d4 ,
               num mod POWER(10, 6) div POWER(10, 5) AS d5 ,
               num mod POWER(10, 7) div POWER(10, 6) AS d6 ,
               num mod POWER(10, 8) div POWER(10, 7) AS d7 ,
               num mod POWER(10, 9) div POWER(10, 8) AS d8 ,
               num mod POWER(10, 10) div POWER(10, 9) AS d9 ,
               num mod POWER(10, 11) div POWER(10, 10) AS d10,
               num mod POWER(10, 12) div POWER(10, 11) AS d11,
               num mod POWER(10, 13) div POWER(10, 12) AS d12
        FROM (SELECT ticket_code AS num) AS Digits;

        SELECT (d12 + d10 + d8 + d6 + d4 + d2) INTO odd FROM Digits;
        SELECT (d11 + d9 + d7 + d5 + d3 + d1) INTO even FROM Digits;

        SELECT (10 - ((odd + 3 * even) % 10)) % 10 INTO errorDigit FROM Digits;

        IF errorDigit != (SELECT d0 FROM Digits LIMIT 1) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong EAN Code';
        END IF;
    END;
    //
DELIMITER ;

```

checkSoldOut

Απλώς ελέγχουμε αν το event που δίνεται σαν είσοδο είναι sold-out. Δηλαδή, αν υπάρχουν τόσα εισιτήρια (μεταβλητή *soldTickets*) με αυτό το *music_event_id* όσο είναι το αντίστοιχο *stage_capacity* (μεταβλητή *capacity*). Αν είναι sold-out, επιστρέφουμε σαν έξοδο μέσω της μεταβλητής *isSoldOut* *TRUE*, αλλιώς επιστρέφουμε *FALSE*.

```

DELIMITER //
CREATE PROCEDURE checkSoldOut(eventId BIGINT, OUT isSoldOut BOOL)
BEGIN
    DECLARE capacity BIGINT;
    DECLARE soldTickets BIGINT;

    SELECT stage_capacity INTO capacity FROM music_event NATURAL JOIN stage WHERE eventId = music_event_id LIMIT 1;
    SELECT COUNT(*) INTO soldTickets FROM ticket NATURAL JOIN music_event WHERE music_event_id = eventId;

    SET isSoldOut = FALSE;
    IF soldTickets = capacity THEN
        SET isSoldOut = TRUE;
    END IF;
END;
//
DELIMITER ;

```

findBuyer

Καλείται κάθε φορά που προσθέτουμε έναν seller. Απλώς πηγαίνει στον πίνακα buyer και ψάχνει να βρει κάποιον αγοραστή που ενδιαφέρεται για τα χαρακτηριστικά του συγκεκριμένου εισιτηρίου (μεταβλητή *potentialBuyer*). Τηρείται σειρά προτεραιότητας. Αν βρεθεί αγοραστής, γίνεται update στο εισιτήριο το *visitor_id* και το αντίστοιχο entry στον πίνακα buyer σημειώνεται ως sold. Άλλιώς, το εισιτήριο προστίθεται στην *resaleQueue*. Τέλος, στην μεταβλητή result αποθηκεύουμε αν πραγματοποιήθηκε αγορά ή όχι.

```
DELIMITER //
CREATE PROCEDURE findBuyer(ticket BIGINT, OUT result BIGINT)
BEGIN
    DECLARE event_id BIGINT;
    DECLARE tckt_type VARCHAR(20);
    DECLARE potentialBuyer BIGINT;

    SELECT music_event_id, ticketType_type INTO event_id, tckt_type
        FROM ticket NATURAL JOIN music_event WHERE ticket_id = ticket LIMIT 1;

    SELECT visitor_id INTO potentialBuyer
        FROM buyer WHERE music_event_id = event_id AND ticketType_type = tckt_type AND buyer_sold = 'N' ORDER BY buyer_timeInserted LIMIT 1;

    IF potentialBuyer IS NOT NULL THEN
        UPDATE ticket SET visitor_id = potentialBuyer, ticket_status = 'NOT USED' WHERE ticket_id = ticket;
        UPDATE buyer SET buyer_sold = 'Y' WHERE visitor_id = potentialBuyer;
        SET result = 1;
    ELSE
        UPDATE ticket SET ticket_status = 'FOR SALE' WHERE ticket_id = ticket;
        INSERT INTO resaleQueue(ticket_id, resaleQueue_timeInserted) VALUES (ticket, CURRENT_TIMESTAMP);
        SET result = 0;
    END IF;

END;
//
DELIMITER ;
```

findTicket

Για κάθε buyer που κάνουμε insert, ψάχνουμε στον πίνακα *resaleQueue* να βρούμε αν υπάρχει εισιτήριο με τα χαρακτηριστικά που ψάχνουμε (μεταβλητή *potentialTicket*). Αν βρέθηκε εισιτήριο, κάνουμε update το *visitor_id*, αλλιώς δεν κάνουμε τίποτα. Πάλι, στην μεταβλητή *result* αποθηκεύουμε αν πραγματοποιήθηκε η αγορά ή όχι.

```
DELIMITER //
CREATE PROCEDURE findTicket(buyer BIGINT, event_id BIGINT, tckt_type VARCHAR(20), OUT result BIGINT)
BEGIN
    DECLARE potentialTicket BIGINT;

    SELECT ticket_id INTO potentialTicket
        FROM resaleQueue NATURAL JOIN ticket NATURAL JOIN music_event
        WHERE music_event_id = event_id AND ticketType_type = tckt_type ORDER BY resaleQueue_timeInserted LIMIT 1;

    IF potentialTicket IS NOT NULL THEN
        UPDATE ticket SET visitor_id = buyer, ticket_status = 'NOT USED' WHERE ticket_id = potentialTicket;
        DELETE FROM resaleQueue WHERE ticket_id = potentialTicket;
        UPDATE seller SET seller_sold = 'Y' WHERE ticket_id = potentialTicket;
        SET result = 1;
    ELSE
        SET result = 0;
    END IF;

END;
//
DELIMITER ;
```

Data

Για τη λειτουργία της βάσης δεδομένων προσθέτουμε ενδεικτικά δεδομένα σε κάθε πίνακα. Ξεκινάμε από τους πίνακες που δεν ενώνονται με foreign key με άλλον πίνακα που δεν έχει ακόμα δεδομένα.

Αναλυτικά για τους προαναλυθέντες πίνακες έχουμε:

- *equipmentCategory*

Περιέχει 10 διαφορετικές κατηγορίες εξοπλισμού.

- *equipment*

Περιλαμβάνει 28 τεμάχια εξοπλισμού.

- *staff*

Τα φεστιβάλ αξιοποιούν 312 άτομα για προσωπικό, εκ των οποίων 178 είναι security, 88 είναι support και 46 technical.

- *location*

Οι τοποθεσίες που πραγματοποιούνται τα event είναι 10, όσες και τα φεστιβάλ.

- *festival*

Η ιστοσελίδα θα διοργανώσει 10 φεστιβάλ από το 2020 έως το 2029, εκ των οποίων 5 έχουν ήδη συμβεί και 5 είναι μελλοντικά (Το φεστιβάλ του 2025 θεωρούμε ότι δεν έχει πραγματοποιηθεί αφού είναι προγραμματισμένο για τον Νοέμβριο).

- *genre*

Στην παρούσα υλοποίηση υποθέσαμε ότι υπάρχουν 11 είδη μουσικής.

- *subgenre*

Η υπόθεση μας συνεχίζεται με το ότι υπάρχουν 52 υποείδη μουσικής.

- *stage*

Τα φεστιβάλ μπορούν να πραγματοποιηθούν σε 31 διαφορετικά stages.

- *music_event*

Διοργανώσαμε για το Pulse University 129 events.

- *performer*

Συνολικά υπάρχουν 162 performers που αποτελούν το άθροισμα των artists και των bands.

- *artist*

Στη διάθεση των διοργανωτών είναι 137 καλλιτέχνες.

➤ *art2genre*

Σχέσεις καλλιτεχνών και ειδών μουσικής είναι σε πλήθος 151, που ξεπερνά το πλήθος των καλλιτεχνών, καθώς κάθε καλλιτέχνης ανήκει σε ένα είδος και κάποιοι ανήκουν σε περισσότερα από ένα είδη μουσικής.

➤ *art2subgenre*

Το πλήθος των σχέσεων καλλιτέχνη με υποείδος μουσικής είναι 35, καθώς δεν θεωρήσαμε υποχρεωτικό κάθε καλλιτέχνης να ανήκει σε κάποιο υποείδος.

➤ *band*

Η βάση μας περιλαμβάνει 25 μπάντες.

➤ *band2genre*

Υπάρχουν 25 προσθήκες δεδομένων για το είδος της μουσικής στις μπάντες.

➤ *band2subgenre*

Πιο συγκεκριμένα έχουμε εισάγει και 12 σχέσεις μπαντών με υποείδη μουσικής.

➤ *members*

Συνολικά οι μπάντες έχουν 73 μέλη.

➤ *equimentUsed*

Οι σχέσεις για τον εξοπλισμό για τα stages είναι 145.

➤ *performance*

Σε όλη τη διάρκεια του festival έχουν συμβεί και θα συμβούν 178 performances. 51 εξ αυτών είναι Headline, 65 Warm-ups και 62 Special Guests.

➤ *worksIn*

Για να καλυφθούν πλήρως οι απαιτήσεις για το προσωπικό σε κάθε event χρειαστήκαμε 13156 σχέσεις.

➤ *visitor*

Οι επισκέπτες των φεστιβάλ είναι 600.

➤ *ticket*

Για όλα τα φεστιβάλ που λήθηκαν 3997 εισιτήρια, εκ των οποίων 1297 είναι regular, 1295 είναι student, backstage είναι 1403 και 2 VIP.

➤ *review*

Τέλος έχουν πραγματοποιηθεί 250 reviews, μόνο για performances που έχουν συμβεί.

Παραδείγματα κώδικα για δημιουργία Data

Για την δημιουργία των μοναδικών κωδικών EAN_13 χρησιμοποιήθηκε η εξής συνάρτηση στην python.

```
import random

def generate_ean13():
    digits = [random.randint(0, 9) for _ in range(12)]

    sum_odd = sum(digits[i] for i in range(0, 12, 2))      # positions 1,3,5,... (index 0,2,4,...)
    sum_even = sum(digits[i] for i in range(1, 12, 2))     # positions 2,4,6,... (index 1,3,5,...)
    check_digit = (10 - ((sum_odd + 3 * sum_even) % 10)) % 10

    digits.append(check_digit)

    return ''.join(map(str, digits))

print("Generated EAN-13:", generate_ean13())
```

Και χρησιμοποιήθηκε στον παρακάτω κώδικα παραγωγής εισιτηρίων.

```
import random

def generate_ean13():
    digits = [random.randint(0, 9) for _ in range(12)]
    sum_odd = sum(digits[i] for i in range(0, 12, 2))
    sum_even = sum(digits[i] for i in range(1, 12, 2))
    check_digit = (10 - ((sum_odd + 3 * sum_even) % 10)) % 10
    digits.append(check_digit)
    return ''.join(map(str, digits))

ticket_types = ['REGULAR', 'STUDENT', 'BACKSTAGE']
payment_methods = ['DEBIT', 'CREDIT', 'BANK_DEPOSIT']
purchase_date = '2004-01-01'
status = 'USED'
#status=['USED','FOR SALE']
vip_count = 0
vip_limit = 5

with open("insert_tickets1.sql", "w") as file:
    file.write("INSERT INTO ticket (music_event_id, visitor_id, ticketType_type, ticket_purchase_date, ticket_price, ticket_payment_method, ticket_EAN_13_code, ticket_status) VALUES\n")

    values = []
    for event_id in range(35, 130): # 1 to 33 inclusive
        for visitor_id in range(1, 21): # 1 to 200 inclusive
            if vip_count < vip_limit and random.random() < 0.0008:
                ticket_type = 'VIP'
                vip_count += 1
            else:
                ticket_type = random.choice(ticket_types)

            ticket_price = round(random.uniform(20, 200), 2)
            payment_method = random.choice(payment_methods)
            ean13 = generate_ean13()
            if visitor_id < 11:
                status = 'NOT USED'
            else:
                status = 'FOR SALE'

            values.append(f"({event_id}, {visitor_id}, '{ticket_type}', '{purchase_date}', {ticket_price:.2f}, '{payment_method}', '{ean13}', '{status}')")

    file.write("\n".join(values) + ");\n")
```

Για την δημιουργία tuples με συγκεκριμένο εύρος αριθμών παρέχεται ενδεικτικά ο κώδικας του *worksIn*

```
# Generate all (number1, number2) pairs
pairs = []

# Generate and write (number1, number2) pairs grouped by number2 per line
with open("number_pairs_grouped_by_number.txt", "w") as f:
    for number2 in range(3, 129, 3):
        line = ", ".join(f"{{number1}}, {{number2}}" for number1 in range(213, 312))
        f.write(line + ",\n")
```

Για τις σχέσεις ειδών και υποειδών μουσικής χρησιμοποιήθηκαν κώδικες της λογικής

```
import random

genres = [
    'Classical',
    'Blues',
    'Country',
    'Electronic',
    'Folk',
    'Hip-Hop',
    'Jazz',
    'Pop',
    'Rock',
    'R&B',
    'EDM'
]

pairs = []

for number in range(1, 138):
    word = random.choice(genres)
    pairs.append((number, word))

with open("number_genre_pairs2.txt", "w") as f:
    for number, word in pairs:
        f.write(f"{number}, {word}\n")
```

Για τη δημιουργία ονομάτων για παράδειγμα για τους visitors χρησιμοποιήθηκαν κώδικες της λογικής:

```
import random

norwegian_first_names = [
    'Lars', 'Ola', 'Knut', 'Per', 'Jan', 'Anders', 'Nils', 'Svein', 'Bjørn', 'Erik',
    'Morten', 'Trond', 'Steinar', 'Rune', 'Geir', 'Harald', 'Einar', 'Leif', 'Tore', 'Vidar'
]
norwegian_surnames = [
    'Hansen', 'Johansen', 'Olsen', 'Larsen', 'Andersen', 'Pedersen', 'Nilsen', 'Kristiansen',
    'Jensen', 'Karlsen', 'Johnsen', 'Pettersen', 'Eriksen', 'Berg', 'Haugen', 'Dahl', 'Moen',
    'Solberg', 'Halvorsen', 'Lie'
]

norwegian_values = []
for i in range(200):
    first = random.choice(norwegian_first_names)
    last = random.choice(norwegian_surnames)
    email = f"{first.lower()}.{last.lower()}@example.com"
    age = random.randint(18, 65)
    norwegian_values.append(f"('{first}', '{last}', '{email}', {age})")

norwegian_sql_insert = "INSERT INTO visitor (visitor_name, visitor_surname, visitor_contact_info, visitor_age) VALUES\n"
norwegian_sql_insert += "\n".join(norwegian_values) + ";"
```

```
norwegian_file_path = "insert_norwegian_visitors.sql"
with open(norwegian_file_path, "w", encoding="utf-8") as f:
    f.write(norwegian_sql_insert)
```

Demo

Για την επίδειξη της εφαρμογής δημιουργήσαμε ένα αρχείο *demo.sql* στο οποίο θα κάνουμε insert κάποια απλά ενδεικτικά δεδομένα για να φανούν τα constraint που έχουμε υλοποιήσει και η λειτουργεία της ουράς μεταπώλησης.

Για τις ανάγκες του demo φτιάξαμε τα εξής δεδομένα:

```
INSERT INTO location VALUES (999999, 'Demo Location', "a", "a", "a", "Europe");
INSERT INTO stage VALUES (999998, "Demo1", "Demo Stage", 10);
INSERT INTO stage VALUES (999999, "Demo2", "Demo Stage", 5);
```

Ημερομηνίες Διεξαγωγής του Φεστιβάλ

Στην παρακάτω γραμμή το λάθος είναι ότι η ημερομηνία λήξης του φεστιβάλ δεν βρίσκεται στο έτος διεξαγωγής του.

```
# Wrong Dates
INSERT INTO festival VALUES (999999, 'Demo', '2050', 999999, '2050-5-14', '2051-5-17', NULL, NULL);
Error Code: 4025. CONSTRAINT `festival_year` failed for `music_festival_ntua`.`festival`
```

Ημερομηνία Διεξαγωγής Event

Το event που θέλουμε να προσθέσουμε διεξάγεται μία μέρα πριν ξεκινήσει το φεστιβάλ, επομένως εμφανίζεται μήνυμα σφάλματος.

```
# Date not included in festival
INSERT INTO music_event VALUES (999999, 999999, 999999, '2050-5-13', '10:00:00', NULL);
Error Code: 1644. Event Date Error
```

Έναρξη Performance

Έχουμε φτιάξει τα εξής events:

```
INSERT INTO music_event VALUES (999998, 999999, 999998, '2050-5-14', '10:00:00', NULL);
INSERT INTO music_event VALUES (999999, 999999, 999999, '2050-5-14', '10:30:00', NULL);
Θέλουμε να εισαγάγουμε το performance. Είναι η 1η παράσταση του event, επομένως η ώρα που ξεκινάει πρέπει να είναι 10:00.
```

```
# Wrong Starting Time
INSERT INTO performance VALUES (999997, 999998, 1, 'Warm-Up', '10:10:00', 100, NULL);
Error Code: 1644. First Performance Must Begin At the Start of the Event
```

Διάρκεια Performance

Ένα άλλο λάθος, θα ήταν η παράσταση να διαρκεί πάνω από 3 ώρες:

```
# Too Long Performance
INSERT INTO performance VALUES (999997, 999998, 1, 'Warm-Up', '10:00:00', 181, NULL);
Error Code: 4025. CONSTRAINT `performance_performance_duration` failed for `music_festival_ntua`.`performan...
```

Διάρκεια Διαλείμματος

Έστω ότι προσθέσαμε το παραπάνω performance. Πάμε τώρα να εισαγάγουμε την επόμενη εμφάνιση που ξεκινάει στις 11:01. Όμως, αφού η προηγούμενη τελειώνει στις 11:00, θα υπήρχε διάλειμμα μόνο ενός λεπτού, δηλαδή πολύ μικρό.

```
# Short Break
INSERT INTO performance VALUES (999998, 999998, 10, 'Warm-Up', '11:01:00', 60, NULL);
```

Error Code: 1644. Break Too Short

Καλλιτέχνης που Εμφανίζεται Δύο Φορές Ταυτόχρονα

Έστω εισάγουμε τα παρακάτω:

```
INSERT INTO performance VALUES (999998, 999998, 10, 'Warm-Up', '11:15:00', 60, NULL);
# Artist Already Performing
INSERT INTO performance VALUES (999999, 999999, 10, 'Headline', '10:30:00', 120, NULL);
```

Η 1^η παράσταση διαρκεί 1 ώρα, δηλαδή μέχρι τις 12:15, ενώ η 2^η ξεκινάει στις 10:30 και τελειώνει στις 12:30 και εμφανίζεται ο ίδιος καλλιτέχνης. Επομένως, έχουμε επικάλυψη.

Error Code: 1644. Artist performing at the same time

Έλλειψη Προσωπικού

Στο event 999999 δεν έχουμε προσθέσει κανέναν εργαζόμενο. Επομένως αν καλέσουμε το Procedure *checkEnoughStaff* θα δημιουργηθεί πρόβλημα.

```
# Not Enough Staff
CALL checkEnoughStaff(999999);
```

Error Code: 1644. Not Enough Security

Λάθος Κωδικός Εισιτηρίου

Ο κωδικός EAN-13 '6969696969696' δεν είναι έγκυρος.

```
# Invalid EAN-13 Code
INSERT INTO ticket VALUES (999990, 999998, 1, 'VIP', '2020-10-10', '10.00', 'DEBIT', '6969696969693', 'USED');
```

Error Code: 1644. Wrong EAN Code

Πολλά VIP Εισιτήρια

Θέλουμε να εισαγάγουμε τα εξής δύο εισιτήρια:

```
INSERT INTO ticket VALUES (999990, 999998, 1, 'VIP', '2020-10-10', '10.00', 'DEBIT', '6969696969692', 'USED');
# Too Many VIP
INSERT INTO ticket VALUES (999991, 999998, 2, 'VIP', '2020-10-10', '10.00', 'DEBIT', '1234567891019', 'NOT USED');
```

Το event λαμβάνει χώρα σε σκηνή με χωρητικότητα 10, άρα μπορεί να πουληθεί μέχρι 1 VIP εισιτήριο.

Error Code: 1644. Too Many VIP

Διπλός Κωδικός EAN

Δε μπορούμε να βάλουμε πάλι εισιτήριο με κωδικό '6969696969692' αφού υπάρχει ήδη.

```
# Duplicate EAN Code
INSERT INTO ticket VALUES (999992, 999998, 3, 'REGULAR', '2020-10-10', '10.00', 'DEBIT', '6969696969692', 'NOT USED');
```

Error Code: 1062. Duplicate entry '6969696969692' for key 'ticket_EAN_13_code'

Επισκέπτης με Δύο Εισιτήρια

Αν βάλουμε τον 1^ο επισκέπτη να ξαναγοράσει εισιτήριο για την ίδια παράσταση θα δημιουργηθεί σφάλμα.

```
# Visitor Multiple Tickets
INSERT INTO ticket VALUES (999993, 999998, 1, 'REGULAR', '2020-10-10', '10.00', 'DEBIT', '1109876543211', 'NOT USED');
```

Error Code: 1644. Visitor Already Owns Ticket

Αξιολόγηση από Επισκέπτη που Δεν Παρακολούθησε την Παράσταση

Ο επισκέπτης με id 2 δεν έχει 'USED' εισιτήριο για την παράσταση 999997, επομένως το review δεν είναι έγκυρο.

```
# Visitor Not Attended Reviewed Event  
INSERT INTO review VALUES (999999, 2, 999997, 5, 5, 5, 5, 5);  
Error Code: 1644. Review Error
```

Ουρά Μεταπώλησης Εισιτηρίων

Ας βάλουμε ένα ενδεικτικό εισιτήριο στην παράσταση 999999. Αν ο ιδιοκτήτης του εισιτηρίου θέλει να το πουλήσει, δε μπορεί, γιατί η σκηνή έχει χωρητικότητα 5, άρα η παράσταση δεν έγινε sold-out.

```
INSERT INTO ticket VALUES  
(999993, 999999, 1, 'REGULAR', '2020-10-10', '10.00', 'DEBIT', '1109876543211', 'NOT USED');
```

```
# Performance Not Sold Out  
INSERT INTO seller(visitor_id, ticket_id) VALUES  
(1, 999993);  
Error Code: 1644. Tickets Not Sold Out
```

Ας προσθέσουμε κι άλλα εισιτήρια.

```
INSERT INTO ticket VALUES  
(999994, 999999, 2, 'VIP', '2020-10-10', '10.00', 'DEBIT', '1234567891019', 'NOT USED'),  
(999995, 999999, 3, 'REGULAR', '2020-10-10', '10.00', 'DEBIT', '1821182118210', 'NOT USED'),  
(999996, 999999, 4, 'STUDENT', '2020-10-10', '10.00', 'DEBIT', '8630138947827', 'NOT USED'),  
(999997, 999999, 5, 'STUDENT', '2020-10-10', '10.00', 'DEBIT', '1938473920480', 'USED');
```

Ο επισκέπτης με id 4 δεν μπορεί να πουλήσει το εισιτήριο με id 999994, καθώς δεν του ανήκει.

```
# Seller Not Owns Ticket  
INSERT INTO seller(visitor_id, ticket_id) VALUES  
(1, 999994);  
Error Code: 1644. Seller Error
```

Επίσης, ο επισκέπτης με id 5 δεν μπορεί να πουλήσει το εισιτήριό του γιατί είναι χρησιμοποιημένο.

```
# Ticket Used  
INSERT INTO seller(visitor_id, ticket_id) VALUES  
(5, 999997);  
Error Code: 1644. Seller Error
```

Ας υποθέσουμε ότι προστίθενται στην ουρά μεταπώλησης τα εξής εισιτήρια:

```
INSERT INTO seller(visitor_id, ticket_id) VALUES  
(1, 999993),  
(2, 999994),  
(3, 999995);
```

Ο πίνακας *resaleQueue* αυτόματα γίνεται:

resaleQueue_id	ticket_id	resaleQueue_timeInserted
1	999993	2025-05-11 19:36:40
2	999994	2025-05-11 19:36:40
3	999995	2025-05-11 19:36:40

Έστω ότι προκύπτει ο αγοραστής με id 6 που ψάχνει 'REGULAR' εισιτήριο για την συγκεκριμένη παράσταση. Από την ουρά μεταπώλησης μπορεί να πουληθεί είτε το 1^ο είτε το 3^ο εισιτήριο. Ωστόσο το 1^ο προηγείται χρονικά, γι' αυτό και το εισιτήριο αυτόματα μεταβιβάζεται στον επισκέπτη με id 6.

```
INSERT INTO buyer(visitor_id, music_event_id, ticketType_type) VALUES
(6, 999999, 'REGULAR');
```

ticket_id	music_event_id	visitor_id	ticketType_type	ticket_purchase_date	ticket_price	ticket_payment_method	ticket_EAN_13_code	ticket_status
999993	999999	6	REGULAR	2020-10-10	10	DEBIT	1109876543211	NOT USED

Τώρα ας υποθέσουμε ότι έρχεται ο εξής αγοραστής:

```
INSERT INTO buyer(visitor_id, music_event_id, ticketType_type) VALUES
(7, 999999, 'STUDENT');
```

Στην ουρά δεν υπάρχει εισιτήριο τύπου 'STUDENT', επομένως δεν πραγματοποιείται καμία αγορά. Έστω, όμως, ότι ο επισκέπτης με id 4 θέλει να διαθέσει το εισιτήριό του προς πώληση. Το εισιτήριο είναι τύπου 'STUDENT' οπότε αυτόματα μεταβιβάζεται στον αγοραστή με id 7.

```
INSERT INTO seller(visitor_id, ticket_id) VALUES
(4, 999996);
```

ticket_id	music_event_id	visitor_id	ticketType_type	ticket_purchase_date	ticket_price	ticket_payment_method	ticket_EAN_13_code	ticket_status
999996	999999	7	STUDENT	2020-10-10	10	DEBIT	8630138947827	NOT USED

Στην ουρά έχουν απομείνει τα εισιτήρια:

resaleQueue_id	ticket_id	resaleQueue_timeInserted
2	999994	2025-05-11 19:44:08
3	999995	2025-05-11 19:44:08

Ο πίνακας *seller* είναι:

visitor_id	ticket_id	seller_sold
1	999993	Y
2	999994	N
3	999995	N
4	999996	Y

Καθώς έχουν πουληθεί τα εισιτήρια των επισκεπτών 1,4 όπως δείχνει η 3^η στήλη.

Ο πίνακας *buyer* είναι ο εξής:

visitor_id	music_event_id	ticketType_type	ticket_id	buyer_timeInserted	buyer_sold
6	999999	REGULAR	NULL	2025-05-11 19:44:12	Y
7	999999	STUDENT	NULL	2025-05-11 19:44:16	Y

Φαίνεται ότι έχουν πουληθεί και τα δύο εισιτήρια.

Queries

Στα περισσότερα ερωτήματα χρειαζόταν κυρίως η ένωση (JOIN) των κατάλληλων πινάκων. Σχεδόν σε κάθε περίπτωση που χρειάστηκε να ενώσουμε δύο πίνακες, αυτό έγινε με χρήση της εντολής NATURAL JOIN, καθώς είχαμε προνοήσει, ώστε τα FOREIGN KEYS των πινάκων να έχουν το ίδιο όνομα με τα PRIMARY KEYS στα οποία αναφέρονταν, με σημαντική εξαίρεση το performer_id στους πίνακες artist και band, καθώς θα δημιουργούταν πρόβλημα σε αυτά. Επομένως, στις περιπτώσεις που θέλαμε τη σύνδεση μεταξύ καλλιτέχνη και συναυλίας για παράδειγμα, χρησιμοποιούσαμε INNER JOIN αντί για NATURAL JOIN με τον πίνακα artist και τον πίνακα performance, καθώς ο πίνακας performance περιέχει την πληροφορία performer_id αντί για artist_id, για λόγους γενικοποίησης.

Query 1

Βρείτε τα έσοδα του φεστιβάλ, ανά έτος από την πώληση εισιτηρίων, λαμβάνοντας υπόψη όλες τις κατηγορίες εισιτηρίων και παρέχοντας ανάλυση ανά είδος πληρωμής.

Στο ερώτημα αυτό χρειάζεται να συνδέσουμε τους πίνακες festival και ticket, για να μπορούμε να κάνουμε την ανάλυση που ζητείται. Καθώς οι δύο αυτοί πίνακες δε συνδέονται άμεσα, πρώτα τρέχουμε ένα NATURAL JOIN ανάμεσα στους πίνακες festival και music_event, κι ύστερα ένα LEFT JOIN με τον πίνακα ticket. Ο λόγος που κάνουμε LEFT JOIN αντί για NATURAL JOIN είναι για να είναι δυνατή η ανάλυση και για την περίπτωση που σε κάποιο φεστιβάλ δεν πουλήθηκε κανένα εισιτήριο, καθώς τότε θέλουμε το αποτέλεσμα να τυπωθεί, και να είναι 0, ενώ με NATURAL JOIN δε θα τυπωνόταν τίποτα. Από τον συνολικό αυτό πίνακα, μας ενδιαφέρουν μόνο η χρονιά του φεστιβάλ για το οποίο είναι το εισιτήριο, η τιμή του εισιτηρίου, καθώς και ο τύπος του εισιτηρίου και ο τρόπος με τον οποίον έγινε η πληρωμή. Ακόμα, κρατάμε και το ticket_id του κάθε εισιτηρίου, για να είμαστε σίγουροι πως κάθε στοιχείο που παίρνουμε από το subquery θα είναι ξεχωριστό. Το αποτέλεσμα του subquery, που κρατάει μόνο όσα μας ενδιαφέρουν, το αποθηκεύουμε σε προσωρινό πίνακα που λέγεται fest_ticket, τον οποίον χρησιμοποιούμε μετά για να κάνουμε την ανάλυση. Υστερα, προχωράμε στην ανάλυση ανά είδος εισιτηρίου και τρόπου πληρωμής.

Ο τρόπος που βρίσκουμε τα έσοδα του φεστιβάλ ανά είδος εισιτηρίου είναι ο εξής: από όλα τα εισιτήρια που βρίσκονται στον πίνακα fest_ticket, αν ο τύπος του εισιτηρίου είναι αυτός που μελετάμε, κρατάμε την τιμή του ως έχει, σε ένα πεδίο με κατάλληλο όνομα, αλλιώς το πεδίο αυτό το μηδενίζουμε. Έτσι, δημιουργούμε στον πίνακα μία στήλη, για παράδειγμα την *vip_price*, της οποίας το κάθε στοιχείο είναι 0 αν το εισιτήριο είναι άλλου είδους, ή η τιμή του αντίστοιχου εισιτηρίου, αν αυτό είναι του τύπου που μας ενδιαφέρει, στο συγκεκριμένο παράδειγμα, αν είναι τύπου VIP. Με την ίδια λογική, εργαστήκαμε και για την ανάλυση των μεθόδων πληρωμής, στο ίδιο subquery. Ακόμα, κρατάμε και μία στήλη η οποία περιέχει την τιμή του εισιτηρίου, ασχέτως τύπου ή μεθόδου πληρωμής. Το αποτέλεσμα του subquery που δημιουργεί αυτές τις στήλες για κάθε τύπο εισιτηρίου, και μέθοδο πληρωμής, αποθηκεύεται στον προσωρινό πίνακα fest_ticket_price. Το άθροισμα των τιμών όλων των στοιχείων στην κάθε στήλη για μία χρονιά, είναι τα έσοδα του συγκεκριμένου φεστιβάλ από τον αντίστοιχο τύπο εισιτηρίων ή μέθοδο πληρωμής, ενώ το άθροισμα των στοιχείων της στήλης *price*, προφανώς είναι τα συνολικά έσοδα του φεστιβάλ.

Τέλος, στο main query, για κάθε μία από αυτές τις στήλες υπολογίζουμε το άθροισμα όλων των στοιχείων. Καθώς είναι μεγάλο πλήθος αριθμών που προστίθενται μαζί, χρειάστηκε να στρογγυλοποιούμε επίσης το αποτέλεσμα, με ακρίβεια δύο δεκαδικών ψηφίων, καθώς τόσα χρησιμοποιούμε για κάθε είδος νομίσματος. Περαιτέρω, για λόγους κομψότητας, το στρογγυλοποιημένο αποτέλεσμα το κάνουμε *concat* με τον χαρακτήρα ‘€’, καθώς θεωρούμε ότι τα έσοδα αυτά τα έχουμε υπολογίσει σε ευρώ. Παρακάτω φαίνεται ο συνολικός κώδικας, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

WITH fest_ticket AS (SELECT festival_fest_year AS fest_year, festival_name AS fest_name,
    ticket_price AS price, ticketType_type AS ticket_Type, ticket_payment_method AS method
    FROM festival NATURAL JOIN music_event LEFT JOIN ticket
    ON music_event.music_event_id = ticket.music_event_id),
fest_ticket_price AS (SELECT fest_year, fest_name, price,
    CASE WHEN ticket_type = 'VIP' THEN price ELSE 0 END AS vip_price,
    CASE WHEN ticket_type = 'BACKSTAGE' THEN price ELSE 0 END AS back_price,
    CASE WHEN ticket_type = 'REGULAR' THEN price ELSE 0 END AS std_price,
    CASE WHEN ticket_type = 'STUDENT' THEN price ELSE 0 END AS student_price,
    CASE WHEN method = 'DEBIT' THEN price ELSE 0 END AS debit_price,
    CASE WHEN method = 'CREDIT' THEN price ELSE 0 END AS credit_price,
    CASE WHEN method = 'BANK_DEPOSIT' THEN price ELSE 0 END AS bank_price
    FROM fest_ticket)
SELECT fest_year AS 'Year', fest_name AS 'Festival',
    CONCAT(ROUND(SUM(price), 2), '€') AS 'Total Earnings',
    CONCAT(ROUND(SUM(std_price), 2), '€') AS 'Regular ticket Earnings',
    CONCAT(ROUND(SUM(vip_price), 2), '€') AS 'VIP Earnings',
    CONCAT(ROUND(SUM(back_price), 2), '€') AS 'Backstage Earnings',
    CONCAT(ROUND(SUM(student_price), 2), '€') AS 'Student ticket Earnings',
    CONCAT(ROUND(SUM(debit_price), 2), '€') AS 'Debit Earnings',
    CONCAT(ROUND(SUM(credit_price), 2), '€') AS 'Credit Earnings',
    CONCAT(ROUND(SUM(bank_price), 2), '€') AS 'Bank Deposit Earnings'
FROM fest_ticket_price
GROUP BY fest_year, fest_name
ORDER BY fest_year ASC;

```

Year	Festival	Total Earnings	Regular ticket Earnings	VIP Earnings	Backstage Earnings	Student ticket Earnings	Debit Earnings	Credit Earnings
2020	Bloom	174668.75€	55369.48€	40.41€	62173.18€	57085.68€	58221.45€	55514.86€
2021	Carnival	47784.63€	13977.48€	0€	18579.93€	15227.22€	16901.11€	12151.58€
2022	Aurora	2791.86€	1392.21€	0€	1172.91€	226.74€	655.48€	916.52€
2023	Eclipse Fest	4018.66€	951.78€	0€	1125.09€	1941.79€	1890.73€	1270.35€
2024	Gathering for tea	5279.32€	2072.56€	0€	1384.64€	1822.12€	1878.87€	1786.3€
2025	Wine	24440.9€	8566.21€	90.93€	8189.86€	7593.9€	7878.02€	9411.8€
2026	Ski	25627.78€	9439.15€	0€	7818.24€	8370.39€	6869.32€	8850.91€
2027	Ocean	48008.52€	16019.44€	0€	16401.73€	15587.35€	14794.72€	18092.15€
2028	Golden Vibes	54875.54€	18766.12€	0€	19553.96€	16555.46€	18400.38€	19808.64€
2029	Flowers	51898.42€	16635€	0€	18110.81€	17152.61€	17339.68€	17266.07€

Bank Deposit Earnings
60932.44€
18731.94€
1219.86€
857.58€
1614.15€
7151.08€
9907.55€
15121.65€
16666.52€
17292.67€

Query 2

Βρείτε όλους τους καλλιτέχνες που ανήκουν σε ένα συγκεκριμένο μουσικό είδος με ένδειξη αν συμμετείχαν σε εκδηλώσεις του φεστιβάλ για το συγκεκριμένο έτος.

Στο ερώτημα αυτό, θέλουμε τη σύνδεση ανάμεσα σε καλλιτέχνη και εμφάνιση. Για να πετύχουμε τη σύνδεση αυτή, κάναμε *INNER JOIN* ανάμεσα σε *artist* και *performance*, καθώς και τα δύο έχουν *FOREIGN KEY* που τα συνδέει με στοιχείο του πίνακα *performer*, και θέλουμε να ενώσουμε τα στοιχεία που αντιστοιχούν στον ίδιο *performer*. Βέβαια, πριν από αυτό, θέλουμε να συνδέσουμε επίσης τον καλλιτέχνη με τα είδη μουσικής στα οποία ανήκει, κάτι που πετυχαίνουμε με το *NATURAL JOIN* ανάμεσα στους πίνακες *art2genre* και *artist*. Ύστερα, το αποτέλεσμα αυτών των πράξεων το συνδέουμε με τον πίνακα *festival*, κάνοντας πρώτα ένα *NATURAL JOIN* με τον πίνακα *music_event*, καθώς ο πίνακας *performance* περιέχει *FOREIGN KEY* που αναφέρεται σε αυτόν, κι ύστερα ένα *NATURAL JOIN* με τον πίνακα *festival*. Ο πίνακας που έχει προκύψει, περιέχει όλα τα στοιχεία των τεσσάρων αυτών πινάκων, απ' τα οποία όμως μας ενδιαφέρουν μόνο ο καλλιτέχνης, τον οποίον τον προσδιορίζουμε με το *artist_id*, σε ποιο είδος μουσικής ανήκει, δηλαδή το *genre_desc*, και οι χρονιές τις οποίες έχει κάνει εμφάνιση στο φεστιβάλ. Επίσης, κρατάμε και το όνομα του καλλιτέχνη, για λόγους ευκρίνειας στο αποτέλεσμα.

Ωστόσο, θέλουμε να λάβουμε υπόψιν και τις παραστάσεις στις οποίες ο κάθε καλλιτέχνης έχει λάβει μέρος ως μέλος κάποιας μπάντας. Ο πίνακας *members* αναπαριστά τη σχέση *many – to – many* ανάμεσα στους καλλιτέχνες και τις μπάντες. Επομένως, μπορούμε να ενώσουμε τους πίνακες *band* και *performance* με ένα *INNER JOIN*, για τον ίδιο λόγο που το κάναμε και πριν, κρατώντας τα πεδία που χρειαζόμαστε για να γίνουν σωστά τα υπόλοιπα *JOINS*, τα οποία τα σώζουμε στο *subquery band_perf*, κι ύστερα να χρησιμοποιήσουμε τα στοιχεία αυτού του πίνακα για να πραγματοποιήσουμε τη σύνδεση ανάμεσα στον καλλιτέχνη και τις παραστάσεις της κάθε μπάντας στην οποία ανήκει. Για να το καταφέρουμε αυτό, αρκεί μία σειρά από πέντε *NATURAL JOINs*, πρώτα ανάμεσα στο *art2genre* και το *artist*, τα οποία τα ενώνουμε με τον πίνακα *members*, ύστερα με τον *band_perf*, τον *music_event* και, τέλος, τον πίνακα *festival*.

Τα αποτελέσματα από τα δύο μεγάλα *subqueries* που περιγράφαμε βρίσκονται στους πίνακες *A* και *B* αντίστοιχα, και χρησιμοποιούμε την πράξη *UNION* για να τα ενώσουμε στον πίνακα *art_g_year*, ο οποίος περιέχει ως πληροφορία όλα τα είδη μουσικής στα οποία ανήκει ο κάθε καλλιτέχνης, καθώς και κάθε χρονιά στης οποίας το φεστιβάλ έκανε κάποια εμφάνιση.

Ακόμα, χρησιμοποιούμε δύο μεταβλητές, τις *@f_year* και *@genre*, οι οποίες ελέγχουν ποια χρονιά και ποιο είδος μουσικής μελετάμε, αντίστοιχα. Μετά από όλα αυτά, αρκεί για κάθε καλλιτέχνη που ανήκει στο μουσικό είδος *@genre*, να ελέγχουμε αν είχε παίξει τη χρονιά *@f_year*, χρησιμοποιώντας τον πίνακα *art_g_year*. Τον έλεγχο αυτόν μπορούμε να τον κάνουμε με τη χρήση ενός *subquery*, το οποίο επιλέγει όλα τα στοιχεία από τη στήλη *fest_year* του *art_g_year* που έχουν το *artist_id* που μελετάει το πρόγραμμα τη συγκεκριμένη χρονική στιγμή. Αν η χρονιά *@f_year* βρίσκεται στο αποτέλεσμα αυτό, τότε τυπώνουμε *YES*, αλλιώς τυπώνουμε *NO*. Παρακάτω φαίνεται ο κώδικας που περιγράφαμε, καθώς και το αποτέλεσμα, για τις δοσμένες τιμές *@f_year = 2022* και *@genre = 'Pop'*:

```

SET @f_year := 2021;
SET @genre := 'Pop';
WITH band_perf (band_id, performance_id, music_event_id) AS (
    SELECT band_id, performance_id, music_event_id
    FROM band INNER JOIN performance ON band.band_performer_id = performance.performer_id),
A AS (
    SELECT artist_id, artist_name, genre_desc, festival_fest_year AS fest_year
    FROM art2genre NATURAL JOIN artist INNER JOIN performance NATURAL JOIN music_event NATURAL JOIN festival
    ON artist.artist_performer_id = performance.performer_id),
B AS (
    SELECT artist_id, artist_name, genre_desc, festival_fest_year AS fest_year
    FROM art2genre NATURAL JOIN artist NATURAL JOIN members NATURAL JOIN band_perf NATURAL JOIN music_event NATURAL JOIN festival),
art_g_year (artist_id, artist_name, genre_desc, fest_year) AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT DISTINCT artist_id AS id, artist_name AS 'Name', case
WHEN @f_year IN (
    SELECT fest_year FROM (
        SELECT artist_id, fest_year FROM art_g_year WHERE artist_id = id)
    AS T) THEN 'YES'
ELSE 'NO'
END AS wasThisYear FROM art_g_year
WHERE genre_desc = @genre;

```

id Name wasThisYear
12 Sam Smith NO
113 David Bryan YES
47 Jonny Buckland YES
124 The Edge NO
64 Jamie Cook NO
66 Matt Helders NO
112 Tico Torres NO
1 Beyoncé NO
44 Giannis Ploutarhos NO
27 Sakis Rouvas NO
55 James Valentine NO
70 Lindsey Buckingham NO
109 Nathan Followill YES

13 rows in set (0.02 sec)

Query 3

Βρείτε ποιοι καλλιτέχνες έχουν εμφανιστεί ως *warm up* περισσότερες από 2 φορές στο ίδιο φεστιβάλ.

Σε αυτό το ερώτημα, όπως προηγουμένως, χρειάζεται να αποκτήσουμε τον πίνακα που συνδέει τον κάθε καλλιτέχνη με όλες τις εμφανίσεις που έχει κάνει, είτε μόνος του ή ως μέλος κάποιας μπάντας. Για να το κάνουμε αυτό, χρησιμοποιούμε subqueries όπως τα περιγράφαμε και στο ερώτημα 2, όπου φτιάχνουμε έναν πίνακα ο οποίος περιέχει την πληροφορία για όλες τις παραστάσεις που έχει δώσει ο κάθε καλλιτέχνης μόνος του, τον A, κι έναν δεύτερο πίνακα για τις παραστάσεις που έχει δώσει ως μέλος κάποιας μπάντας, και μετά τους ενώνουμε με την πράξη *UNION*. Ακόμα, μέσα στα ίδια τα subqueries έχουμε βάλει κατάλληλο έλεγχο ώστε να περιέχουν πληροφορίες μόνο για τις εμφανίσεις που ήταν *Warm – Up*, ώστε να σώζονται προσωρινά λιγότερα περιττά δεδομένα. Επίσης, χρειάζεται να κρατάμε μόνο τα id των πινάκων που έχουμε χρησιμοποιήσει, αλλά για λόγους κομψότητας στο αποτέλεσμα, επιλέξαμε να κρατάμε επίσης τα ονόματα των καλλιτεχνών, καθώς και τη χρονιά του κάθε φεστιβάλ αντί για το id του φεστιβάλ. Όλη αυτή η πληροφορία αποθηκεύεται εν τέλει στον προσωρινό πίνακα *final*.

Τέλος, το μόνο που μας μένει να κάνουμε, είναι για κάθε συνδυασμό καλλιτέχνη-φεστιβάλ στον πίνακα *festival*, να μετρήσουμε πόσες εμφανίσεις έγιναν, καθώς έχουμε μόνο τις εμφανίσεις που μας ενδιαφέρουν, και να κρατήσουμε τα ζευγάρια στα οποία το πλήθος αυτό είναι μεγαλύτερο ή ίσο του τρία. Επιλέξαμε ακόμα να τυπώνεται το πλήθος αυτό στο αποτέλεσμα, καθώς και να τυπώνονται τα αποτελέσματα σε φθίνουσα σειρά

πλήθους εμφανίσεων. Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

WITH band_perf AS (select band_id, performance_id FROM band INNER JOIN performance
    ON band.band_performer_id = performance.performer_id WHERE type_performance_desc = 'Warm-Up'),
A AS (SELECT artist_id, artist_name, festival_fest_year AS fest_year, performance_id FROM artist INNER JOIN performance
    NATURAL JOIN music_event NATURAL JOIN festival ON artist.artist_performer_id = performance.performer_id
    WHERE type_performance_desc = 'Warm-Up'),
B AS (SELECT artist_id, artist_name, festival_fest_year AS fest_year, performance_id FROM artist NATURAL JOIN members
    NATURAL JOIN band_perf NATURAL JOIN music_event NATURAL JOIN festival),
final AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT artist_id, artist_name, fest_year, COUNT(*) AS warmups FROM final
GROUP BY artist_id, artist_name, fest_year
HAVING warmups >= 3
ORDER BY warmups DESC;

```

ID	Name	Year	warmups
123	Bono	2020	3
105	Bryan Yoder	2024	3
103	Drew Brown	2027	3

Query 4

Για κάποιο καλλιτέχνη, θρείτε το μέσο όρο αξιολογήσεων (Ερμηνεία καλλιτεχνών) και εμφάνιση (Συνολική εντύπωση). Η απάντηση σας θα πρέπει να περιέχει εκτός από το query, εναλλακτικό Query Plan, τα αντίστοιχα traces και τα συμπεράσματα σας από την μελέτη αυτών.

Αρχικά, χρησιμοποιούμε τη μεταβλητή `@artist_id`, για τον καθορισμό του καλλιτέχνη που μελετάμε. Στην εκτέλεση του Query που φαίνεται παρακάτω, του έχουμε δώσει την τιμή 12. Καθώς το ερώτημα ζητάει τη σύνδεση του καλλιτέχνη με τις κριτικές που του έχουν γίνει, θα πρέπει πρώτα να συνδέσουμε τον πίνακα με τις πληροφορίες για τους καλλιτέχνες (`artist`) με τον πίνακα με τις πληροφορίες για τις εμφανίσεις που έχουν κάνει (`performance`), καθώς ο πίνακας για τις κριτικές (`review`) περιέχει πληροφορία για την εμφάνιση στην οποία έγινε, όχι για τον καλλιτέχνη για τον οποίον είναι. Καθώς η κριτική είναι για όλη την εμφάνιση κι όχι μόνο για τον καλλιτέχνη, θα κρατήσουμε μόνο τα πεδία που αφορούν στην ερμηνεία του καλλιτέχνη (`review_interpretation`) και στη γενική εντύπωση για την εμφάνιση (`review_overall_impression`).

Το πρόβλημα της σύνδεσης του καλλιτέχνη με τις εμφανίσεις που έχει δώσει χωρίζεται σε δύο περιπτώσεις, καθώς είναι όλες ή solo εμφανίσεις, ή ως μέλος κάποιας μπάντας. Για την πρώτη περίπτωση, κάνουμε απλώς ένα `INNER JOIN` ανάμεσα στους πίνακες `artist` και `performance`, ώστε τα στοιχεία που ενώνουμε και στους δύο πίνακες να αναφέρονται στον ίδιο `performer`. Υστερα, κάνουμε ένα `NATURAL JOIN` με τον πίνακα `review`. Από τον πίνακα που προκύπτει με τα δύο αυτά `JOINS`, χρειάζεται να κρατήσουμε μόνο το `id` του κάθε καλλιτέχνη, και τις κριτικές που έχουν γίνει σε αυτόν (`review_interpretation` και `review_overall_impression`). Ακόμα, μπορούμε να μειώσουμε τον φόρτο του `DBMS` με το να αποθηκεύσουμε προσωρινά λιγότερα δεδομένα, τρέχοντας από τώρα τον έλεγχο ότι ο καλλιτέχνης είναι αυτός που μας ενδιαφέρει, χρησιμοποιώντας τη μεταβλητή `@artist_id`. Το αποτέλεσμα αποθηκεύεται προσωρινά στον πίνακα `A`.

Στη δεύτερη περίπτωση, χρειάζεται να παρεμβάλουμε τη σύνδεση με τις μπάντες ανάμεσα στους πίνακες `artist` και `performance`, που το κάνουμε με δύο `NATURAL JOINS` ανάμεσα στους πίνακες `artist`, `members` και `band`. Ακολουθεί ένα `INNER JOIN` με τον πίνακα `performance`, ώστε τα στοιχεία που ενώνουμε και στους δύο πίνακες να αναφέρονται στον `performer` που αντιστοιχεί στη μπάντα κι όχι στον καλλιτέχνη.

Έστερα, κάνουμε ένα *NATURAL JOIN* με τον πίνακα *review*. Από τον πίνακα που προκύπτει με αυτά τα *JOINS*, πάλι χρειάζεται να κρατήσουμε μόνο το *id* του κάθε καλλιτέχνη, και τις κριτικές που έχουν γίνει σε αυτόν (*review_interpretation* και *review_overall_impression*). Ακόμα, μπορούμε να μειώσουμε τον φόρτο του *DBMS* με το να αποθηκεύσουμε προσωρινά λιγότερα δεδομένα, τρέχοντας από τώρα τον έλεγχο ότι ο καλλιτέχνης είναι αυτός που μας ενδιαφέρει, χρησιμοποιώντας τη μεταβλητή `@artist_id`. Το αποτέλεσμα αποθηκεύεται προσωρινά στον πίνακα *B*. Στο τελευταίο *subquery*, ενώνουμε τα δεδομένα των πινάκων *A* και *B* με την πράξη *UNION*, κι αποθηκεύουμε το αποτέλεσμα στον προσωρινό πίνακα *art_rev*.

Τέλος, στο *main query*, τρέχουμε ένα *LEFT JOIN* ανάμεσα στους πίνακες *artist* και *art_rev*. Ο λόγος που το κάνουμε αυτό είναι ώστε να υπάρχει κάποιο αποτέλεσμα ακόμα κι αν δεν έχει γίνει κριτική στον δοσμένο καλλιτέχνη, καθώς σε αυτή την περίπτωση ο πίνακας *art_rev* θα είναι κενός. Επομένως, αν χρησιμοποιούσαμε *NATURAL* ή *INNER JOIN*, δε θα υπήρχε αποτέλεσμα, που δεν το θέλουμε. Ωστόσο, αυτό σημαίνει πως πρέπει άλλη μία φορά να βάλουμε έλεγχο ότι ο καλλιτέχνης είναι αυτός που μελετάμε. Από τον πίνακα, λοιπόν, που προκύπτει, τυπώνουμε το όνομα του καλλιτέχνη, καθώς και τους μέσους όρους των πεδίων *review_interpretation* και *review_overall_impression* που τον αφορούν.

Κάναμε αρκετές δοκιμές με το *FORCE INDEX* σε διάφορα σημεία του κώδικα και με διαφορετικά *INDEXES*, αλλά δεν προσέξαμε ουσιαστική διαφορά στον χρόνο εκτέλεσης του κώδικα. Μάλιστα, ενώ το αναμενόμενο θα ήταν χωρίς το *FORCE INDEX* να τερματίζει το *query* πιο γρήγορα, αφού ο *optimizer* βρίσκει τον βέλτιστο τρόπο να το εκτελέσει, στην πράξη βρήκαμε πως με το *FORCE INDEX* έτρεχε λίγο πιο γρήγορα, ασχέτως από τη θέση ή το *INDEX* που επιλέξαμε. Θεωρήσαμε πως αυτό μπορεί να οφείλεται στον σχετικά μικρό όγκο των δεδομένων, καθώς και στο ότι σώζει χρόνο ο *optimizer*, καθώς δε χρειάζεται να βρει τη βέλτιστη εκτέλεση. Ωστόσο, είτε το ένα έτρεχε πιο γρήγορα ή το άλλο, η διαφορά χρόνου στη μέση περίπτωση είναι πολύ μικρή για να έχει σημασία. Χρησιμοποιώντας *profiling*, μπορούμε να συγκρίνουμε τους χρόνους εκτέλεσης για δύο υλοποιήσεις του κώδικα, μία όπως φαίνεται πιο κάτω και μία με κάποιο *FORCE INDEX*. Ο πάνω χρόνος αντιστοιχεί στην κανονική υλοποίηση, ενώ ο κάτω χρόνος στην υλοποίηση με *FORCE INDEX*. Και στις δύο περιπτώσεις, το αποτέλεσμα είναι το ίδιο, και ο χρόνος μετριέται σε δευτερόλεπτα.

0.00151650

0.00140050

Δυστυχώς, δε βρήκαμε τρόπο να αλλάξουμε τη στρατηγική που θα χρησιμοποιηθεί για τα *JOINS*, καθώς η *MySQL*, αν και υποστηρίζει άλλα είδη, δε μας επιτρέπει να επιλέξουμε ποιο θα χρησιμοποιηθεί. Συγκεκριμένα, βρήκαμε πως με *optimization hints* προτείνουμε στον *optimizer* να προσπαθήσει *HASH JOIN*, αλλά τρέχοντας *EXPLAIN* σε δύο διαφορετικές περιπτώσεις, μία με το *hint* και μία χωρίς, είδαμε πως αυτά δεν τον αναγκάζουν να το τρέξει όπως θέλουμε, οπότε σε κάθε περίπτωση χρησιμοποιούσε *NESTED LOOP JOINS*. Παρακάτω φαίνεται το *EXPLAIN* για την υλοποίηση του κώδικα που έχουμε στο αρχείο *Q04.sql*, καθώς και το δέντρο που προκύπτει από την εντολή *EXPLAIN ANALYZE*.

```

+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key
+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY    | artist |        1 | const  | PRIMARY      | PRIMARY
| 1 | PRIMARY    | <derived2> |        1 | ref    | <auto_key1>  | <auto_key1>
| 2 | DERIVED    | artist |        1 | const  | PRIMARY,artist_idx | PRIMARY
| 2 | DERIVED    | performance |        1 | ref    | PRIMARY,performer_idx | performer_idx
| 2 | DERIVED    | review |        1 | ref    | review_performance_idx | review_performance_idx
| 4 | UNCACHEABLE UNION | artist |        1 | const  | PRIMARY      | PRIMARY
| 4 | UNCACHEABLE UNION | members |        1 | ref    | PRIMARY,memb_band_idx,memb_artist_idx | PRIMARY
| 4 | UNCACHEABLE UNION | band |        1 | eq_ref | PRIMARY,band_idx | PRIMARY
| 4 | UNCACHEABLE UNION | performance |        1 | ref    | PRIMARY,performer_idx | performer_idx
| 4 | UNCACHEABLE UNION | review |        1 | ref    | review_performance_idx | review_performance_idx
| 7 | UNION RESULT | <union2,4> |        1 | ALL   | NULL          | NULL
+-----+-----+-----+-----+-----+-----+
11 rows in set, 1 warning (0.01 sec)

+-----+-----+-----+-----+-----+
| key_len | ref | rows | filtered | Extra
+-----+-----+-----+-----+-----+
| 8   | const | 1    | 100.00 | NULL
| 8   | const | 2    | 100.00 | NULL
| 8   | const | 1    | 100.00 | NULL
| 8   | const | 3    | 100.00 | Using index
| 8   | music_festival_ntua.performance.performance_id | 6    | 100.00 | NULL
| 8   | const | 1    | 100.00 | Using index
| 8   | const | 1    | 100.00 | Using index
| 8   | music_festival_ntua.members.band_id | 1    | 100.00 | Using where
| 8   | music_festival_ntua.band.band_performer_id | 1    | 100.00 | Using index
| 8   | music_festival_ntua.performance.performance_id | 6    | 100.00 | NULL
| NULL | NULL  | NULL | NULL   | Using temporary
+-----+-----+-----+-----+-----+

```

-> Group aggregate: avg(art_rev.interpretation), avg(art_rev.overall_impression) (cost=6.51 rows=29.1) (actual time=0.312..0.312 rows=1 loops=1)
 -> Nested loop left join (cost=3.61 rows=29.1) (actual time=0.302..0.305 rows=9 loops=1)
 -> Rows fetched before execution (cost=0..0 rows=1) (actual time=0.00e-5..0.00e-6 rows=1 loops=1)
 -> Index lookup on art_rev using <auto_key1> (artist_id=@artist_id) (cost=11.3..11.3 rows=2.8) (actual time=0.3..0.301 rows=9 loops=1)
 -> Nested loop inner join (cost=4.68 rows=18.8) (actual time=0.189..0.205 rows=9 loops=1)
 -> Covering index lookup on performance using performer_idx (performer_id='12') (cost=0.551 rows=3) (actual time=0.0119..0.0137 rows=3 loops=1)
 -> Index lookup on review using review.performance_idx (performance_id=performance.performance_id) (cost=0.958 rows=6.25) (actual time=0.0698..0.063 rows=3 loops=3)
 -> Nested loop inner join (cost=3.38 rows=19.3) (actual time=0.0622..0.0622 rows=9 loops=1)
 -> Nested loop inner join (cost=1.12 rows=1.65) (actual time=0.0618..0.0618 rows=0 loops=1)
 -> Covering index lookup on members using PRIMARY (artist_id=@artist_id) (cost=0.35 rows=1) (actual time=0.0615..0.0615 rows=0 loops=1)
 -> Filter: (band.band_performer_id is not null) (cost=0.00..0.00 rows=1) (never executed)
 -> Single-row index lookup on band using PRIMARY (band_id=members.band_id) (cost=0.35 rows=1) (never executed)
 -> Covering index lookup on performance using performer_idx (performer_id=band.band_performer_id) (cost=0.415 rows=1.65) (never executed)
 -> Index lookup on review using review.performance_idx (performance_id=performance.performance_id) (cost=1.13 rows=6.25) (never executed)
 -> Index lookup on review using review.performance_idx (performance_id=performance.performance_id) (cost=1.13 rows=6.25) (never executed)

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

SET @artist_id = 12;
WITH band_rev AS (SELECT band_id, review_interpretation AS interpretation, review_overall_impression AS overall_impression
                  FROM band INNER JOIN performance NATURAL JOIN review
                  ON band.band_performer_id = performance.performer_id ),
     A AS (SELECT artist_id, review_interpretation AS interpretation, review_overall_impression AS overall_impression
            FROM artist INNER JOIN performance NATURAL JOIN review
            ON artist.artist_performer_id = performance.performer_id
            WHERE artist_id = @artist_id),
     B AS (SELECT artist_id, interpretation, overall_impression
            FROM artist NATURAL JOIN members NATURAL JOIN band_rev
            WHERE artist_id = @artist_id),
     art_rev AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT artist_name AS 'Name', AVG(interpretation) AS 'Average Interpretation',
       AVG(overall_impression) AS 'Average Overall Impression'
  FROM artist LEFT JOIN art_rev
  ON artist.artist_id = art_rev.artist_id
 WHERE artist.artist_id = @artist_id
 GROUP BY artist_name;

```

Name	Average Interpretation	Average Overall Impression
Sam Smith	2.6875	2.9375

Query 5

Βρείτε τους νέους καλλιτέχνες (ηλικία < 30 ετών) που έχουν τις περισσότερες συμμετοχές σε φεστιβάλ.

Για άλλη μία φορά, χρειάζεται να δημιουργήσουμε έναν προσωρινό πίνακα, στον οποίον να υπάρχει η πληροφορία που συνδέει κάθε καλλιτέχνη με όλες τις εμφανίσεις του στα φεστιβάλ μας. Η μεθοδολογία για να το πετύχουμε αυτό έχει περιγραφεί στα ερωτήματα 2 και 3 πιο αναλυτικά, αλλά συνοπτικά έγκειται στη χρήση subqueries για να φτιάξουμε προσωρινούς πίνακες που περιέχουν ένας τις εμφανίσεις του κάθε καλλιτέχνη ως solo act, κι ένας δεύτερος για τις εμφανίσεις του κάθε καλλιτέχνη ως μέλος μπάντας, τους οποίους πίνακες ενώνουμε για τη συνολική πληροφορία. Στους πίνακες αυτούς έχουμε αποθηκεύσει μόνο τα ids των καλλιτεχνών και των εμφανίσεων, το όνομα του κάθε καλλιτέχνη για να είναι πιο κομψό το αποτέλεσμα, καθώς και την ηλικία του καλλιτέχνη, την οποία την υπολογίζουμε από τη διαφορά της σημερινής ημερομηνίας (δηλαδή της μέρας που τρέχει το query) και της ημερομηνίας γέννησης του κάθε καλλιτέχνη, με την εντολή *TIMESTAMPDIFF*. Για λόγους βελτιστοποίησης, στους πίνακες *A* και *B*, δεν παίρνουμε τα δεδομένα κατευθείαν από τον πίνακα *artist*, αλλά από έναν προσωρινό πίνακα, τον *art_age*, ο οποίος δημιουργείται από ένα subquery στο οποίο κρατάμε μόνο τις πληροφορίες για τους καλλιτέχνες που είναι μικρότεροι από 30 χρονών, καθώς τα υπόλοιπα δεδομένα μας είναι περιττά. Συγκεκριμένα, τα δεδομένα που περιέχονται σε αυτόν τον πίνακα είναι μόνο το id του κάθε καλλιτέχνη, το *performer_id* του, το οποίο το χρειαζόμαστε για να γίνει σωστά το JOIN με τον πίνακα *performance*, και το όνομα του καλλιτέχνη, για λόγους κομψότητας στο αποτέλεσμα. Την ηλικία την υπολογίζουμε με τη χρήση της συνάρτησης *TIMESTAMPDIFF*, την οποία χρησιμοποιούμε για να πάρουμε τη διαφορά από τη σημερινή ημερομηνία και την ημερομηνία γέννησης κάθε καλλιτέχνη, μετρημένη σε χρόνια. Ύστερα, ενώνουμε τους δύο αυτούς πίνακες, σε έναν πίνακα που περιέχει όλα τα δεδομένα που χρειαζόμαστε, με την εντολή *UNION*, τον οποίον τον ονομάζουμε *young_artists*.

Τέλος, έχοντας ενώσει τους δύο πίνακες, χρειάζεται απλώς να μετρήσουμε το πλήθος των *performance_id* για κάθε καλλιτέχνη. Παρακάτω φαίνονται ο κώδικας που περιγράφαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```
WITH band_perf AS (SELECT band_id, performance_id
  FROM band INNER JOIN performance
  ON band.band_performer_id = performance.performer_id),
  art_age AS (SELECT artist_id, artist_name, artist_performer_id AS performer_id,
    |TIMESTAMPDIFF(year, artist_birthdate, CURRENT_DATE()) AS age
   FROM artist
   HAVING age < 30),
  A AS (SELECT artist_id, artist_name, performance_id, age
    FROM art_age NATURAL JOIN performance),
  B AS (SELECT artist_id, artist_name, performance_id, age
    FROM art_age NATURAL JOIN members NATURAL JOIN band_perf),
  young_artists AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT artist_id AS 'ID', artist_name AS 'Name', age AS Age, COUNT(*) AS Appearances
  FROM young_artists
 GROUP BY artist_id, artist_name, age
 ORDER BY Appearances DESC;
```

ID	Name	Age	Appearances
17	Doja Cat	29	2
7	Billie Eilish	23	1
10	Olivia Rodrigo	22	1

Query 6

Για κάποιο επισκέπτη, βρείτε τις παραστάσεις που έχει παρακολουθήσει και το μέσο όρο της αξιολόγησης του, ανά παράσταση. Η απάντηση σας θα πρέπει να περιέχει εκτός από το query, εναλλακτικό Query Plan, τα αντίστοιχα traces και τα συμπεράσματα σας από την μελέτη αυτών.

Αρχικά, χρησιμοποιούμε τη μεταβλητή `@visitor_id`, για τον καθορισμό του επισκέπτη που μελετάμε. Συγκεκριμένα, στις φωτογραφίες που παρατίθενται στο τέλος αυτού του Query, η μεταβλητή αυτή έχει την τιμή 201. Καθώς το ερώτημα ζητάει τη σύνδεση του επισκέπτη (`visitor`) με τις παραστάσεις (`music_event`) που έχει παρακολουθήσει, θα πρέπει πρώτα να συνδέσουμε τον πίνακα με τις πληροφορίες για τους επισκέπτες (`visitor`) με τον πίνακα με τις πληροφορίες για τις κριτικές που έχει αφήσει (`review`), καθώς ο πίνακας για τις κριτικές (`review`) περιέχει πληροφορία για την εμφάνιση στην οποία έγινε. Αυτό το πετυχαίνουμε με δύο *NATURAL JOINs* ανάμεσα στους πίνακες που αναφέραμε. Από τον πίνακα που προκύπτει, κρατάμε μόνο το `id` του επισκέπτη, το `id` της εμφάνισης στην οποία αναφέρεται η κριτική, καθώς και το σκορ της κριτικής, το οποίο το υπολογίζουμε από το άθροισμα των διαφόρων πεδίων της. Ακόμα, για να μην επιβαρύνουμε το *DBMS* με περιπτή πληροφορία, τρέχουμε ήδη τον έλεγχο ότι το `id` του επισκέπτη είναι αυτό που μας ενδιαφέρει, κι ύστερα αποθηκεύουμε το αποτέλεσμα στον προσωρινό πίνακα `rev_event`.

Τέλος, στο main query, τρέχουμε ένα *LEFT JOIN* ανάμεσα στους πίνακες `visitor` και `rev_event`. Ο λόγος που το κάνουμε αυτό, είναι ώστε να υπάρχει κάποιο αποτέλεσμα, ακόμα κι αν ο επισκέπτης δεν έχει αφήσει κάποια κριτική, καθώς σε αυτή την περίπτωση ο πίνακας `rev_event` θα είναι κενός. Επομένως, αν χρησιμοποιούσαμε *NATURAL* ή *INNER JOIN*, δε θα υπήρχε αποτέλεσμα. Ωστόσο, αυτό σημαίνει πως πρέπει να βάλουμε έλεγχο ότι ο επισκέπτης είναι αυτός που μελετάμε. Από τον πίνακα, λοιπόν, που προκύπτει, τυπώνουμε για κάθε παράσταση στην οποία άφησε κριτική το `id` της, καθώς και τον μέσο όρο της κριτικής που άφησε ο επισκέπτης για αυτήν, τον οποίον τον υπολογίζουμε διαιρώντας το άθροισμα που έχουμε κρατήσει δια πέντε. Για λόγους κομψότητας στο αποτέλεσμα, τα τυπώνουμε με αύξουσα σειρά `id` παράστασης.

Θεωρητικά, με το να βάλουμε *FORCE INDEX* σε κάποιον πίνακα, θα έπρεπε η εκτέλεση του κώδικα να πάρει περισσότερη ώρα, καθώς το *DBMS* δε θα τον τρέξει με τον βέλτιστο τρόπο. Για να τεστάρουμε ότι ισχύει αυτό, κάναμε δύο υλοποίήσεις του κώδικα, μία χωρίς *FORCE INDEX*, και μία στην οποία επίτηδες βάλαμε *INDEXES* τα οποία πιστεύουμε δεν διευκολύνουν τη γρήγορη εκτέλεση του κώδικα:

```
FROM review FORCE INDEX(review_visitor_idx) NATURAL JOIN performance FORCE INDEX(performer_idx)
    | NATURAL JOIN music_event FORCE INDEX(festival_idx)
```

Ο λόγος που τα *indexes* αυτά δε θεωρούμε πως βελτιώνουν την απόδοση του κώδικα, είναι επειδή δε σχετίζονται με τα *JOINS* που γίνονται στους πίνακες αυτούς, οπότε δυσχεραίνεται η ένωση των πινάκων.

Όντως, αν τρέξουμε τους δύο κώδικες στη σειρά, και μετά την εντολή *SHOW PROFILES* βλέπουμε δραματική διαφορά στους χρόνους εκτέλεσης:

```
0.00075275
```

```
0.01868050
```

Ο πάνω χρόνος αντιστοιχεί στην υλοποίηση χωρίς *FORCE INDEXES*, ενώ ο κάτω χρόνος στην υλοποίηση με αυτά, κι εμφανίζονται κι οι δύο χρόνοι σε δευτερόλεπτα. Προφανώς, ο άνθρωπος δε συνειδητοποιεί τη διαφορά χρόνου ανάμεσα σε 18.6 ms και 0.7 ms. Ωστόσο, είναι σημαντικό να αναφέρουμε πως οι χρόνοι αυτοί, και η διαφορά τους, θα ήταν αρκετά μεγαλύτεροι σε βάσεις με τάξεις μεγέθους περισσότερα δεδομένα, καθώς η παρούσα βάση δεδομένων έχει σχετικά μικρό όγκο πληροφορίας. Επομένως, είναι σημαντικό

το ότι ο optimizer βρίσκει πολύ καλύτερη λύση αν τον αφήσουμε να τρέξει μόνος του σε τέτοιες περιπτώσεις.

Δυστυχώς, δε βρήκαμε τρόπο να αλλάξουμε τη στρατηγική που θα χρησιμοποιηθεί για τα *JOINS*, καθώς η *MySQL*, αν και υποστηρίζει άλλα είδη, δε μας επιτρέπει να επιλέξουμε ποιο θα χρησιμοποιηθεί. Συγκεκριμένα, βρήκαμε πως με *optimization hints* μπορεί να προταθεί στον optimizer να χρησιμοποιήσει HASH JOIN, αλλά τρέχοντας *EXPLAIN* σε δύο διαφορετικές περιπτώσεις, μία με το hint και μία χωρίς, είδαμε πως αυτά δεν τον αναγκάζουν να το τρέξει όπως θέλουμε, οπότε σε κάθε περίπτωση χρησιμοποιούσε *NESTED LOOP JOINS*. Παρακάτω φαίνεται το *EXPLAIN* για την υλοποίηση του κώδικα που έχουμε στο αρχείο *Q06.sql*, καθώς και το δέντρο που προκύπτει από την εντολή *EXPLAIN ANALYZE*.

```
+-----+-----+-----+-----+-----+
| id | select_type | table      | partitions | type    | possible_keys          | key
+-----+-----+-----+-----+-----+
| 1 | SIMPLE     | visitor    | NULL       | const   | PRIMARY                | PRIMARY
| 1 | SIMPLE     | review     | NULL       | ref     | visitor_rating,performance_rated | visitor_rating
| 1 | SIMPLE     | performance | NULL      | eq_ref  | PRIMARY,event_performed | PRIMARY
| 1 | SIMPLE     | music_event | NULL      | eq_ref  | PRIMARY                | PRIMARY
+-----+-----+-----+-----+-----+
4 rows in set, 1 warning (0.00 sec)

+-----+-----+-----+
| ref           | rows | filtered | Extra
+-----+-----+-----+
| const          | 1   | 100.00   | Using index; Using temporary; Using filesort |
| const          | 1   | 100.00   | NULL
| music_festival_ntua.review.performance_id | 1   | 100.00   | NULL
| music_festival_ntua.performance.music_event_id | 1   | 100.00   | Using index
+-----+-----+-----+
-> Sort: rev_event.music_event_id (actual time=0.0239..0.0239 rows=1 loops=1)
-> Stream results (cost=0.35 rows=1) (actual time=0.0128..0.013 rows=1 loops=1)
-> Nested loop left join (cost=0.35 rows=1) (actual time=0.01..0.0102 rows=1 loops=1)
  -> Rows fetched before execution (cost=0..0 rows=1) (actual time=100e-6..100e-6 rows=1 loops=1)
  -> Nested loop inner join (cost=1.05 rows=1) (actual time=0.0077..0.0077 rows=0 loops=1)
    -> Nested loop inner join (cost=0..7 rows=1) (actual time=0.0075..0.0075 rows=0 loops=1)
      -> Index lookup on review using visitor_rating (visitor_id=@visitor_id)) (cost=0.35 rows=1) (actual time=0.0074..0.0074 rows=0 loops=1)
      -> Single-row index lookup on performance using PRIMARY (performance_id=review.performance_id) (cost=0..35 rows=1) (never executed)
    -> Single-row covering index lookup on music_event using PRIMARY (music_event_id=performance.music_event_id) (cost=0..35 rows=1) (never executed)
```

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

SET @visitor_id = 201;
WITH rev_event AS (SELECT visitor_id, music_event_id,
    (review_interpretation + review_sound_and_lighting + review_stage_presence
     + review_organization + review_overall_impression) AS score
  FROM review NATURAL JOIN performance NATURAL JOIN music_event
 WHERE visitor_id = @visitor_id)
SELECT music_event_id, score/5 AS 'Average Review'
  FROM visitor LEFT JOIN rev_event
    ON visitor.visitor_id = rev_event.visitor_id
   WHERE visitor.visitor_id = @visitor_id
 ORDER BY music_event_id ASC;

SET @visitor_id = 14;
WITH rev_event AS (SELECT visitor_id, music_event_id,
    (review_interpretation + review_sound_and_lighting + review_stage_presence +
     review_organization + review_overall_impression) AS score
  FROM review NATURAL JOIN performance NATURAL JOIN music_event
 WHERE visitor_id = @visitor_id)
SELECT music_event_id, score/5 AS 'Average Review'
  FROM visitor LEFT JOIN rev_event
    ON visitor.visitor_id = rev_event.visitor_id
   WHERE visitor.visitor_id = @visitor_id
 ORDER BY music_event_id ASC;

```

music_event_id	Average Review
1	3.4000
1	2.4000
4	3.0000

Query 7

Βρείτε ποιο φεστιβάλ είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού.

Για το ερώτημα αυτό, χρειάζεται να συνδέσουμε μεταξύ τους τους πίνακες *festival* και *staff*. Αυτό γίνεται υλοποιείται με μερικά *NATURAL JOINs* στη σειρά. Αρχικά, θα χρειαστούμε τον πίνακα *worksIn*, ο οποίος μπορεί να συνδέσει μεταξύ τους τους πίνακες *music_event* και *staff*. Όμως μας ενδιαφέρει το *festival*, οπότε πρώτα πρέπει να κάνουμε ένα *NATURAL JOIN* ανάμεσα στους πίνακες *festival* και *music_event*, κι ύστερα ακολουθούν τα *NATURAL JOINs* που συνδέουν αυτά με τον πίνακα *staff*. Σημαντικό είναι να κρατήσουμε τα ξεχωριστά ζευγάρια (*fest_year*, *staff_id*), καθώς προκύπτουν πολλά στοιχεία με το ίδιο ζευγάρι, και γι' αυτό έχουμε βάλει τα στοιχεία μας σε *GROUPs*, έτσι ώστε να σώζονται στον πίνακα του *subquery* μόνο μία φορά.

Από τον πίνακα *festival* χρειάζεται να κρατήσουμε μόνο το *festival_id*, αλλά για λόγους κομψότητας στο αποτέλεσμα, συμφωνήσαμε να κρατήσουμε τα πεδία *fest_year* και *fest_name* αντί για αυτό. Ακόμα, από τον πίνακα *staff* χρειαζόμαστε μόνο το πεδίο *levels_experience_desc*, το οποίο όμως είναι σε μορφή *VARCHAR*. Επειδή το χρειαζόμαστε σε μορφή αριθμού, για να μπορούμε να υπολογίσουμε τον ΜΟ του επιπέδου εμπειρίας ανά φεστιβάλ, θα κάνουμε ένα ακόμα *NATURAL JOIN* με τον πίνακα *staff_levels*, ο οποίος έχει τα επίπεδα εμπειρίας στη σειρά, κι άρα το χαμηλότερο id σημαίνει και λιγότερη εμπειρία.

Συνεπώς, έχοντας κάνει τα κατάλληλα *JOINS*, είναι δυνατό να βρούμε τον μέσο όρο του επιπέδου εμπειρίας ανά έτος, κάνοντας χρήση της συνάρτησης *AVG*. Καθώς θέλουμε να βρούμε ποια χρονιά έχει τον ελάχιστο μέσο όρο, τα κατατάσσουμε σε αύξουσα σειρά και κρατάμε μόνο το πρώτο στοιχείο. Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```
WITH fest_staff AS (SELECT festival_fest_year AS fest_year, festival_name AS fest_name, staff_id,
    staff_levels_id AS experience_level
    FROM festival NATURAL JOIN music_event NATURAL JOIN worksIn NATURAL JOIN staff NATURAL JOIN staff_levels
    GROUP BY fest_year, fest_name, staff_id
    HAVING COUNT(*) > 0)
SELECT fest_year, fest_name AS 'Festival', AVG(experience_level) AS average_exp_level
    FROM fest_staff
    GROUP BY fest_year, fest_name
    ORDER BY average_exp_level ASC
    LIMIT 1;
```

fest_year	Festival	average_exp_level
2021	Carnival	3.0754

Query 8

Βρείτε το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία.

Προφανώς, είναι πιο εύκολο να βρούμε το προσωπικό υποστήριξης που έχει εργασία μία δοσμένη ημερομηνία, κάνοντας ένα *NATURAL JOIN* ανάμεσα στους πίνακες *staff* και *worksIn*, κι ύστερα άλλο ένα *NATURAL JOIN* με τον πίνακα *music_event*, κι από αυτά να κρατήσουμε μόνο όσα στοιχεία έχουν ως *music_event_date* τη δοσμένη ημερομηνία, και ως *staff_role_desc* την τιμή *Support*. Επομένως, για να λύσουμε αυτό το ερώτημα, μπορούμε να τρέξουμε αυτό ως subquery, και να κρατήσουμε μόνο τα στοιχεία για τα οποία ισχύει μεν ότι το πεδίο *staff_role_desc* έχει την τιμή *Support*, αλλά δε βρίσκονται στο αποτέλεσμα αυτού του subquery δε. Για λόγους απλότητας, προσθέσαμε ένα subquery το οποίο κρατάει μόνο το *id* και το όνομα των ανθρώπων του προσωπικού υποστήριξης, από όλο τον πίνακα *staff*, τα αποτελέσματα του οποίου αποθηκεύονται προσωρινά στον πίνακα *support*. Ύστερα, χρειάζεται μόνο να ελέγχουμε για το *staff_id* αν υπάρχει στο subquery που περιγράψαμε παραπάνω.

Τέλος, έχουμε χρησιμοποιήσει τη μεταβλητή *@event_date*, η οποία ελέγχει την ημερομηνία την οποία μελετάμε. Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε, με δοσμένη τιμή στη μεταβλητή *@event_date* την ημερομηνία '2020-05-14':

```
SET @event_date = '2020-05-14';
WITH support AS (SELECT staff_id, staff_name
    FROM staff
    WHERE role_staff_desc = 'Support')
SELECT staff_id AS ID, staff_name AS 'Name'
    FROM support
    WHERE staff_id NOT IN (SELECT staff_id
        FROM support NATURAL JOIN worksIn NATURAL JOIN music_event
        WHERE music_event_date = @event_date);
```

ID	Name
263	Patrick Weiss
264	Vanessa Schuster
265	Sebastian Maier
266	Lisa Krüger
267	Tom Heinrich
268	Nicole Hahn
269	Daniel Bergmann
270	Johanna Scholz
271	Alexander Ernst
272	Melanie Nowak
273	Christopher Fabian
274	Annalena Lenz
275	Timo Metzger
276	Verena Adler
277	Marcel Thiel
278	Julia Stahl
279	André Blum
280	Carolin Dietrich
281	Markus Barth
282	Sara König
283	Kevin Sauer
284	Nadine Haas
285	René Otto
286	Mara Lorenz
287	Dennis Fritz
288	Kathrin Voigt
289	Robin Jung
290	Helene Rudolph
291	Lennart Busch
292	Stella Meier

Query 9

Βρείτε ποιοι επισκέπτες έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με περισσότερες από 3 παρακολουθήσεις.

Για το ερώτημα αυτό, χρειάζεται να συνδέσουμε τον πίνακα των επισκεπτών *visitor*, με τον πίνακα των παραστάσεων *music_event*. Το κάθε εισιτήριο στον πίνακα *ticket* περιέχει ως πληροφορία τον επισκέπτη στον οποίον ανήκει, και την παράσταση στην οποία αντιστοιχεί. Επομένως, μπορούμε να φτιάξουμε έναν προσωρινό πίνακα που να περιέχει ως πληροφορία όλους τους επισκέπτες και όλες τις παραστάσεις που έχει παρακολουθήσει ο κάθε επισκέπτης. Αυτό το κάνουμε με δύο *NATURAL JOINs*, πρώτα ανάμεσα στους πίνακες *visitor* και *ticket*, κι ύστερα με τον πίνακα *music_event*. Ωστόσο, από τον πίνακα που προκύπτει, τα μόνα στοιχεία που χρειαζόμαστε είναι το *id* του επισκέπτη και το πλήθος των παραστάσεων που έχει παρακολουθήσει, το οποίο το βρίσκουμε με την εντολή *COUNT*, έχοντας κάνει το κατάλληλο *GROUP BY* στα στοιχεία του. Ακόμα, για λόγους κομψότητας στο τελικό αποτέλεσμα, αποφασίσαμε να κρατήσουμε επίσης και το ονοματεπώνυμο του κάθε επισκέπτη.

Περαιτέρω, όταν υπολογίζουμε το πλήθος των παραστάσεων που έχει παρακολουθήσει, θέλουμε να μετρήσουμε μόνο όσες από αυτές ήταν στο χρονικό διάστημα που μας ενδιαφέρει. Για να το κάνουμε αυτό, χρησιμοποιούμε τη μεταβλητή *@start_date*, και μετράμε μόνο τις ημερομηνίες που είναι ανάμεσα στην ημερομηνία αυτή, κι ακριβώς έναν χρόνο μετά από αυτήν. Τα δεδομένα που προκύπτουν από όλη αυτήν την διαδικασία αποθηκεύονται στον προσωρινό πίνακα *vis_event*. Τέλος, για να διευκολύνουμε το *DBMS* αποθηκεύοντας προσωρινά λιγότερη πληροφορία, μπορούμε να βάλουμε και τον έλεγχο ότι το πλήθος των παραστάσεων είναι μεγαλύτερο ή ίσο του 3, ώστε να μην κρατάμε πληροφορία για τους επισκέπτες που δεν παρακολούθησαν τουλάχιστον 3 παραστάσεις μέσα στο χρονικό διάστημα που μελετάμε.

Έστερα, χρειάζεται να ελέγχουμε για κάθε επισκέπτη που υπάρχει στον πίνακα *vis_event*, αν υπάρχει κι άλλος επισκέπτης στον πίνακα αυτόν, που να έχει παρακολουθήσει το ίδιο πλήθος παραστάσεων. Αυτό γίνεται πραγματοποιείται, ελέγχοντας αν το πλήθος των παραστάσεων που έχει παρακολουθήσει ο συγκεκριμένος επισκέπτης, υπάρχει και άλλη φορά στον πίνακα *vis_event*. Επομένως, σε ένα *subquery* κρατάμε τα πλήθη από τον πίνακα *vis_event*, τα οποία αντιστοιχούν σε διαφορετικό *visitor* από αυτόν που μελετάει την τρέχουσα στιγμή το *main query*, και ελέγχουμε αν το πλήθος βρίσκεται μέσα σε αυτά, με την εντολή *IN*.

Τέλος, για λόγους κομψότητας στο τελικό αποτέλεσμα, αποφασίσαμε να το τυπώσουμε σε αύξουσα σειρά πλήθους παραστάσεων, ώστε όλοι οι επισκέπτες με ίδιο

πλήθος να τυπώνονται μαζί. Παρακάτω φαίνονται ο κώδικας που περιγράφαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε, με δοσμένη τιμή στη μεταβλητή `@event_date` την ημερομηνία '2029-01-01':

```
SET @start_date := '2029-01-01';
WITH vis_event AS (SELECT visitor_id, CONCAT(visitor_name, ' ', visitor_surname) AS visitor_name,
    COUNT(DISTINCT music_event_id) AS shows
    FROM visitor NATURAL JOIN ticket NATURAL JOIN music_event
    WHERE music_event_date BETWEEN @start_date AND DATE_ADD(@start_date, INTERVAL 1 YEAR)
    GROUP BY visitor_id, visitor_name
    HAVING shows >= 3)
SELECT V.shows AS Shows, V.visitor_id AS ID, V.visitor_name AS 'Name'
    FROM vis_event V
    WHERE V.shows IN (SELECT shows
        FROM vis_event
        WHERE visitor_id <> V.visitor_id)
    ORDER BY V.shows ASC;
```

Shows	ID	Name	Shows	ID	Name
7	211	Henri Leroux	8	5	Jian He
7	418	Erik Larsen	8	12	Hao Li
7	213	Claude Marchand	8	13	Dong Liu
7	214	Pascal Chevalier	8	14	Bo Ma
7	212	Henri Rolland	8	16	Feng Luo
7	416	Morten Andersen	8	17	Bo Xu
7	415	Harald Haugen	8	18	Hao Gao
7	420	Rune Johnsen	8	201	Philippe Lefebvre
7	414	Anders Moen	8	202	François Noël
7	412	Harald Berg	8	203	Roger Rolland
7	413	Svein Haugen	8	205	Christian Girard
7	210	Olivier Perrot	8	206	Marc Lemoine
7	419	Steinar Johansen	9	2	Peng He
8	3	Chao Liu	9	11	Hao Wu
9	8	Ming Guo	8	407	Leif Dahl
9			8	408	Jan Nilsen
			8	409	Svein Eriksen
			8	410	Jan Eriksen
			8	411	Ola Pedersen
			8	204	Jean Langlois
			8	417	Vidar Eriksen
			8	1	Tao Zhu
			8	406	Steinar Moen
			9	10	Tao Li
			9	20	Jie Ma
			9	9	Wei Zhu
			9	7	Tao Luo
			9	6	Tao Gao
			9	19	Xiang Wang

Query 10

Πολλοί καλλιτέχνες καλύπτουν περισσότερα από ένα μουσικά είδη. Ανάμεσα σε ζεύγη πεδίων (π.χ. ροκ, τζαζ) που είναι κοινά στους καλλιτέχνες, βρείτε τα 3 κορυφαία (top-3) ζεύγη που εμφανίστηκαν σε φεστιβάλ.

Στο ερώτημα αυτό χρειάζεται να συνδέσουμε τα είδη μουσικής genre με τις εμφανίσεις σε φεστιβάλ. Αρχικά, πρέπει να χρησιμοποιήσουμε τον ενδιάμεσο πίνακα `art2genre`, ο οποίος περιέχει την πληροφορία για την many-to-many σχέση ανάμεσα στις οντότητες των πινάκων genre και artist. Η σύνδεση αυτών των πινάκων γίνεται με τη χρήση ενός `NATURAL JOIN` ανάμεσα στους πίνακες `art2genre` και `artist`. Καθώς ο πίνακας `art2genre` περιέχει το πεδίο `genre_desc` του πίνακα `genre`, δε χρειαζόμαστε κάποια πληροφορία από τον πίνακα `genre`.

Ακολούθως, χρειάζεται να συνδέσουμε τον πίνακα αυτόν με τις εμφανίσεις που έχει πραγματοποιήσει ο κάθε καλλιτέχνης. Η σύνδεση αυτή έχει περιγραφεί πιο αναλυτικά σε προηγούμενα ερωτήματα, και δημιουργείται από το *UNION* δύο διαφορετικών *subqueries*, εκ των οποίων το πρώτο, που είναι αποθηκευμένο στον προσωρινό πίνακα A, περιέχει την πληροφορία για τις εμφανίσεις του κάθε καλλιτέχνη σόλο, ενώ το δεύτερο, αποθηκευμένο στον προσωρινό πίνακα B, τις εμφανίσεις κάθε καλλιτέχνη ως μέλος κάποιας μπάντας. Ο πίνακας A προκύπτει από τη σύνδεση που μόλις περιγράψαμε κι ύστερα ένα *INNER JOIN* για τη σύνδεση με τις εμφανίσεις. Ο πίνακας B δημιουργείται με παρόμοιο τρόπο, με τη διαφορά ότι συνδέουμε τον κάθε καλλιτέχνη, μέσω του πίνακα *members*, με τις τυχόν μπάντες στις οποίες μπορεί να ανήκει, έχοντας πρώτα συνδέσει τις μπάντες με ένα *INNER JOIN* με τις εμφανίσεις που έχουν κάνει. Ο προσωρινός πίνακας που περιέχει την ένωση των πινάκων A και B λέγεται *g_art_perf*.

Στα *subqueries* χρειάζεται να αποθηκεύσουμε μόνο τα πεδία που μας λένε το είδος μουσικής (*genre_desc*), τον καλλιτέχνη (*artist_id*) και την εμφάνιση (*performance_id*). Ο λόγος που χρειαζόμαστε τον καλλιτέχνη είναι για να δούμε τα ζευγάρια ειδών που υπάρχουν, ενώ τις εμφανίσεις για να μετρήσουμε πόσες φορές έχει παίξει καλλιτέχνης που ανήκει σε περισσότερα από ένα είδη μουσικής. Τέλος, υλοποιούμε ένα *SELF JOIN* του *g_art_perf*, χρησιμοποιώντας δύο αντίγραφά του, τα οποία ενώνουμε στα στοιχεία που έχουν κοινό *artist_id* και *performance_id*, αλλά διαφορετικά *genre_desc*, καθώς δε θέλουμε να κρατήσουμε ως ζεύγος ειδών κάποιο που είναι και τα δύο ίδια. Για να σιγουρευτούμε πως δε διπλομετράμε κάποιο ζεύγος από *genres* (πχ προφανώς οι συνδυασμοί ('Pop', 'Rock') και ('Rock', 'Pop')) θα έχουν ίδιο πλήθος εμφανίσεων, αλλά είναι το ίδιο ζεύγος), αντί να χρησιμοποιήσουμε τον τελεστή '<>' για την ανισότητα, χρησιμοποιούμε τον τελεστή '<', ώστε το πρώτο είδος στο ζεύγος να είναι το μικρότερο λεξικογραφικά. Υστερα, για τα στοιχεία που έχουμε κρατήσει, τρέχουμε μία εντολή *GROUP BY* ώστε να βρεθούν σε ομάδες ανάλογα το ζευγάρι μουσικών ειδών, κι ύστερα τα κατατάσσουμε σε φθίνουσα σειρά πλήθους εμφανίσεων, κάνοντας χρήση της *aggregate function COUNT()*, από τις οποίες ομάδες τυπώνουμε τις τρεις πρώτες. Δηλαδή τυπώνουμε μόνο τα τρία ζευγάρια μουσικών ειδών με τις περισσότερες κοινές εμφανίσεις.

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

WITH band_perf AS (SELECT band_id, performance_id
                   FROM band INNER JOIN performance
                   ON band.band_performer_id = performance.performer_id),
     A AS (SELECT genre_desc, artist_id, performance_id
            FROM art2genre NATURAL JOIN artist INNER JOIN performance
            ON artist.artist_performer_id = performance.performer_id),
     B AS (SELECT genre_desc, artist_id, performance_id
            FROM art2genre NATURAL JOIN artist NATURAL JOIN members NATURAL JOIN band_perf),
g_art_perf AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT G1.genre_desc AS first_genre, G2.genre_desc AS second_genre, COUNT(G1.performance_id) AS apps
      FROM g_art_perf G1, g_art_perf G2
     WHERE G1.artist_id = G2.artist_id AND G1.genre_desc < G2.genre_desc AND G1.performance_id = G2.performance_id
   GROUP BY first_genre, second_genre
  ORDER BY apps DESC LIMIT 3;

```

first_genre	second_genre	apps
Folk	Hip-Hop	5
Country	Hip-Hop	3
Electronic	Pop	2

Query 11

Βρείτε όλους τους καλλιτέχνες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον καλλιτέχνη με τις περισσότερες συμμετοχές σε φεστιβάλ.

Αρχικά, πρέπει να βρούμε ποιος καλλιτέχνης έχει τις περισσότερες συμμετοχές σε φεστιβάλ. Υστερα, μπορούμε να κρατήσουμε μόνο τους καλλιτέχνες που έχουν τουλάχιστον 5 λιγότερες εμφανίσεις. Για να το κάνουμε αυτό, ακολουθήσαμε τη συνηθισμένη μεθοδολογία για να συνδέσουμε τους καλλιτέχνες με τις εμφανίσεις που έχουν κάνει σε φεστιβάλ, είτε μόνοι τους ή ως μέλη κάποιας μπάντας. Πρώτα φτιάξαμε έναν προσωρινό πίνακα, τον A, ο οποίος περιέχει την πληροφορία που μας ενδιαφέρει για τις σόλο εμφανίσεις του κάθε καλλιτέχνη, κι έναν δεύτερο πίνακα B, στον οποίον είναι αποθηκευμένα τα χρήσιμα δεδομένα για τις εμφανίσεις που έχει κάνει κάθε καλλιτέχνης ως μέλος κάποιας μπάντας. Οι πληροφορίες που μας ενδιαφέρουν να κρατήσουμε στους πίνακες αυτούς είναι μόνο σε ποιον καλλιτέχνη αναφερόμαστε (*artist_id*), καθώς και σε ποια εμφάνιση αναφερόμαστε (*performance_id*), αλλά για λόγους κομψότητας στο αποτέλεσμα κρατάμε και το όνομα του κάθε καλλιτέχνη. Υστερα, αποθηκεύουμε την ένωση των δύο αυτών πινάκων στον προσωρινό πίνακα *all_apps*. Τέλος, τρέχουμε ένα subquery, του οποίου τα αποτελέσματα αποθηκεύονται στον προσωρινό πίνακα *app_counts*, στον οποίον από τα δεδομένα που περιέχει ο πίνακας *all_apps*, μετράμε τις συνολικές εμφανίσεις του κάθε καλλιτέχνη, με το aggregate function *COUNT()*, έχοντας πρώτα οργανώσει τα δεδομένα σε ομάδες με κοινό *id* και όνομα καλλιτέχνη, και σώζουμε το πλήθος σε πεδίο που ονομάζεται *apps*.

Τώρα για να βρούμε ποιος καλλιτέχνης έχει τις περισσότερες συμμετοχές σε φεστιβάλ, χρησιμοποιούμε ένα τελευταίο subquery, στο οποίο κατατάσσουμε τα περιεχόμενα του πίνακα *app_counts* σε φθίνουσα σειρά και κρατάμε μόνο το πεδίο *apps* του πρώτου στοιχείου, έχοντας πρώτα αφαιρέσει 4 από αυτό, για διευκόλυνση του *DBSM* πιο μετά, και το αποθηκεύουμε στον πίνακα *top_apps*.

Τέλος, στο main query, από τα στοιχεία του πίνακα *app_counts*, κρατάμε όλα όσα έχουν *apps* μικρότερο από το μοναδικό στοιχείο που βρίσκεται στον πίνακα *top_apps*. Αν δεν είχαμε αφαιρέσει στο ίδιο το subquery τέσσερεις από τις εμφανίσεις, θα έπρεπε να τις αφαιρέσουμε τώρα, κι ίσως αυξανόταν ο φόρτος εργασίας του *DBSM*, αν δεν το κάναμε σωστά.

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```
WITH band_perf AS (SELECT band_id, performance_id
    FROM band INNER JOIN performance
    ON band.band_performer_id = performance.performer_id),
A AS (SELECT artist_id, artist_name, performance_id
    FROM artist INNER JOIN performance
    ON artist.artist_performer_id = performance.performer_id),
B AS (SELECT artist_id, artist_name, performance_id
    FROM artist NATURAL JOIN members NATURAL JOIN band_perf),
all_apps AS (SELECT * FROM A UNION SELECT * FROM B),
app_counts AS (SELECT artist_id, artist_name, COUNT(*) AS apps
    FROM all_apps
    GROUP BY artist_id, artist_name),
top_apps AS (SELECT apps-4 as most_apps
    FROM app_counts
    ORDER BY apps DESC LIMIT 1)
SELECT artist_id, artist_name AS 'Name', apps AS Appearances
    FROM app_counts
    WHERE apps < ANY (SELECT * FROM top_apps);
```

artist_id	Name	Appearances	51	Wayne Sermon	1	130	Kurt Cobain	1
3	Adele	1	53	Daniel Platzman	1	132	Dave Grohls	1
7	Billie Eilish	1	80	Lea	1	136	Jeff Ament	1
10	Olivia Rodrigo	1	82	Chad Smith	1	46	Chris Martin	1
19	Lana Del Rey	1	98	Dave Keuning	1	54	Adam Levine	1
20	SZA	1	100	Ronnie Vann...	1	55	James Valent...	1
23	Miley Cyrus	1	114	Matt Bellamy	1	56	Mickey Madden	1
24	Halsey	1	115	Chris Wolste...	1	61	Pat Smear	1
26	Anna Vissi	1	117	Alex Pall	1	71	Mick Jagger	1
27	Sakis Rouvas	1	118	Drew Taggart	1	73	Charlie Watts	1
28	Despina Vandi	1	126	Larry Mullen Jr.	1	78	Dave "Phoeni..."	1
32	Katy Garbi	1	129	Tom DeLonge	1	92	Josh Dun	1
33	Antonis Remos	1				93	Hayley Williams	1
37	Michalis Hatzi...	1						
40	Nikos Oikono...	1						

Query 12

Βρείτε το προσωπικό που απαιτείται για κάθε ημέρα του φεστιβάλ, παρέχοντας ανάλυση ανά κατηγορία (τεχνικό προσωπικό, προσωπικό ασφαλείας, βοηθητικό προσωπικό).

Στο ερώτημα αυτό, πρέπει να βρούμε τρόπο να υπολογίσουμε το προσωπικό που χρειάζεται κάποιο φεστιβάλ, ανά μέρα, καθώς και το προσωπικό που όντως έχει. Για να γνωρίζει ο κώδικας σε ποιο φεστιβάλ αναφερόμαστε, χρησιμοποιούμε τη μεταβλητή `@f_id`, στην οποία έχουμε δώσει την τιμή 2 στο παράδειγμα που θα τρέξουμε παρακάτω.

Για το πρώτο κομμάτι, μπορούμε να υπολογίσουμε πόσους χρειαζόμαστε στο κομμάτι της ασφαλείας και πόσους χρειαζόμαστε για υποστηρικτικό προσωπικό, από τις χωρητικότητες των σκηνών στις οποίες λαμβάνουν χώρα οι διάφορες παραστάσεις του φεστιβάλ που μας ενδιαφέρει. Αυτό μπορούμε να το πετύχουμε συνδέοντας τους πίνακες `music_event` και `stage` με ένα `NATURAL JOIN`. Για να μειώσουμε τον φόρτο εργασίας του DBMS, τρέχουμε εδώ τον έλεγχο ότι η παράσταση στην οποία αναφέρεται το κάθε στοιχείο του πίνακα `music_event` ανήκει στο φεστιβάλ που μας ενδιαφέρει, ελέγχοντας αν το `festival_id` ισούται με την τιμή της μεταβλητής `@f_id`. Τα μόνα δεδομένα που χρειαζόμαστε εδώ είναι η ημερομηνία της παράστασης (`music_event_date`), και η χωρητικότητα της σκηνής (`stage_capacity`) στην οποία γίνεται η παράσταση αυτή. Καθώς

μπορεί να τύχει δύο παραστάσεις να γίνουν ίδια μέρα, απλά σε διαφορετικές σκηνές, κρατάμε το άθροισμα όλων των χωρητικοτήτων. Το αποτέλεσμα το αποθηκεύουμε προσωρινά στον πίνακα *date_cap*.

Ακόμα, αποφασίσαμε να τυπώνουμε και πόσους υπαλλήλους όντως έχουμε στις ημερομηνίες του φεστιβάλ που μελετάμε. Για να το κάνουμε αυτό, μπορούμε να χρησιμοποιήσουμε τον πίνακα *worksIn*, ο οποίος υλοποιεί τη *many-to-many* σχέση ανάμεσα στους πίνακες για το προσωπικό *staff* και για τις παραστάσεις *music_event*. Ο πίνακας αυτός περιέχει το *staff_id* για τον πίνακα *staff* και το *music_event_id* για τον πίνακα *music_event*, οπότε μπορούμε να κάνουμε *NATURAL JOIN* ανάμεσα σε αυτούς τους πίνακες. Από τον πίνακα που προκύπτει μετά τα δύο *JOINS*, κρατάμε μόνο τα πεδία που αφορούν την ημερομηνία (*music_event_date*) και τη θέση στην οποία δουλεύει κάθε εργαζόμενος (*role_staff_desc*). Για να είμαστε σίγουροι πως δε δημιουργείται στη συνέχεια επιπρόσθετο πρόβλημα, κρατάμε και το *id* του κάθε εργαζομένου, ώστε να αποφύγουμε τυχόν συμπίεση των δεδομένων για *duplicate data*. Αποθηκεύουμε το αποτέλεσμα στον προσωρινό πίνακα *date_all*.

Έστερα, για να μπορέσουμε να μετρήσουμε πόσους εργαζομένους έχουμε στην πραγματικότητα, ακολουθούμε μεθοδολογία παρόμοια με αυτήν του ερωτήματος 1, δηλαδή για κάθε θέση στην οποία μπορεί να δουλεύει κάποιος, κρατάμε μία *ξεχωριστή* στήλη, στην οποία για κάθε εργαζόμενο, το στοιχείο της έχει την τιμή 1 αν δουλεύει ο εργαζόμενος στη θέση αυτή, ή 0 αν δουλεύει σε κάποια άλλη θέση. Το αποτέλεσμα του *subquery* αυτού αποθηκεύεται στον προσωρινό πίνακα *date_staff_type*, και περιέχει επίσης την ημερομηνία της κάθε παράστασης στην οποία δουλεύει ο κάθε εργαζόμενος. Ακολούθως, μπορούμε να χρησιμοποιήσουμε τον πίνακα αυτόν για να αποθηκεύσουμε στον προσωρινό πίνακα *date_act* το πλήθος των εργαζομένων σε κάθε θέση που εργάζονται ανά ημερομηνία, προσθέτοντας τα στοιχεία κάθε στήλης.

Έχοντας αποθηκεύσει τη συνολική χωρητικότητα ανά ημερομηνία, μπορούμε να υπολογίσουμε πόσους εργαζομένους χρειαζόμαστε όντως σε κάθε ημερομηνία, με τους περιορισμούς που δίνονται στην εκφώνηση. Καθώς δε δίνεται περιορισμός για πόσους τεχνικούς χρειαζόμαστε σε κάθε παράσταση, υπολογίζουμε μόνο το υποστηρικτικό προσωπικό και το υποστηρικτικό ασφαλείας που μας είναι απαραίτητα ανά μέρα. Το *subquery* που υπολογίζει πόσους χρειαζόμαστε, κρατάει επίσης πόσους όντως έχουμε, και αποθηκεύουμε το αποτέλεσμα στον προσωρινό πίνακα *all_staff*.

Τέλος, στο *main query*, μπορούμε να υπολογίσουμε πόσο προσωπικό χρειαζόμαστε συνολικά, από το άθροισμα του πλήθους των εργαζομένων που χρειαζόμαστε στην ασφάλεια και ως υποστηρικτικό προσωπικό, και τελικά τυπώνουμε τα περιεχόμενα του πίνακα *all_staff*.

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

SET @f_id := 2;
WITH date_cap AS (SELECT music_event_date AS event_date, SUM(stage_capacity) AS capacity
    FROM music_event NATURAL JOIN stage
    WHERE festival_id = @f_id
    GROUP BY music_event_date),
date_all AS (SELECT music_event_date AS event_date, staff_id, role_staff_desc
    FROM staff NATURAL JOIN worksIn NATURAL JOIN music_event
    WHERE festival_id = @f_id),
date_staff_type AS (SELECT event_date,
CASE WHEN role_staff_desc = 'Technical' THEN 1 ELSE 0 END AS tech_all,
CASE WHEN role_staff_desc = 'Security' THEN 1 ELSE 0 END AS sec_all,
CASE WHEN role_staff_desc = 'Support' THEN 1 ELSE 0 END AS sup_all
    FROM date_all),
date_act AS (SELECT event_date, COUNT(tech_all) AS tech,
    COUNT(sec_all) AS sec, COUNT(sup_all) AS sup
    FROM date_staff_type
    GROUP BY event_date),
all_staff AS (SELECT event_date, CEILING(capacity/20) AS sec_needs,
    CEILING(capacity/50) AS sup_needs, sec, sup, tech
    FROM date_cap NATURAL JOIN date_act)
SELECT event_date AS 'Day', sec_needs + sup_needs AS 'Total Needed', sup_needs AS 'Support Needed',
    sup AS 'Actual Support', sec_needs AS 'Security Needed', sec AS 'Actual Security', tech AS 'Actual Technicians'
    FROM all_staff;

```

Day	Total Needed	Support Needed	Actual Support	Security Needed	Actual Security	Actual Technicians
2021-11-01	44	13	99	31	99	99
2021-11-02	97	28	206	69	206	206
2021-11-03	44	13	99	31	99	99

Query 13

Βρείτε τους καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους.

Χρειάζεται να βρούμε σε πόσες ηπείρους έχει κάνει εμφάνιση ο κάθε καλλιτέχνης. Μπορούμε να ξεκινήσουμε με την ίδια μεθοδολογία που έχουμε χρησιμοποιήσει και σε άλλα ερωτήματα για να συνδέσουμε τους καλλιτέχνες με τις εμφανίσεις που έχουν κάνει. Ωστόσο, πρέπει να επεκτείνουμε τους πίνακες ώστε να περιέχουν τις κατάλληλες πληροφορίες για τις ηπείρους στις οποίες έχει εμφανιστεί ο κάθε καλλιτέχνης. Την ήπειρο μπορούμε να τη βρούμε από τον πίνακα location. Για να συνδέσουμε τον πίνακα με τους καλλιτέχνες με τον πίνακα location, πρέπει να κάνουμε μία σειρά από NATURAL JOINs. Αρχικά, ο πίνακας *performance* περιέχει το πεδίο *music_event_id* που μας λέει σε ποια συναυλία ανήκει, και μπορεί έτσι να γίνει κατάλληλο NATURAL JOIN με τον πίνακα *music_event*. Ο πίνακας αυτός περιέχει το πεδίο *festival_id*, με το οποίο μπορούμε να κάνουμε NATURAL JOIN με τον πίνακα *festival*, καθώς δείχνει για ποιο φεστιβάλ έγινε η κάθε παράσταση. Τέλος, ο πίνακας *festival*, περιέχει το πεδίο *location_id*, το οποίο περιέχει την κατάλληλη πληροφορία για το NATURAL JOIN με τον πίνακα *location*.

Η διεργασία αυτή πραγματοποιείται δύο φορές, μία για τις σόλο εμφανίσεις που έχει κάνει ο κάθε καλλιτέχνης, όπου η πληροφορία αποθηκεύεται στον προσωρινό πίνακα A, και μία φορά για τις εμφανίσεις που έχει κάνει ως μέλος κάποιας μπάντας, και αποθηκεύουμε την αντίστοιχη πληροφορία στον πίνακα B. Ο κάθε πίνακας από τους δύο περιέχει το id του κάθε καλλιτέχνη, για να γνωρίζουμε σε ποιον αναφέρομαστε, και το *location_continent*, δηλαδή την ήπειρο. Ακόμα, αποθηκεύουμε και το όνομα του κάθε καλλιτέχνη, για να είναι πιο κομψό το αποτέλεσμα. Υστερα, αποθηκεύουμε στον πίνακα *art_cont* το UNION των πινάκων A και B.

Τέλος, στο main query, τα στοιχεία του πίνακα *art_cont* τα ομαδοποιούμε ανάλογα σε ποιον καλλιτέχνη αναφέρονται, και μετράμε πόσα βρίσκονται σε κάθε ομάδα, για να μετρήσουμε σε πόσες ηπείρους έχει κάνει οποιουδήποτε είδους εμφάνιση. Από αυτά, τυπώνουμε μόνο όσα έχουν 3 ή περισσότερες ηπείρους.

Θεωρούμε σημαντικό να αναφέρουμε πως αν μας ενδιέφεραν μόνο οι σόλο εμφανίσεις του κάθε καλλιτέχνη, ή μόνο οι εμφανίσεις ως μέλος ομάδας, θα έπρεπε να χρησιμοποιήσουμε τη λέξη-κλειδί *DISTINCT* στο μέτρημα των ηπείρων, καθώς στον πίνακα A (ή στον πίνακα B αντίστοιχα) αποθηκεύονται duplicates των ίδιων πλειάδων. Για παράδειγμα, δείχνουμε τα στοιχεία του πίνακα A στα οποία το *artist_id* είναι ίσο με 123, όπου αριστερά δε χρησιμοποιούμε τη λέξη-κλειδί *DISTINCT*, ενώ δεξιά τη χρησιμοποιούμε:

<i>artist_id</i>	<i>artist_name</i>	<i>continent</i>	<i>artist_id</i>	<i>artist_name</i>	<i>continent</i>
123	Bono	Europe	123	Bono	Europe
123	Bono	Europe	123	Bono	Asia
123	Bono	Europe	123	Bono	Asia

Οστόσο, δε χρειάζεται να χρησιμοποιήσουμε τη λέξη-κλειδί *DISTINCT* στον κώδικα μας, καθώς η πράξη *UNION* δεν κρατάει στο αποτέλεσμα τις duplicate τιμές.

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

WITH band_perf AS (SELECT band_id, music_event_id
    FROM band INNER JOIN performance
    ON band.band_performer_id = performance.performer_id),
A AS (SELECT artist_id, artist_name, location_continent AS continent
    FROM artist INNER JOIN performance NATURAL JOIN music_event NATURAL JOIN festival NATURAL JOIN location
    ON artist.artist_performer_id = performance.performer_id),
B AS (SELECT artist_id, artist_name, location_continent AS continent
    FROM artist NATURAL JOIN members NATURAL JOIN band_perf NATURAL JOIN music_event NATURAL JOIN festival
    NATURAL JOIN location),
art_cont AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT artist_id AS ID, artist_name AS 'Name', COUNT(continent) AS different_continents
    FROM art_cont
    GROUP BY artist_id, artist_name
    HAVING different_continents >= 3;

```

ID	Name	different_continents
101	Ryan Tedder	3
75	Mike Shinoda	3
85	Mike Dirnt	3
63	Alex Turner	4
108	Caleb Followill	5
89	Adam Hann	4
76	Brad Delson	4
124	The Edge	3
105	Bryan Yoder	4
106	Jared Followill	3
69	Stevie Nicks	3
60	Nate Mendel	3
34	Peggy Zina	3
103	Drew Brown	3
90	Ross MacDo...	3
		84 Billie Joe Arm... 3
		86 Tre Cool 3
		102 Zach Filkins 3
		104 Eddie Fisher 3

Query 14

Βρείτε ποια μουσικά είδη είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές με τουλάχιστον 3 εμφανίσεις ανά έτος.

Χρειάζεται να συνδέσουμε κατάλληλα την πληροφορία που βρίσκεται στον πίνακα *genre*, για να έχουμε τα μουσικά είδη, με την πληροφορία που βρίσκεται στον πίνακα *festival*, για να έχουμε πληροφορίες για το έτος. Όπως και στο ερώτημα 10, μπορούμε να συνδέσουμε την πληροφορία για τα μουσικά είδη με τις διάφορες εμφανίσεις που έχουν γίνει. Ύστερα, χρειάζεται να τα συνδέσουμε με τις παραστάσεις στις οποίες έγιναν οι εμφανίσεις αυτές, και μετά μπορούμε να τα συνδέσουμε με τα φεστιβάλ τα οποία αποτελούσαν οι παραστάσεις αυτές, και να πάρουμε έτσι τα έτη.

Καθώς το όνομα του μουσικού είδους υπάρχει στους πίνακες *art2genre* και *band2genre* που υλοποιούν τις many-to-many σχέσεις ανάμεσα στις οντότητες *artist* και *genre*, και *band* και *genre* αντίστοιχα, δε χρειάζεται να συνδέσουμε κάτι με τον πίνακα *genre*. Ωστόσο, τώρα πρέπει να μελετήσουμε τις εμφανίσεις των μουσικών ειδών σε τρεις διαφορετικές περιπτώσεις, πρώτον τις σόλο εμφανίσεις καλλιτεχνών, δεύτερον τα μουσικά είδη στα οποία ανήκουν καλλιτέχνες σε εμφανίσεις με μπάντες, και τρίτον τα μουσικά είδη των μπαντών για τις δικές τους εμφανίσεις. Αρχικά, αποθηκεύουμε σε έναν προσωρινό πίνακα, τον οποίον ονομάζουμε *band_perf_year* τη σύνδεση της κάθε μπάντας με τις εμφανίσεις της, και τις χρονιές στις οποίες έγιναν αυτές, χρησιμοποιώντας ένα *INNER JOIN* ανάμεσα στους πίνακες *band* και *performance*, κι ύστερα 2 *NATURAL JOINs*, πρώτα με τον πίνακα *music_event*, κι ύστερα με τον πίνακα *festival*. Τα δεδομένα που περιέχει αυτός ο πίνακας είναι το id της κάθε μπάντας, τα id όλων των εμφανίσεών τους, και οι χρονιές στην οποία πραγματοποιήθηκαν αυτές.

Στην πρώτη περίπτωση, κάνουμε πρώτα ένα *NATURAL JOIN* ανάμεσα στους πίνακες *art2genre* και *artist*, κι ύστερα συνδέουμε τα δεδομένα για τους καλλιτέχνες με τις σόλο εμφανίσεις τους, κάνοντας ένα *NATURAL JOIN* με τον πίνακα *performance*. Το αποτέλεσμα αυτό το κάνουμε 2 *NATURAL JOINs*, πρώτα με τον πίνακα *music_event*, και μετά με τον πίνακα *festival*, κι ύστερα αποθηκεύουμε το τελικό αποτέλεσμα του subquery αυτού στον προσωρινό πίνακα A. Αντίστοιχα, για τη δεύτερη περίπτωση, κάνουμε πάλι *NATURAL JOIN* ανάμεσα στους πίνακες *art2genre* και *artist*, στους οποίους μετά κάνουμε *NATURAL JOINs* με τους πίνακες *members* και μετά *band_perf_year*. Το αποτέλεσμα αυτό βρίσκεται στον προσωρινό πίνακα B. Για την τρίτη περίπτωση, κάνουμε ένα *NATURAL JOIN* ανάμεσα στους πίνακες *band2genre* και *band_perf_year*, κι αποθηκεύουμε το αποτέλεσμα στον προσωρινό πίνακα C. Και στις τρεις περιπτώσεις, τα μόνα δεδομένα που χρειάζεται να κρατάμε είναι το μουσικό είδος στο οποίο αναφερόμαστε (*genre_desc*), η κάθε εμφάνιση που έχει γίνει που συμπεριλάμβανε το είδος αυτό (*performance_id*) και το έτος στο οποίο έγινε αυτή η εμφάνιση (*festival_fest_year*). Τέλος, χρησιμοποιούμε *UNION* ανάμεσα στα δεδομένα των πινάκων A, B και C, και σώζουμε το αποτέλεσμα στον πίνακα *g_perf_year*. Καθώς η πράξη *UNION* σβήνει τα duplicate date από το αποτέλεσμά της, μπορούμε μετά να οργανώσουμε τα δεδομένα του πίνακα *g_perf_year* ανά μουσικό είδος και έτος, και να μετρήσουμε πόσες εμφανίσεις είχε το κάθε μουσικό είδος ανά έτος. Το αποτέλεσμα αυτό αποθηκεύεται στον προσωρινό πίνακα *g_year_apps*. Ακόμα, για να μειώσουμε τον φόρτο εργασίας του DBMS, αποθηκεύοντας λιγότερα αρχεία, στον πίνακα *g_year_apps* έχουμε αποθηκεύσει μόνο στοιχεία που έχουν περισσότερες από 3 εμφανίσεις.

Τέλος, στο main query, μπορούμε να υλοποιήσουμε ένα *SELF JOIN* στο *g_year_apps*, χρησιμοποιώντας δύο αντίγραφά του, και παίρνοντας από αυτά τα στοιχεία

που έχουν κοινό μουσικό είδος και πλήθος εμφανίσεων, αλλά στο στοιχείο του δεύτερου πίνακα, ο χρόνος είναι ο αμέσως επόμενος από τον χρόνο του στοιχείου του πρώτου πίνακα.

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```

WITH band_perf_year AS (SELECT band_id, performance_id, festival_fest_year AS fest_year
    FROM band INNER JOIN performance NATURAL JOIN music_event NATURAL JOIN festival
    ON band.band_performer_id = performance.performer_id),
A AS (SELECT genre_desc, performance_id, festival_fest_year AS fest_year
    FROM art2genre NATURAL JOIN artist INNER JOIN performance NATURAL JOIN music_event NATURAL JOIN festival
    ON artist.artist_performer_id = performance.performer_id),
B AS (SELECT genre_desc, performance_id, fest_year
    FROM art2genre NATURAL JOIN artist NATURAL JOIN members NATURAL JOIN band_perf_year),
C AS (SELECT genre_desc, performance_id, fest_year
    FROM band2genre NATURAL JOIN band_perf_year),
g_perf_year AS (SELECT * FROM A UNION SELECT * FROM B UNION SELECT * FROM C),
g_year_apps AS (SELECT genre_desc, fest_year, COUNT(*) AS appearances
    FROM g_perf_year
    GROUP BY genre_desc, fest_year
    HAVING appearances >= 3)
SELECT G1.genre_desc AS Genre, G1.fest_year AS 'First Year', G2.fest_year AS 'Second Year', G1.appearances
    FROM g_year_apps G1, g_year_apps G2
    WHERE G1.genre_desc = G2.genre_desc AND G1.fest_year + 1 = G2.fest_year AND G1.appearances = G2.appearances
    ORDER BY G1.genre_desc ASC, G1.fest_year ASC;

```

Genre	First Year	Second Year	appearances
Blues	2021	2022	3
Classical	2028	2029	4
Folk	2027	2028	4

Query 15

Βρείτε τους top-5 επισκέπτες που έχουν δώσει συνολικά την υψηλότερη βαθμολόγηση σε ένα καλλιτέχνη (όνομα επισκέπτη, όνομα καλλιτέχνη και συνολικό σκορ βαθμολόγησης).

Για το ερώτημα αυτό, πρέπει να συνδέσουμε τον πίνακα οντοτήτων για τους επισκέπτες visitor, μέσω του πίνακα οντοτήτων για τις κριτικές review, με τον πίνακα οντοτήτων για τους καλλιτέχνες artist. Ο πίνακας review περιέχει πληροφορία για την εμφάνιση performance στην οποία αναφέρεται η κάθε κριτική, που σημαίνει πως θα πρέπει ξανά να δουλέψουμε για δύο περιπτώσεις, καθώς η κριτική έγινε ή για σόλο εμφάνιση, ή για την εμφάνιση κάποιας μπάντας.

Στην πρώτη περίπτωση, κάνουμε *INNER JOIN* ανάμεσα στους πίνακες *artist* και *performance*, καθώς και οι δύο πίνακες περιέχουν πληροφορία για τον *performer* στον οποίον ανήκουν. Έπειτα, κάνουμε δύο *NATURAL JOINS*, ένα με τον πίνακα *review*, κι ένα με τον πίνακα *visitor*. Ωστόσο, για ευκολία στο DBMS αποφασίσαμε τα δύο τελευταία *JOINS* να τα κάνουμε αφού ενώσουμε τις δύο περιπτώσεις, για να μη τα εκτελέσει και στις δύο περιπτώσεις μαζί. Ομοίως, στη δεύτερη περίπτωση, αφού κάνουμε *NATURAL JOINS* ανάμεσα στους πίνακες *artist*, *members* και *band*, κι ένα *INNER JOIN* με τον πίνακα *performance* για τις εμφανίσεις των μπαντών, αποφασίσαμε να μη συνεχίσουμε με τη σύνδεση με τις κριτικές. Και στις δύο περιπτώσεις, τα αποτελέσματα των δύο subqueries περιέχουν μόνο το id και το όνομα του καλλιτέχνη (*artist_id* και *artist_name* αντίστοιχα), και το id της εμφάνισης την οποία μελετάμε. Αν είχαμε αποφασίσει να συνεχίσουμε στα

subqueries για τη σύνδεση με τις κριτικές και τους επισκέπτες, το DBMS θα έπρεπε να αποθηκεύσει προσωρινά περισσότερες πληροφορίες, εκτός απ' το ότι θα έκανε *JOINS* σε παρόμοιους πίνακες δύο φορές. Αντιθέτως, μετά ενώνουμε τα δεδομένα των δύο πινάκων στον προσωρινό πίνακα *art_perf*, χρησιμοποιώντας την πράξη *UNION*.

Τέλος, στο main query, κάνουμε τα *NATURAL JOINS* ανάμεσα στους πίνακες *art_perf*, *review* και *visitor*, για να τελειώσουμε τη σύνδεση ανάμεσα σε καλλιτέχνη και επισκέπτη που άφησε κριτική. Από τον τελικό αυτόν πίνακα, τυπώνουμε το όνομα του επισκέπτη και το όνομα του καλλιτέχνη, καθώς και το σκορ της κριτικής για τον καλλιτέχνη, το οποίο υπολογίζεται ως το άθροισμα της κριτικής για την ερμηνεία που έκανε ο καλλιτέχνης και της κριτικής για τη γενική εικόνα της εμφάνισης. Καθώς θέλουμε να δούμε μόνο της πορέντε κριτικές με τα καλύτερα σκορ, κατατάσσουμε τα τελικά δεδομένα σε φθίνουσα σειρά, με βάση το σκορ, και τυπώνουμε μόνο τα πέντε πρώτα αποτελέσματα.

Παρακάτω φαίνονται ο κώδικας που περιγράψαμε, καθώς και το αποτέλεσμα που λαμβάνουμε όταν τον τρέχουμε:

```
WITH band_perf (band_id, performance_id) AS (SELECT band_id, performance_id
    FROM band INNER JOIN performance
    ON band.band_performer_id = performance.performer_id),
A AS (SELECT artist_id, artist_name, performance_id
    FROM artist INNER JOIN performance
    ON artist.artist_performer_id = performance.performer_id),
B AS (SELECT artist_id, artist_name, performance_id
    FROM artist NATURAL JOIN members NATURAL JOIN band_perf),
art_perf AS (SELECT * FROM A UNION SELECT * FROM B)
SELECT CONCAT(visitor_name, ' ', visitor_surname) AS Reviewer, artist_name AS Artist,
    review_interpretation + review_overall_impression AS Score
FROM visitor NATURAL JOIN review NATURAL JOIN art_perf
ORDER BY Score DESC LIMIT 5;
```

Reviewer	Artist	Score
Rong Wu	Bono	10
Jun Wu	Bono	10
Tao Chen	Bono	10
François Noël	Sam Smith	10
Pierre Girard	Sam Smith	10

Βιβλιογραφία και χρήσιμες Ιστοσελίδες

- ❖ Για τη δημιουργία του ER-diagram έγινε χρήση της ιστοσελίδας draw.io
- ❖ Πηγή για εικόνες: <https://images.unsplash.com>
- ❖ Έγινε χρήση AI/LLM για την ανάπτυξη δεδομένων, όπως οι καλλιτέχνες και οι ημερομηνίες γέννησης τους και οι μπάντες με τα μέλη τους, ώστε τα δεδομένα αυτά να είναι υπαρκτά.
- ❖ Για αναφορά για τις εντολές της SQL και για ρυθμίσεις της MySQL βασιστήκαμε στην <https://dev.mysql.com/doc/>