



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ

Χιώτης Χαράλαμπος (03122142)
Τσουρούφλη Ζωή (03122062)
Αλεξάνδρα Καραλή (03121084)

Πίνακας Περιεχομένων

Μέρος Α: Σχεδιασμός και υλοποίηση Βάσης Δεδομένων

1. ER DIAGRAM

2. Relational Diagram

- 2.1. Να ορίσετε όλους τους απαραίτητους περιορισμούς που θα εξασφαλίζουν την ορθότητα της ΒΔ. Αυτοί είναι περιορισμοί ακεραιότητας, κλειδιά, αναφορική ακεραιότητα, ακεραιότητα πεδίου τιμών και περιορισμοί οριζόμενοι από τον χρήστη.
- 2.2. Να ορίσετε κατάλληλα ευρετήρια (indexes) για τους πίνακες της ΒΔ και να δικαιολογήσετε την επιλογή σας με βάση την χρησιμότητα τους για τα ερωτήματα στα οποία χρησιμοποιούνται.

Δημιουργία Πινάκων

Δημιουργία Triggers/Procedures

Μέρος Β: QUERIES ΕΡΩΤΗΜΑΤΩΝ

1. Βρείτε τα έσοδα του φεστιβάλ, ανά έτος από την πώληση εισιτηρίων, λαμβάνοντας υπόψη όλες τις κατηγορίες εισιτηρίων και παρέχοντας ανάλυση ανά είδος πληρωμής.
2. Βρείτε όλους τους καλλιτέχνες που ανήκουν σε ένα συγκεκριμένο μουσικό είδος με ένδειξη αν συμμετέχουν σε εκδηλώσεις του φεστιβάλ για το συγκεκριμένο έτος
3. Βρείτε ποιοι καλλιτέχνες έχουν εμφανιστεί ως warm up περισσότερες από 2 φορές στο ίδιο φεστιβάλ;
4. Για κάποιο καλλιτέχνη, βρείτε το μέσο όρο αξιολογήσεων (Ερμηνεία καλλιτεχνών) και εμφάνιση (Συνολική εντύπωση).
5. Βρείτε τους νέους καλλιτέχνες (ηλικία < 30 ετών) που έχουν τις περισσότερες συμμετοχές σε φεστιβάλ;
6. Για κάποιο επισκέπτη, βρείτε τις παραστάσεις που έχει παρακολουθήσει και το μέσο όρο της αξιολόγησης του, ανά παράσταση.
7. Βρείτε ποιο φεστιβάλ είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού;
8. Βρείτε το προσωπικό υποστήριξης που δεν έχει προγραμματισμένη εργασία σε συγκεκριμένη ημερομηνία;
9. Βρείτε ποιοι επισκέπτες έχουν παρακολουθήσει τον ίδιο αριθμό παραστάσεων σε διάστημα ενός έτους με περισσότερες από 3 παρακολουθήσεις;
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.
11. Βρείτε όλους τους καλλιτέχνες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον καλλιτέχνη με τις περισσότερες συμμετοχές σε φεστιβάλ.
12. Βρείτε το προσωπικό που απαιτείται για κάθε ημέρα του φεστιβάλ, παρέχοντας ανάλυση ανά κατηγορία (τεχνικό, προσωπικό ασφαλείας, βοηθητικό προσωπικό);
13. Βρείτε τους καλλιτέχνες που έχουν συμμετάσχει σε φεστιβάλ σε τουλάχιστον 3 διαφορετικές ηπείρους.
14. Βρείτε ποια μουσικά είδη είχαν τον ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές με τουλάχιστον 3 εμφανίσεις ανά έτος;
15. Βρείτε τους top-5 επισκέπτες που έχουν δώσει συνολικά την υψηλότερη βαθμολογία σε ένα καλλιτέχνη. (όνομα επισκέπτη, όνομα καλλιτέχνη και συνολικό σκορ βαθμολόγησης);

Dummy data

Μέρος Α: Σχεδιασμός και υλοποίηση Βάσης Δεδομένων

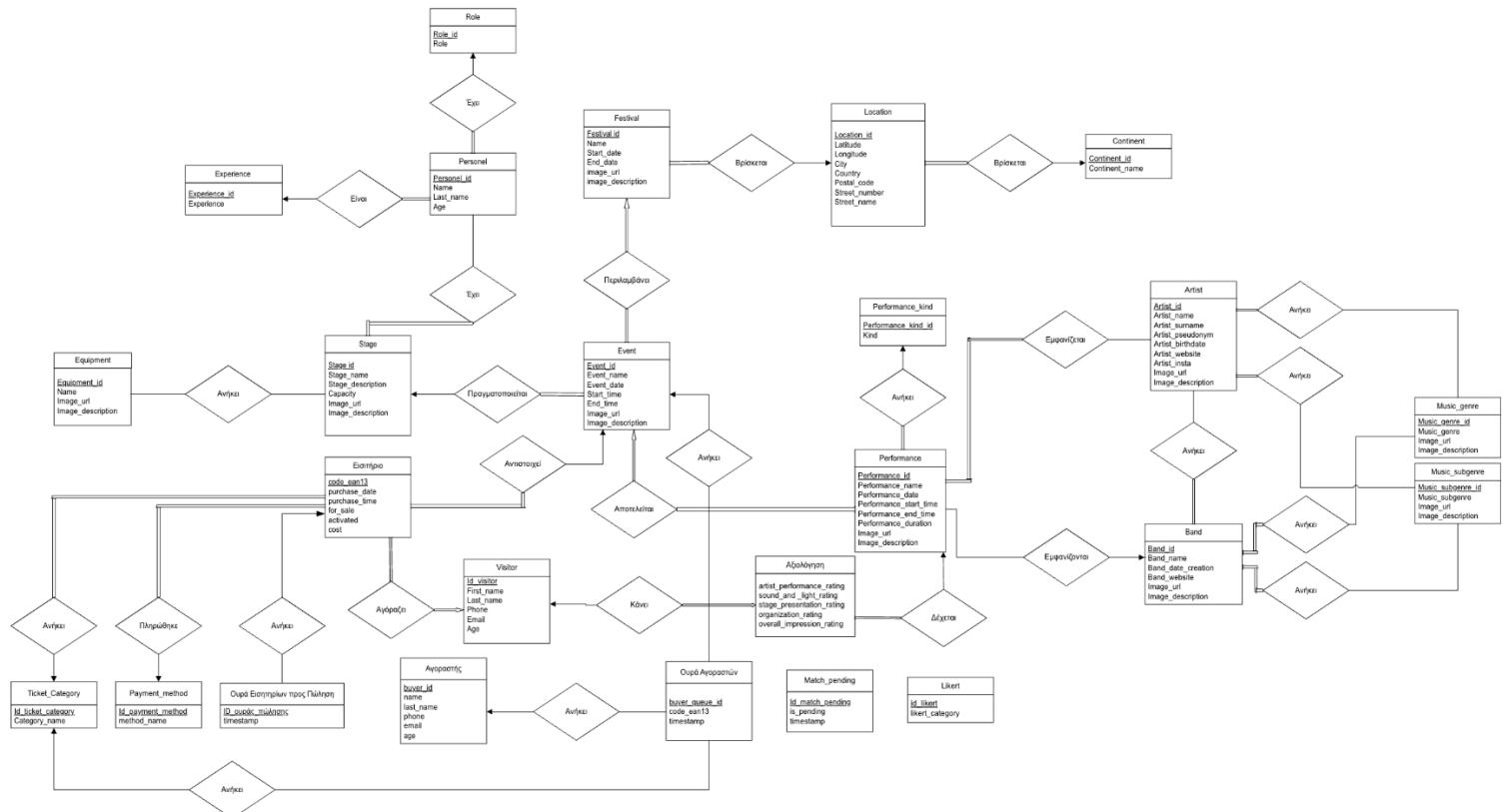
1. Να σχεδιάσετε το ER διάγραμμα που προκύπτει από την εκφώνηση.

2. Να σχεδιάσετε το σχεσιακό διάγραμμα και να αναπτύξετε την ΒΔ.

2.1 Να ορίσετε όλους τους απαραίτητους περιορισμούς που θα εξασφαλίζουν την ορθότητα της ΒΔ. Αυτοί είναι περιορισμοί ακεραιότητας, κλειδιά, αναφορική ακεραιότητα, ακεραιότητα πεδίου τιμών και περιορισμοί οριζόμενοι από τον χρήστη.

2.2. Να ορίσετε κατάλληλα ευρετήρια (indexes) για τους πίνακες της ΒΔ και να δικαιολογήσετε την επιλογή σας με βάση την χρησιμότητα τους για τα ερωτήματα στα οποία χρησιμοποιούνται.

1. ER DIAGRAM



Αρχικά, σχεδιάσαμε το διάγραμμα οντοτήτων συσχετίσεων, το οποίο φαίνεται παραπάνω, με τη βοήθεια του my.sharepoint.

Σχετικά με τη θεωρία:

- ❖ Με παραλληλόγραμμα απεικονίζονται τα entities (tables) και τα στοιχεία μέσα είναι τα attributes (columns). Ότι είναι υπογραμμισμένο πρόκειται για primary key.
- ❖ Οι ρόμβοι περιγράφουν τη σχέση του ενός entity με ένα άλλο. Π.χ. ένα event πραγματοποιείται σε ένα stage. Οι σχέσεις είναι πολύ σημαντικές ώστε να δημιουργηθεί η βάση ορθά.
- ❖ Οι γραμμές που συνδέουν τις οντότητες με τους ρόμβους (σχέσεις) εκφράζουν τη συμμετοχή των οντοτήτων σε κάθε σχέση, δηλαδή τον πληθικό περιορισμό (cardinality constraints). Ο πληθικός περιορισμός εκφράζεται από την ύπαρξη και τη φορά ενός βέλους. Δείχνει προς το one.

Στο διάγραμμα χρησιμοποιήσαμε:

One to many (βέλος προς τα αριστερά). Π.χ. η σχέση του stage με το event. Ένα stage μπορεί να έχει πολλά events, ενώ ένα event μπορεί να έχει το πολύ ένα stage.

Many to one (βέλος προς τα δεξιά). Π.χ. η σχέση του festival με το location. Ένα φεστιβάλ μπορεί να έχει το πολύ ένα location, ενώ ένα location μπορεί να έχει πολλά φεστιβάλ.

Many to many (κανένα βέλος). Π.χ. η σχέση του stage με το equipment. Ένα stage μπορεί να έχει πολλά equipment και ένα equipment μπορεί να έχει πολλά stages

Επίσης, η διπλή γραμμή εκφράζει το total participation και η λεπτή το partial participation.

Η ολική συμμετοχή (total participation) δηλώνει ότι όλες οι εγγραφές μιας οντότητας πρέπει να συνδέονται με τουλάχιστον μία εγγραφή στη σχέση.

Η μερική συμμετοχή (partial participation) δηλώνει ότι ορισμένες οντότητες ενδέχεται να μην συμμετέχουν καθόλου σε καμία σχέση του συγκεκριμένου συνόλου σχέσεων.

Π.χ. το performance έχει total participation με τους καλλιτέχνες, δηλαδή, ένα performance δεν μπορεί να μείνει χωρίς καλλιτέχνη. Αντιθέτως, η συμμετοχή του καλλιτέχνη στο performance είναι partial, δηλαδή ένας καλλιτέχνης μπορεί να μην έχει καμια εμφάνιση

Όλες οι σχέσεις του φαίνονται στο διάγραμμα εξηγούνται αναλυτικά στην υλοποίηση των πινάκων και των triggers/procedures.

2. Relational Diagram

Παρακάτω παρουσιάζεται το relational diagram, το οποίο παράχθηκε αυτόματα μέσω του εργαλείου MySQL Workbench με βάση το σχήμα της βάσης δεδομένων που υλοποιήσαμε. Σκοπός ενός relational diagram είναι να απεικονίσει γραφικά τις σχέσεις μεταξύ των πινάκων, τις πρωτεύοντα και ξένα κλειδιά, τους τύπους των

δεδομένων, τα constraints, καθώς και τους τύπους των συσχετίσεων που έχουν καθοριστεί (π.χ. one-to-many, many-to-many), βοηθώντας έτσι στην καλύτερη κατανόηση της δομής του συστήματος.



Στο διάγραμμα, κάθε ορθογώνιο πλαίσιο αντιστοιχεί σε έναν πίνακα της βάσης. Τα πεδία εντός κάθε πίνακα εμφανίζονται με διαφορετικά εικονίδια:

- Ένα κλειδί δίπλα σε ένα πεδίο δηλώνει ότι πρόκειται για primary key
- Τα σημεία με τα οποία ενώνονται οι πίνακες είναι τα foreign keys
- Οι γραμμές που συνδέουν πίνακες αντιπροσωπεύουν τις σχέσεις μεταξύ τους.
 - Μία γραμμή με ένα πολλαπλό πόδι (crow's foot) στη μία άκρη και ένα κάθετο τικ (|) στην άλλη υποδηλώνει μια σχέση one-to-many
 - Η αντίθετη της υποδηλώνει μια σχέση many-to-one

Ακόμα, αξίζει να σημειωθεί πως σε αυτό το διάγραμμα φαίνονται και οι ενδιάμεσοι πίνακες για τις σχέσεις many-to-many.

Τέλος, παρατηρούμε ότι το relational diagram που προέκυψε αντιστοιχεί πλήρως στο αρχικό ER διάγραμμα που σχεδιάστηκε, τόσο ως προς τις οντότητες όσο και ως προς τις σχέσεις μεταξύ τους. Αυτό επιβεβαιώνει ότι η υλοποίησή μας σε SQL αντικατοπτρίζει πιστά τον εννοιολογικό σχεδιασμό, κάτι που αποτελεί ένδειξη σωστού και ολοκληρωμένου σχεδιασμού της βάσης δεδομένων.

2.1. Να ορίσετε όλους τους απαραίτητους περιορισμούς που θα εξασφαλίζουν την ορθότητα της ΒΔ. Αυτοί είναι περιορισμοί ακεραιότητας, κλειδιά, αναφορική ακεραιότητα, ακεραιότητα πεδίου τιμών και περιορισμοί οριζόμενοι από τον χρήστη.

- ❖ **Περιορισμοί Ακεραιότητας**
Οι περιορισμοί ακεραιότητας φροντίζουν να διατηρούν τα δεδομένα σωστά και συνεπή.
Είναι όλα τα κλειδιά (primary keys) που έχουμε αξιοποιήσει, όπως επίσης περιορισμοί μοναδικότητας, όπως το UNIQUE. Επίσης, ο περιορισμός NOT NULL που έχουμε τοποθετήσει σε πολλά attributes φροντίζει για την πληρότητα των δεδομένων μας.
- ❖ **Κλειδιά**
τα primary keys που χρησιμοποιήσαμε
- ❖ **Αναφορική Ακεραιότητα**
τα foreign keys με τα οποία συνδέουμε πίνακες με τη σχέση one-to-many
- ❖ **Ακεραιότητα Πεδίου Τιμών**
Στην κατηγορία αυτή συμπεριλαμβάνονται οι περιορισμοί NOT NULL και όποια CHECK έχουμε βάλει πχ για την ώρα ή το πεδίο τιμών που μπορεί να πάρει ένα attribute.
- ❖ **Περιορισμοί οριζόμενοι από τον χρήστη**
Οι περιορισμοί αυτοί είναι όλα τα triggers και τα procedures τα οποία φτιάξαμε, για να είμαστε σίγουροι ότι η βάση μας λειτουργεί σωστά, στα πλαίσια της εκφώνησης και τα insert μας δεν παραβιάζουν βασικούς περιορισμούς.

2.2. Να ορίσετε κατάλληλα ευρετήρια (indexes) για τους πίνακες της ΒΔ και να δικαιολογήσετε την επιλογή σας με βάση την χρησιμότητα τους για τα ερωτήματα στα οποία χρησιμοποιούνται.

Τα indexes είναι μια ειδική δομή δεδομένων, σαν πίνακας περιεχομένων που επιταχύνουν την αναζήτηση σε μία βάση δεδομένων. Μπορούμε δηλαδή να βρούμε στοιχεία και αποτελέσματα πολύ πιο γρήγορα, χωρίς να χρειάζεται να διατρέχουμε κάθε γραμμή.

Μας ζητείται να δημιουργήσουμε indexes παρόλα αυτά παρατηρούμε ότι κατά τη δημιουργία της βάσης μας ήδη κάποια indexes δημιουργούνται αυτόματα. Αυτά είναι indexes των attributes που είναι primary keys, foreign keys ή έχουν UNIQUE constraint.

Επειδή λοιπόν, πολλά indexes και μάλιστα αυτά που χρησιμοποιούνται συχνότερα στα joins στα queries μας (ως foreign και primary keys κυρίως) είναι ήδη έτοιμα θεωρήσαμε ότι δεν μας χρειάζεται να φτιάξουμε κάποιο επιπλέον.

Ενδεικτικά παραθέτουμε:

SHOW INDEX FROM performance;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶ performance	0	PRIMARY	1	performance_id	A	0	NULL	NULL		BTREE		
▶ performance	1	event_id	1	event_id	A	0	NULL	NULL		BTREE		
▶ performance	1	performance_kind_id	1	performance_kind_id	A	0	NULL	NULL		BTREE		
▶ performance	1	band_id	1	band_id	A	0	NULL	NULL	YES	BTREE		
▶ performance	1	stage_id	1	stage_id	A	0	NULL	NULL	YES	BTREE		

SHOW INDEX FROM event;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶ event	0	PRIMARY	1	event_id	A	0	NULL	NULL		BTREE		
▶ event	0	event_name	1	event_name	A	0	NULL	NULL		BTREE		
▶ event	1	festival_id	1	festival_id	A	0	NULL	NULL		BTREE		
▶ event	1	stage_id	1	stage_id	A	0	NULL	NULL		BTREE		

SHOW INDEX FROM visitor;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶ visitor	0	PRIMARY	1	id_visitor	A	0	NULL	NULL		BTREE		
visitor	0	first_name	1	first_name	A	0	NULL	NULL		BTREE		
visitor	0	first_name	2	last_name	A	0	NULL	NULL		BTREE		
visitor	0	first_name	3	phone	A	0	NULL	NULL		BTREE		

Όπως βλέπουμε πράγματι δημιουργούνται τα indexes που αναφέραμε και μάλιστα δηλώνεται και το index_type τους.

Δημιουργία Πινάκων

1. Δημιουργούμε τον πίνακα **continent** στον οποίο έχουμε primary key το continent_id με auto_increment, για να αυξάνεται κάθε φορά κατά 1. Λειτουργεί ως look-up πινάκας για να βεβαιωθούμε ότι είναι σωστές οι ήπειροι. Το name είναι VARCHAR NOT NULL, εξασφαλίζοντας ότι δε θα μείνει κάποια column κενό. Έχουμε βάλει και UNIQUE, ώστε να μην μπει η ίδια ήπειρος 2 φορές
2. Δημιουργούμε τον πίνακα **location**. Βάζουμε primary key το location_id με auto_increment και NOT NULL, έτσι όλα τα locations θα έχουν σίγουρα ID. Βάλαμε UNIQUE (latitude,longitude) ώστε να εξασφαλίσουμε πως δε θα μπει το ίδιο location 2 φορές. Τέλος, βάλαμε foreign key το continent id, ώστε να μπορεί να έχει και τις ήπειρους. Βάλαμε NOT NULL, καλύπτοντας ότι ένα festival δε θα μείνει χωρίς ήπειρο (total participation).
3. Δημιουργούμε τον πίνακα **festival**. Πάλι έχουμε primary key το festival id NOT NULL AUTO_INCREMENT. Βάλαμε image url VARCHAR με CHECK ότι θα είναι https:// ,

όπως και description με VARCHAR. Βάλαμε foreign key το location id INT NOT NULL, ώστε το φεστιβάλ να βλέπει το location και με το NOT NULL εξασφαλίζουμε ότι δε θα μείνει festival χωρίς location. Επίσης, με το foreign key καλύπτεται η σχέση many to one, δηλαδή ότι ένα φεστιβάλ μπορεί να πάρει το πολύ μια τοποθεσία, ενώ μια τοποθεσία μπορεί να πάρει πολλά φεστιβάλ.

4. Δημιουργούμε τον πίνακα **stage**. Βάζουμε primary key stage_id με AUTO_INCREMENT και στο stage name βάζουμε UNIQUE ώστε να μην υπάρχουν δυο stages με το ίδιο όνομα. Όλα τα άλλα είναι not null. Έχουμε βάλει εικόνα και λεκτική περιγραφή για τη σκηνή.
5. Δημιουργούμε τον πίνακα **event**. Βάζουμε primary key το event id με not null και auto_increment οπως κάθε φορά και όλα τα υπόλοιπα στοιχεία είναι not null. Βάζουμε foreign key το festival id. Έτσι, καλύπτουμε ότι τα events έχουν το πολύ ένα φεστιβάλ, ενώ τα φεστιβάλ μπορούν να έχουν πολλά events (many to one). Αντίστοιχα, βάζουμε foreign key το stage_id. Έτσι καλύπτουμε ότι ένα event γίνεται το πολύ σε μια σκηνή, ενώ μια σκηνή μπορεί να πάρει πολλά events (όχι ταυτοχρόνα φυσικά). Με το int not null στα foreign keys εξασφαλίζουμε ότι ένα event δεν μπορεί να μείνει χωρίς stage και χωρίς φεστιβάλ (total participation). Επίσης, έχουμε βάλει εικόνα και λεκτική περιγραφή για το event.
6. Δημιουργούμε τον πίνακα **equipment**. Έχει primary key equipment id, το name του equipment είναι not null και unique, ώστε να μην έχουμε τον ίδιο εξοπλισμό δύο φορες π.χ. ηχεία, φώτα, μικρόφωνα, κονσόλες, ειδικά εφέ, κ.λπ.). Επισης, βάλαμε εικόνα και λεκτική περιγραφή.
7. Δημιουργούμε τον πίνακα **stage_equipment**. Χρειαζόμαστε ξεχωριστό πίνακα, για να τηρηθεί η σχέση many to many. Δηλαδή, μια σκηνή μπορεί να έχει πολλούς εξοπλισμούς, αλλά και ένας εξοπλισμός μπορεί να έχει πολλές σκηνές. Πέρα από το ότι οι εξοπλισμοί είναι πολλοί και άρα χρειαζόντουσαν δικό τους lookup πινακα, η many to many δεν υλοποιείται διαφορετικά, π.χ. με foreign key equipment στο stage, καθώς αυτό δημιουργεί αναγκαστικά one to many συνθήκη, η οποία δεν είναι δεκτή. Έτσι, το stage_equipment έχει απλα τα foreign keys stage id και equipment id με NOT NULL, και ως primary key είναι το ζεύγος (stage_id, equipment_id), αφού κάθε τέτοια σχέση είναι μοναδική.
8. Δημιουργούμε τον πίνακα **role**. Αφορά την ιδιότητα του προσωπικού, αν είναι τεχνικό, βιοηθητικό ή ασφαλείας. Βάλαμε role id ως primary key και role VARCHAR NOT NULL UNIQUE, ώστε να μην μπει ο ίδιος ρόλος δύο φορές.
9. Δημιουργούμε τον πίνακα **experience**, ο οποίος αφορά τον βαθμό εμπερίας του προσωπικού. Πάλι βάζουμε primary key το experience id και στο experience βάζουμε VARCHAR NOT NULL UNIQUE.
10. Δημιουργούμε τον πίνακα **personnel**. Primary key είναι το personnel id με not null και auto_increment. Όλα τα άλλα πεδία είναι not null. Βάλαμε foreign key experience id και role id. Έτσι, βεβαιωνόμαστε ότι κάθε personnel έχει το πολύ ένα βαθμό εμπειρίας και έναν ρόλο. Με το not null εξασφαλίζουμε ότι δε θα μείνει personnel χωρίς βαθμό

εμπειρίας και ρόλο. Κάναμε την παραδοχή ότι το personnel μπορει να έχει μόνο έναν ρόλο, έτσι έχουμε συνθήκη many to one.

11. Φτιάχνουμε τον πίνακα **stage_personel**. Πάλι, όπως ο stage_equipment, χρειάζεται για να τηρηθεί η συνθήκη many to many, δηλαδή ότι ένα stage έχει πολλούς personnel, αλλά και ότι ένα personnel μπορει να έχει πολλά stages. Βάζουμε foreign keys τα stage id και personnel id και ως primary key τον συνδυασμό των δυο, δηλαδή (stage id, personnel id).
12. Δημιουργούμε τον πίνακα **performance_kind**, στον οποίο αποθηκεύουμε ξεχωριστά τα είδη των εμφανίσεων του φεστιβάλ. Το primary key είναι AUTO_INCREMENT, άρα αυξάνεται κάθε φορά αυτόματα κατά 1 και φτάνει μέχρι και τον αριθμό που δηλώνει το πλήθος των διαφορετικών ειδών εμφάνισης. Ο πίνακας αυτός λειτουργεί ως look up πίνακας στο performance. Ένα performance_kind μπορεί να συνδέεται με πολλά performances και είναι UNIQUE, άρα στον πίνακα δεν θα αποθηκευτεί δύο φορές το είδος πχ. "warm-up".
13. Δημιουργούμε τον πίνακα **band**, για τα συγκροτήματα τα οποία εμφανίζονται στο φεστιβάλ. Το κάθε συγκρότημα έχει δικό του id AUTO_INCREMENT και για να βεβαιωθούμε ότι δεν αποθηκεύεται στον πίνακα δύο φορές το ίδιο band, βάζουμε τον έλεγχο UNIQUE, με συνδυασμό δύο attributes: του band_name και του band_date_creation, τα οποία είναι NOT NULL και θεωρήσαμε ότι συνδυαστικά εξασφαλίζουν μοναδικότητα. Επίσης, προσθέσαμε εικόνα και λεκτική περιγραφή και με ενα CHECK βεβαιωθήκαμε ότι η εικόνα θα δίνεται σίγουρα με τη μορφή "<https...>".
14. Δημιουργούμε τον πίνακα **performance**. Βάζουμε primary key auto increment και όλες τις πληροφορίες σχετικά με το πότε λαμβάνει χώρα το performance (name,date,start-end time, duration) NOT NULL, ώστε να εκτελούμε τους κατάλληλους ελέγχους και να έχουμε τις αναγκαίες πληροφορίες. Επίσης, ελέγχουμε το start-time να είναι προφανώς πριν το end-time.

Από το σημείο αυτό και μετά ορίζουμε τις σχέσεις των πινάκων που συνδέονται με το performance. Αρχικά, με foreign key το event δηλώνουμε τη σχέση one-to-many του event-performance. Άρα ένα event μπορεί να έχει πολλά performances και όχι το αντίθετο. Επίσης ένα performance δεν γίνεται να μην ανήκει σε event και για αυτό το λόγο δηλώνουμε στο performance το event_id NOT NULL.

Δηλώνουμε τη σχέση one-to-many του performance_kind-performance, βάζοντας στον πίνακα performances foreign key το performance_kind id. Μάλιστα, δεν θέλουμε ένα performance να μείνει χωρίς kind άρα και εδώ το δηλώνουμε ως not null. Σημαντικό είναι στον πίνακα performance να μην δηλώσουμε κάποιο performance kind id που δεν υπάρχει και για αυτό βάζουμε ένα CHECK να είναι το id μεταξύ του 1 και του 7, που είναι το πλήθος των performance_kind εισαγωγών που έχουμε.

Δηλώνουμε τη σχέση one-to-many του band-performance, με foreign key το band_id στον πίνακα performance. Το band μπορεί να είναι NULL στον πίνακα performance, γιατί μπορεί σε ένα performance να παίζει ένας solo καλλιτέχνης και όχι απαραίτητα κάποιο band.

Τέλος, δηλώνουμε τη σχέση one-to-many του stage-performance, χρησιμοποιώντας το stage από το event στο οποίο είναι δηλωμένο. Άρα δεν συνδέουμε τους πίνακες performance και stage, αλλά τραβάμε το stage μέσω του event το οποίο ήδη είναι συνδεδεμένο με το performance. Αυτό το κάνουμε γιατί η εκφώνηση ζητάει ρητά κάθε εμφάνιση να περιέχει πληροφορίες για την παράσταση στην οποία πραγματοποιείται και γιατί με αυτό τον τρόπο μπορούμε να διασφαλίσουμε ότι ένα performance, το οποίο ανήκει σε ένα event, το οποίο γίνεται σε μία συγκεκριμένη σκηνή ΔΕΝ θα γίνεται σε άλλη σκηνή από ότι το event.

15. Δημιουργούμε τον πίνακα **artist**. Έχει primary key auto_increment και not null τα βασικά στοιχεία του. Για να φροντίσουμε να μην γίνει inserted δύο φορές ο ίδιος artist βάζουμε UNIQUE περιορισμό, τον συνδυασμό ονόματος, επίθετο και γενέθλια, γιατί πάλι θεωρούμε ότι αρκεί. Ο artist συνδέεται με το performance πίνακα με τη σχέση many-to-many.
16. Εδώ πάλι δημιουργούμε νέο πίνακα **performance_artist**, για να δηλώσουμε τη σχέση αυτή. Ένα performance μπορεί να έχει πολλούς artists (στην περίπτωση του band που θα το δούμε στη συνέχεια) και ένας artist μπορεί να παίζει σε πολλά performances. Στον πίνακα **performance_artist** λοιπόν, δηλώνουμε σαν primary keys τον συνδυασμό των foreign keys artist και performance id και κάνουμε insert εκεί ποιοι καλλιτέχνες εμφανίζονται που.
17. Δημιουργούμε επίσης τον πίνακα **artist_band** για να δηλώσουμε τη σχέση many-to-many μεταξύ των δύο πινάκων. Οι δύο αυτοί πίνακες συνδέονται ως εξής. Ένας καλλιτέχνης μπορεί να ανήκει και σε πάνω από ένα συγκροτήμα, αλλά δεν χρειάζεται και απαραίτητα να ανήκει σε συγκρότημα, (άρα δεν έχουμε total participation). Ένα συγκρότημα όμως πρέπει οπωσδήποτε να έχει πολλούς καλλιτέχνες (total participation). Με τον πίνακα αυτόν λοιπόν, ενώνουμε ποιος καλλιτέχνης ανήκει σε ποιο band και για αυτό βάζουμε σαν primary key πάλι τον συνδυασμό των foreign keys artist και band id.
18. Δημιουργούμε τον πίνακα **music_genre**, ο οποίος θα λειτουργήσει σαν look up πίνακας για τους artist και band. Θέλουμε δηλαδή στον artist και στο band να ορίσουμε το είδος μουσικής του καθενός και φτιάχνουμε αντίστοιχο πίνακα που αποθηκεύει όλα τα είδη μουσικής που μπορούν να αντιστοιχηθούν με τους πίνακες αυτούς. Ορίζουμε primary key AUTO_INCREMENT και unique περιορισμό για να περιέχει σίγουρα το όνομα του είδους από μία φορά (no double entries). Επίσης, βάζουμε φωτογραφία και λεκτική περιγραφή με παρόμοιο τρόπο με πριν.

19. Δημιουργούμε τον πίνακα **music_subgenre**, με ακριβώς ίδιο τρόπο με τον προηγούμενο ορίζοντας και πάλι primary key auto increment.
20. Αυτό που θέλουμε τώρα είναι να συνδέσουμε τους music_genre και music_subgenre πίνακες με τους artist και bands. Η σχέση μεταξύ αυτών των πινάκων είναι many-to-many, δηλαδή ένας artist μπορεί να έχει πολλά music_genre και πολλά music_subgenre. Αντίστοιχα, ένα music_genre και ένα music_subgenre μπορούν να συνδέονται με πολλούς artists. Ομοίως και για τον πίνακα band. Άρα δημιουργούμε 4 νέους πίνακες τους εξής:
- **artist_genre**
 - **artist_subgenre**
 - **band_genre**
 - **band_subgenre**
- Και στους 4 ορίζουμε σαν primary keys τον συνδυασμό των 2 κάθε φορά αντίστοιχων foreign keys.
21. Δημιουργούμε τον πίνακα **ticket_category**. Έχει primary key το id_ticket_category με AUTO_INCREMENT, ώστε κάθε κατηγορία εισιτηρίου να έχει μοναδικό αναγνωριστικό. Το πεδίο category_name είναι VARCHAR(30) και ορίζεται NOT NULL και UNIQUE, ώστε να εξασφαλίσουμε ότι δεν θα υπάρχει κατηγορία χωρίς όνομα και ότι δεν θα καταχωρηθεί η ίδια κατηγορία παραπάνω από μία φορά. Ο πίνακας αυτός λειτουργεί ως πίνακας αναφοράς (lookup) και σχετίζεται με τον πίνακα ticket, όπου το id_ticket_category χρησιμοποιείται ως foreign key, διαμορφώνοντας έτσι μία σχέση many-to-one: πολλά εισιτήρια ανήκουν σε μία κατηγορία. Η δήλωση NOT NULL στο ticket καλύπτει το total participation, διασφαλίζοντας ότι κανένα εισιτήριο δεν θα μείνει χωρίς κατηγορία.
22. Δημιουργούμε τον πίνακα **payment_method**, ο οποίος αποτελεί επίσης lookup πίνακα, που συνδέεται με τον πίνακα ticket μέσω της σχέσης ticket – payment_method (many-to-one). Το id_payment_method είναι primary key (AUTO_INCREMENT), και το method_name είναι UNIQUE και NOT NULL, ώστε να καταγράφονται μόνο έγκυροι και διακριτοί τρόποι πληρωμής (π.χ. Πιστωτική, Χρεωστική, Τραπεζικός Λογαριασμός).
23. Δημιουργούμε τον πίνακα **visitor**, στον οποίο έχουμε primary key το id_visitor με AUTO_INCREMENT. Ο πίνακας περιλαμβάνει τα πεδία first_name, last_name, phone, email, age, με τα τρία πρώτα να είναι NOT NULL. Το phone είναι CHAR(10) και περιλαμβάνει CHECK ώστε να έχει ακριβώς 10 ψηφία, διασφαλίζοντας εγκυρότητα τηλεφώνου. Ορίζεται UNIQUE(first_name, last_name, phone) ώστε να αποφευχθεί η πολλαπλή καταχώρηση του ίδιου επισκέπτη.

24. Δημιουργούμε τον πίνακα **ticket** που περιέχει τις πληροφορίες εισιτηρίων. Το `code_ean13` χρησιμοποιείται ως primary key και είναι τύπου BIGINT, με CHECK ώστε να έχει ακριβώς 13 ψηφία. Περιέχει τα πεδία `purchase_date`, `purchase_time`, `for_sale`, `activated`, `cost`, `id_ticket_category`, `id_payment_method`, `event_id` και `id_visitor`. Τα πεδία `for_sale` και `activated` είναι TINYINT(1) με CHECK ώστε να δέχονται μόνο τιμές 0 ή 1 (boolean) και έχουν σκοπό να δείχνουν πότε ένα εισιτήριο έχει μπει στην ουρά μεταπώλησης και πότε έχει ενεργοποιηθεί αντίστοιχα. Ακόμα, υπάρχει unigue περιορισμός στο (`id_visitor`, `event_id`), διασφαλίζοντας ότι ένας επισκέπτης δεν μπορεί να αγοράσει περισσότερα από ένα εισιτήρια για την ίδια παράσταση. Το εισιτήριο συνδέεται με `visitor`, `event`, `ticket_category` και `payment_method` μέσω foreign keys, διαμορφώνοντας τέσσερις many-to-one σχέσεις. Όλα τα foreign keys είναι NOT NULL, επομένως έχουμε total participation, δηλαδή κανένα εισιτήριο δεν μπορεί να υπάρχει χωρίς πλήρη πληροφορία για το σε ποιον ανήκει, για ποιο `event` είναι, ποιας κατηγορίας είναι και με ποιον τρόπο πληρώθηκε.
25. Δημιουργούμε τον πίνακα **buyer**, που περιέχει τα στοιχεία των υποψήφιων αγοραστών εισιτηρίων μεταπώλησης. Έχει primary key το `buyer_id` με AUTO_INCREMENT και τα πεδία `name`, `last_name`, `phone`, `email`, `age`, όλα NOT NULL. Το `phone` έχει CHECK για 10 χαρακτήρες, και το UNIQUE(`name`, `last_name`, `phone`) αποτρέπει την πολλαπλή εγγραφή ίδιου αγοραστή.
26. Δημιουργούμε τον πίνακα **buyer_queue**, που καταγράφει το ενδιαφέρον αγοραστών για εισιτήρια που βρίσκονται ή πρόκειται να μπουν σε μεταπώληση. Έχει primary key το `buyer_queue_id` με AUTO_INCREMENT και foreign keys προς `event`, `ticket_category` και `buyer`. Έχει timestamp με DEFAULT NOW() για την υλοποίηση FIFO ουράς. Χρησιμοποιείται CHECK για την εγκυρότητα του κωδικού EAN 13 και UNIQUE(`buyer_id`, `event_id`) και UNIQUE(`buyer_id`, `code_ean13`), ώστε να μην υπάρχει διπλό ενδιαφέρον για το ίδιο εισιτήριο ή event από τον ίδιο αγοραστή. Οι σχέσεις που δημιουργούνται είναι many-to-one, και το NOT NULL στα foreign keys καλύπτει την total participation, εξασφαλίζοντας ότι κάθε εγγραφή ενδιαφέροντος θα έχει έγκυρο αγοραστή και αναφορά είτε σε `event` είτε σε εισιτήριο.
27. Δημιουργούμε τον πίνακα **resell_queue**, ο οποίος καταγράφει τα εισιτήρια που τίθενται προς μεταπώληση. Έχει primary key το `resell_queue_id` με AUTO_INCREMENT και πεδίο `code_ean13`, το οποίο είναι FOREIGN KEY προς το `ticket`, με CHECK για 13ψήφια τιμή. Το timestamp καταγράφει τη χρονική στιγμή εισαγωγής του εισιτηρίου στην ουρά, με DEFAULT NOW(). Η χρήση NOT NULL στα πεδία και CHECK περιορισμών διασφαλίζει ακεραιότητα και μοναδικότητα.

28. Δημιουργούμε τον πίνακα **Likert**, στον οποίο αποθηκεύονται οι κατηγορίες αξιολόγησης σύμφωνα με την κλίμακα Likert. Έχει primary key το id_liker με AUTO_INCREMENT, και το likert_category είναι VARCHAR(30) NOT NULL UNIQUE, ώστε να εξασφαλιστεί η μοναδικότητα κάθε βαθμίδας. Παρόλο που δεν έχει foreign key, λειτουργεί ως βοηθητικός πίνακας για την επέκταση του συστήματος αξιολόγησης και αξιοποιείται στο αντίστοιχο view.
29. Δημιουργούμε τον πίνακα **evaluation**, που καταγράφει τις αξιολογήσεις των επισκεπτών για εμφανίσεις. Χρησιμοποιούμε σύνθετο primary key (id_visitor, performance_id) για να διασφαλίσουμε ότι κάθε επισκέπτης μπορεί να αξιολογήσει κάθε εμφάνιση μόνο μία φορά. Περιλαμβάνει πέντε TINYINT(1) πεδία, όλα NOT NULL με CHECK για τιμές από 1 έως 5: artist_performance_rating, sound_and_light_rating, stage_presentation_rating, organization_rating, και overall_impression_rating. Οι σχέσεις που δημιουργούνται είναι many-to-one: evaluation – visitor και evaluation – performance, και τα foreign keys είναι NOT NULL, καλύπτοντας το total participation και για τις δύο οντότητες.
30. Δημιουργούμε τον πίνακα **match_pending**, που χρησιμοποιείται ως βοηθητική δομή για την παρακολούθηση εκκρεμοτήτων στη μεταπώληση. Δεν συνδέεται με άλλους πίνακες μέσω foreign keys, αλλά έχει συγκεκριμένη λειτουργία: περιλαμβάνει ένα μοναδικό row με id = 1 (primary key) και πεδία is_pending και last_checked, που χρησιμεύουν ως σημαία συστήματος για τον έλεγχο αυτόματων συναλλαγών. Δεν υπάρχουν επιπλέον περιορισμοί, αλλά η ύπαρξή του είναι κρίσιμη για την υλοποίηση της FIFO λογικής μεταπώλησης.

Δημιουργία Triggers/Procedures

Τα **triggers** είναι ειδικές αποθηκευμένες εντολές SQL που εκτελούνται αυτόμata όταν συμβαίνει ένα συγκεκριμένο γεγονός (event) σε έναν πίνακα. Ενεργοποιούνται πριν ή μετά από insert, delete ή edit μιας γραμμής ενός πίνακα.

Τα **stored procedures** είναι ένα σύνολο SQL εντολών που αποθηκεύονται μία φορά στη βάση και μπορούν να εκτελούνται όσες φορές χρειαστεί, σαν "συνάρτηση". Τις καλούμε με την εντολή CALL. Άρα, σε αντίθεση με τα triggers, δεν γίνονται αυτόματα. Επειδή θέλαμε όμως να αυτοματοποιηθεί παραπάνω η βάση μας, πολλά procedures καλούνται από μόνα τους, μέσω ενός event, κάθε 6 μήνες. Έτσι, δε χρειάζεται ο χρήστης να τα τρέχει χειροκίνητα. Αν θέλει πιο συχνά μπορεί να το αλλάξει.

Σε κάθε trigger ή procedure γράφουμε στην αρχή DELIMITER // και στο τέλος // DELIMITER ; . Με το delimiter αλλάζουμε τον χαρακτήρα που χρησιμοποιείται για να τερματίζουν οι εντολές. Έτσι, μπορούμε να εκτελέσουμε ένα μεγάλο block εντολών, διαφορετικά, αν βάλουμε σε κάθε εντολή ; ο sql parser μπερδεύεται. Με το delimiter, δεν εκτελεί τις εντολές μέχρι να δει //. Έπειτα επαναφέρουμε το delimiter σε ;

- Ελέγχουμε τα events να μην κάνουν overlap με άλλο event στο ίδιο stage
Δημιουργούμε trigger που ελέγχει μια γραμμή πριν προστεθεί στον πίνακα event

```

DELIMITER //
CREATE TRIGGER check_event_overlap
BEFORE INSERT ON event
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1 FROM event
        WHERE stage_id = NEW.stage_id
        AND event_date = NEW.event_date
        AND (
            (NEW.start_time BETWEEN start_time AND end_time)
            OR
            (NEW.end_time BETWEEN start_time AND end_time)
            OR
            (start_time BETWEEN NEW.start_time AND NEW.end_time)
            OR
            (end_time BETWEEN NEW.start_time AND NEW.end_time)
        )
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Time slot overlaps with existing event on this stage.' ;
    END IF;
END;
//
```

Πολύ απλά, ελέγχει αν στον πίνακα event υπάρχει ήδη αυτό το stage id και η ίδια ημερομηνία για το event. Έπειτα, συγκρίνει τις ώρες για να δει αν υπάρχει επικάλυψη (4 περιπτώσεις) . Αν υπάρχει επικάλυψη τότε πετάει σφάλμα SIGNAL SQLSTATE '45000' και το αντίστοιχο error message. Έτσι, διακόπτεται η insert και το νέο event δε θα αποθηκευτεί στη βάση.

- Δεν επιτρέπεται ένα event να ακυρωθεί.

```

DELIMITER //
CREATE TRIGGER prevent_event_delete
BEFORE DELETE ON event
FOR EACH ROW
BEGIN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Δεν επιτρέπεται η διαγραφή παράστασης.' ;
END;
//
```

Για αυτό βάλαμε trigger που απαγορεύει τη διαγραφή ενός event. Πριν κάνει delete πετάει error.

3. Το ίδιο και για το φεστιβάλ. Ένα φεστιβάλ δεν μπορεί να ακυρωθεί

```
DELIMITER //
CREATE TRIGGER prevent_festival_delete
BEFORE DELETE ON festival
FOR EACH ROW
BEGIN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Δεν επιτρέπεται η διαγραφή φεστιβάλ.';
END;
//
DELIMITER ;
```

4. Ελέγχουμε αν το προσωπικό ασφαλείας καλύπτει το 5% του capacity του stage και το βοηθητικό το 2%. Το υλοποιήσαμε αναγκαστικά με procedure, καθώς πρέπει πρώτα να ανατεθεί το προσωπικό και μετά να ελεγχουμε αν είναι αρκετό, ενώ το trigger μας υποχρεώνει να ελέγχουμε μόνο μια γραμμή, το οποίο προφανώς οδηγεί σε σφάλματα.

```
CREATE PROCEDURE check_stage_personel(IN p_stage_id INT)
BEGIN
    DECLARE security_count INT DEFAULT 0;
    DECLARE helper_count INT DEFAULT 0;
    DECLARE stage_capacity INT DEFAULT 0;
    DECLARE stage_name2 VARCHAR(50);
    DECLARE err_msg TEXT;

    -- Προσωπικό ασφαλείας (role_id = 3)
    SELECT COUNT(*) INTO security_count
    FROM stage_personel sp
    JOIN personnel p ON sp.personnel_id = p.personnel_id
    WHERE sp.stage_id = p_stage_id AND p.role_id = 3;

    -- Βοηθητικό προσωπικό (role_id = 2)
    SELECT COUNT(*) INTO helper_count
    FROM stage_personel sp
    JOIN personnel p ON sp.personnel_id = p.personnel_id
    WHERE sp.stage_id = p_stage_id AND p.role_id = 2;

    -- Χωρητικότητα σκηνής + όνομα
    SELECT capacity, stage_name
    INTO stage_capacity, stage_name2
    FROM stage
    WHERE stage_id = p_stage_id;
```

Το procedure δέχεται ως όρισμα το stage id της σκηνής την οποία θα ελέγξει.

Αρχικοποιούνται μεταβλητές ο οποίες θα κρατήσουν: αριθμό security και βοηθών ,τη χωρητικότητα της σκηνής, το όνομά της και το πιθανό μήνυμα σφάλματος.

Μέσω join, για να έχουμε πρόσβαση στους ρόλους του προσωπικού βρίσκουμε πόσοι security και helper έχει η συγκεκριμένη σκηνή.

Μετά, αποθηκεύουμε την χωρητικότητα και το όνομα της σκηνής.

```
-- Έλεγχος για προσωπικό ασφαλείας
IF security_count < stage_capacity * 0.05 THEN
    SET err_msg = CONCAT('Stage "', stage_name2, '" has insufficient security staff (needs ≥5%)');
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
END IF;

-- Έλεγχος για βοηθητικό προσωπικό
IF helper_count < stage_capacity * 0.02 THEN
    SET err_msg = CONCAT('Stage "', stage_name2, '" has insufficient helper staff (needs ≥2%)');
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
END IF;
END;
// 
DELIMITER ;
```

Τέλος, αν το security count είναι μικρότερο από το stage capacity * 0.05 (5%) πετάει error μαζί με το αντίστοιχο μήνυμα. Ομοίως και για το helper count, απλά με stage capacity * 0.02 (2%). Φροντίσαμε το μήνυμα να μας λέει και ποια σκηνή δεν έχει αρκετό προσωπικό για ευκολότερη διόρθωση

5. Φτιάξαμε μια procedure για να ελέγξει με τη μια τα capacities όλων των σκηνών, αντί να κάνουμε call check_stage_personel 30 φορές (μια για κάθε σκηνή) Στην ουσία καλεί το προηγούμενο procedure για κάθε σκηνή

```
DELIMITER //
CREATE PROCEDURE check_all_stages()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE sid INT;
    DECLARE cur CURSOR FOR SELECT stage_id FROM stage;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;

    stage_loop: LOOP
        FETCH cur INTO sid;
        IF done THEN
            LEAVE stage_loop;
        END IF;
        CALL check_stage_personel(sid);
    END LOOP;

    CLOSE cur;
END;
// 
DELIMITER ;
```

Συγκεκριμένα, με την done σταματάει η επανάληψη όταν ελέγξει όλα τα stages και με το sid κρατάει κάθε φορά το stage id.

Δημιουργούμε έναν cursor που διατρέχει όλα τα stage_id από τον πίνακα stage. Λειτουργεί σαν iterator. Επιτρέπει δηλαδή να επεξεργαστείς σειριακά κάθε γραμμή του stage. Όταν τελειώσουν τα δεδομένα του cursor, η μεταβλητή done γίνεται TRUE.

Ανοίγουμε τον cursor για χρήση και έπειτα εκτελούμε το loop. Κάθε φορά ο cursor φέρνει στο sid το επόμενο stage id. Αν το done είναι true τότε κάνουμε leave loop, καθώς είδαμε όλα τα stages. Άλλιως, καλεί την προηγούμενη procedure και ελέγχει την current σκηνή.

Τέλος κάνουμε close cur, απελευθερώνοντας την μνήμη.

6. Δημιουργούμε procedure για να καλύψουμε το total participation του festival με τα events. Δηλαδή, ότι ένα φεστιβάλ δεν μπορεί να μείνει χωρίς event (το αντιστροφό το καλύψαμε στα tables με το foreign key)

```

DELIMITER //
• CREATE PROCEDURE check_festival_total_participation()
BEGIN
    DECLARE missing_festival_id INT;
    DECLARE err_msg TEXT;

    SELECT f.festival_id INTO missing_festival_id
    FROM festival f
    WHERE NOT EXISTS (
        SELECT 1
        FROM event e
        WHERE e.festival_id = f.festival_id
    )
    ORDER BY f.festival_id
    LIMIT 1;

    IF missing_festival_id IS NOT NULL THEN
        SET err_msg = CONCAT('Festival ', missing_festival_id, ' has no events assigned.');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
    END IF;
END //
DELIMITER ;

```

Ορίζουμε μεταβλητές για να κρατήσουμε το id του φεστιβάλ χωρίς event και το πιθανό error message

Με το select παίρνει κάθε festival id από τον πίνακα festival.

Ψάχνει να βρει αν υπάρχει event με αυτό το festival id. Αν υπάρχει, η NOT EXISTS απορρίπτει αυτό το φεστιβάλ και δεν το βάζει στο missing_festival_id, άρα μένει null. Αν δεν υπάρχει event με αυτό το festival id, τότε η NOT EXISTS επιστρέφει στο missing_festival_id το id του current festival.

Κάνουμε limit 1, ώστε να πάρουμε το πρώτο φεστιβάλ χωρίς event.

Έτσι, αν το missing_festival_id δεν είναι NULL επιστρέφει σφάλμα με το αντίστοιχο μήνυμα, το οποίο μας λέει ποιο stage δεν έχει event, και άρα μπορούμε εύκολα να το διορθώσουμε.

7. Ομοίως ελέγχουμε το total participation του stage στο equipment, δηλαδή ότι δεν μπορεί να μείνει stage χωρίς equipment. Το αντίστροφο, δηλαδή equipment χωρίς stage κάναμε παραδοχή ότι επιτρέπεται.

```

DELIMITER //
CREATE PROCEDURE stage_total_participation_in_technical_equipment()
BEGIN
    DECLARE missing_stage_id INT;
    DECLARE err_msg TEXT;
    SELECT s.stage_id INTO missing_stage_id
    FROM stage s
    WHERE NOT EXISTS (
        SELECT 1
        FROM stage_equipment se
        WHERE se.stage_id = s.stage_id
    )
    ORDER BY stage_id
    LIMIT 1;

    IF missing_stage_id IS NOT NULL THEN
        SET err_msg = CONCAT('Stage ', missing_stage_id, ' has no Technical Equipment.');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT=err_msg;
    END IF;
END;
// 
DELIMITER ;

```

Πάλι, μας λέει ποιο stage δεν έχει εξοπλισμό, για εύκολη διόρθωση

8. Δημιουργούμε procedure για να ελέγχουμε ότι κάθε stage έχει τουλάχιστον ένα personnel από κάθε ρόλο. Θα μπορούσαμε να βάλουμε μόνο για το τεχνικό προσωπικό, αφού το προσωπικό ασφαλείας και το βοηθητικό προσωπικό καλύπτονται από το stage_capacity procedure, αλλά αποφασίσαμε να ελέγχουμε και για τους 3 το total participation.

```

DELIMITER //
CREATE PROCEDURE check_stage_role_coverage()
BEGIN
    DECLARE missing_stage_id INT DEFAULT NULL;
    DECLARE missing_role_id INT DEFAULT NULL;
    DECLARE role_name VARCHAR(50) DEFAULT NULL;
    DECLARE err_msg TEXT;

    -- Προσπάθεια να φορτώθει ένα μόνο κενό
    SELECT s.stage_id, r.role_id, r.role
    INTO missing_stage_id, missing_role_id, role_name
    FROM stage s
    CROSS JOIN role r
    WHERE NOT EXISTS (
        SELECT 1
        FROM stage_personel sp
        JOIN personnel p ON sp.personel_id = p.personel_id
        WHERE sp.stage_id = s.stage_id AND p.role_id = r.role_id
    )
    ORDER BY s.stage_id, r.role_id
    LIMIT 1;

    -- Εδώ γίνεται η απλή IF
    IF missing_stage_id IS NOT NULL THEN
        SET err_msg = CONCAT('Stage ', missing_stage_id, ' is missing role ', role_name, " (role_id='", missing_role_id, "')");
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
    END IF;
END;
// 
DELIMITER ;

```

Αρχικά, αποθηκεύουμε το stage_id της σκηνής που έχει πρόβλημα, το role_id που λείπει, το role_name και το τελικό μήνυμα σφάλματος

Με το select και το cross join συνδυάζουμε κάθε σκηνή με κάθε ρόλο. Δηλαδή, με 30 σκηνές και 3 ρόλους, φτιάχνονται 90 ζεύγη. Έπειτα, για κάθε τέτοιο ζεύγος ελέγχει αν υπάρχει κάποιος άνθρωπος από το personnel, με αυτό τον ρόλο, που δουλεύει στη σκηνή αυτή. Αν ναι, δεν επιστρέφει τίποτα και το missing_stage_id μένει null. Αν όχι, βάζει το id της σκηνής αυτής, το role id καθώς και το όνομα του ρόλου που λείπει στις μεταβλητές μας. Άρα, missing_stage_id θα είναι not null και θα πετάξει error, με το αντίστοιχο μήνυμα που θα αναφέρει τη σκηνή και τον ρόλο που λείπει.

9. Δημιουργούμε trigger που ελέγχει before insert σε ένα event, αν το event είναι εντός των ημερών του φεστιβάλ.

```

DELIMITER //
CREATE TRIGGER check_event_within_festival
BEFORE INSERT ON event
FOR EACH ROW
BEGIN
    DECLARE fest_start DATE;
    DECLARE fest_end DATE;
    DECLARE err_msg TEXT;

    -- Παίρνουμε τις ημερομηνίες του festival
    SELECT start, end
    INTO fest_start, fest_end
    FROM festival
    WHERE festival_id = NEW.festival_id;

    -- Έλεγχος: η ημερομηνία του event πρέπει να είναι εντός των ορίων
    IF NEW.event_date < fest_start OR NEW.event_date > fest_end THEN
        SET err_msg=CONCAT(
            'Event must be within festival period: ',
            DATE_FORMAT(fest_start, '%Y-%m-%d'),
            ' to ',
            DATE_FORMAT(fest_end, '%Y-%m-%d')
        );
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
    END IF;
END;
// 
DELIMITER ;

```

Ορίζουμε μεταβλητές για το πότε αρχίζει και τελειώνει το φεστιβάλ, καθώς και για το πιθανό μήνυμα σφάλματος.

Κάνουμε select το πότε αρχίζει και τελειώνει το συγκεκριμένο φεστιβάλ στο οποίο βρίσκεται το event από τον πίνακα φεστιβαλ.

Έπειτα, ελέγχουμε αν το event date είναι πριν αρχίζει το φεστιβάλ, ή αφότου τελειώσει. Αν έστω μια από τις δύο συνθήκες ισχύει, πετάει error, διευκρινίζοντας ποια ημερομηνία αρχίζει και τελειώνει το φεστιβαλ.

10. Το trigger check_ticket_limits ενεργοποιείται πριν από κάθε εισαγωγή (BEFORE INSERT) στον πίνακα ticket, και έχει σκοπό να διασφαλίσει πολλαπλούς περιορισμούς

Καταρχάς, απαγορεύεται κατά την εισαγωγή ενός νέου εισιτηρίου να οριστεί το πεδίο for_sale ως 1. Αυτό γίνεται μέσω ελέγχου με IF NEW.for_sale = 1, και σε περίπτωση παραβίασης εγείρεται σφάλμα με SIGNAL SQLSTATE. Ακολουθεί η δημιουργία του purchase_datetime, το οποίο προκύπτει από τη συνένωση της purchase_date και purchase_time χρησιμοποιώντας TIMESTAMP(). Στη συνέχεια, γίνεται έλεγχος για το αν η ώρα αγοράς βρίσκεται στο μέλλον σε σχέση με την τρέχουσα χρονική στιγμή του συστήματος (NOW()), και σε τέτοια περίπτωση απορρίπτεται η εισαγωγή. Αυτό διασφαλίζει ότι δεν θα υπάρχει εισιτήριο με μελλοντική ημερομηνία/ώρα αγοράς, κάτι που θα ήταν λογικά μη αποδεκτό.

```

DELIMITER //
CREATE TRIGGER check_ticket_limits
BEFORE INSERT ON ticket
FOR EACH ROW
BEGIN
    DECLARE vip_category_id TINYINT;
    DECLARE stage_capacity INT;
    DECLARE total_tickets INT;
    DECLARE vip_tickets INT;
    DECLARE event_end DATETIME;
    DECLARE purchase_datetime DATETIME;

    -- Απαγόρευση για for_sale ή activated κατά την εισαγωγή
    IF NEW.for_sale = 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Tickets cannot be marked as for sale during creation.';
    END IF;

    -- Συνένωση purchase_date και CURTIME() σε DATETIME
    SET purchase_datetime = TIMESTAMP(NEW.purchase_date, NEW.purchase_time);

    -- Έλεγχος: δεν επιτρέπεται η αγορά από το μέλλον
    IF purchase_datetime > NOW() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Purchase datetime cannot be in the future.';
    END IF;

```

To trigger συνεχίζει με έλεγχο χωρητικότητας της σκηνής. Γίνεται JOIN μεταξύ event και stage, μέσω του stage_id, και εξάγεται η χωρητικότητα της σκηνής (s.capacity) για το συγκεκριμένο event (NEW.event_id). Στη συνέχεια, μετρώνται όλα τα εισιτήρια που έχουν ήδη εκδοθεί για την παράσταση (event_id) μέσω COUNT(*). Αν ο συνολικός αριθμός των υπαρχόντων εισιτηρίων φτάσει ή ξεπεράσει τη χωρητικότητα της σκηνής, τότε η εισαγωγή απορρίπτεται με σχετικό μήνυμα ("Stage is full").

```

-- Πάρε τη χωρητικότητα της σκηνής για το event
SELECT s.capacity
INTO stage_capacity
FROM event e
JOIN stage s ON e.stage_id = s.stage_id
WHERE e.event_id = NEW.event_id;

-- Μέτρησε εισιτήρια που έχουν εκδοθεί ήδη
SELECT COUNT(*) INTO total_tickets
FROM ticket
WHERE event_id = NEW.event_id;

-- Έλεγχος 1: αν έχει γεμίσει η σκηνή
IF total_tickets >= stage_capacity THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Stage is full';
END IF;

```

Στη συνέχεια, γίνεται ειδικός έλεγχος για τα VIP εισιτήρια. Αναζητείται από τον πίνακα ticket_category το id_ticket_category που αντιστοιχεί στην κατηγορία VIP, και στη συνέχεια ελέγχεται αν το νέο εισιτήριο είναι αυτής της κατηγορίας. Αν είναι, τότε μετρώνται τα ήδη εκδοθέντα VIP εισιτήρια για το ίδιο event, και συγκρίνονται με το 10% της χωρητικότητας της σκηνής. Αν το όριο έχει ξεπεραστεί, η εισαγωγή απορρίπτεται με μήνυμα "The maximum number of VIP tickets has been reached for this event."

Τέλος, γίνεται έλεγχος αν η αγορά του εισιτηρίου επιχειρείται μετά το τέλος της παράστασης. Υπολογίζεται το event_end χρονικό σημείο συνδυάζοντας την event_date και την end_time του συγκεκριμένου event. Αν η purchase_datetime είναι ίδια ή μεταγενέστερη του event_end, τότε η αγορά απορρίπτεται.

```

-- Έλεγχος 2: VIP περιορισμός (μέγιστο 10% της χωρητικότητας)
SELECT id_ticket_category INTO vip_category_id
FROM ticket_category
WHERE category_name = 'VIP';

IF NEW.id_ticket_category = vip_category_id THEN
    SELECT COUNT(*) INTO vip_tickets
    FROM ticket
    WHERE event_id = NEW.event_id
        AND id_ticket_category = vip_category_id;

    IF vip_tickets >= stage_capacity / 10 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The maximum number of VIP tickets has been reached for this event.';
    END IF;
END IF;

-- Έλεγχος 3: Μην επιτρέπεις εισιτήριο για event που έχει τελειώσει
SELECT TIMESTAMP(e.event_date, e.end_time)
INTO event_end
FROM event e
WHERE e.event_id = NEW.event_id;

IF purchase_datetime >= event_end THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot issue ticket for an event that has already ended.';
END IF;

END;
DELETER //
```

11. Το trigger ticket_update_restrictions ενεργοποιείται πριν από κάθε ενημέρωση (BEFORE UPDATE) στον πίνακα ticket, με σκοπό να επιβάλει περιορισμούς που αφορούν την κατάσταση ενεργοποίησης (activated) και τη μεταπώληση (for_sale) ενός εισιτηρίου, καθώς και να διαχειριστεί την εισαγωγή του στη σχετική ουρά μεταπώλησης (resell_queue), όταν πληρούνται οι προϋποθέσεις.

Αρχικά, ελέγχεται αν γίνεται προσπάθεια αλλαγής της τιμής του πεδίου activated από 1 σε 0, δηλαδή αν επιχειρείται να απενεργοποιηθεί ένα εισιτήριο που έχει ήδη ενεργοποιηθεί. Αυτό δεν επιτρέπεται, καθώς η ενεργοποίηση αντιστοιχεί σε πραγματική είσοδο του επισκέπτη στο φεστιβάλ και τυχόν απενεργοποίηση εκ των υστέρων θα μπορούσε να οδηγήσει σε καταχρηστικές ενέργειες, όπως η πολλαπλή χρήση του ίδιου εισιτηρίου. Ο έλεγχος υλοποιείται με τη συνθήκη IF OLD.activated = 1 AND NEW.activated = 0, και σε περίπτωση που ισχύει, ο trigger αποτρέπει την ενημέρωση μέσω SIGNAL SQLSTATE, εγείροντας σφάλμα.

Στη συνέχεια, ελέγχεται η συνθήκη κατά την οποία είτε το παλιό είτε το νέο activated είναι 1 και ταυτόχρονα γίνεται προσπάθεια να τεθεί for_sale = 1. Αυτό αντιστοιχεί στην απαγορευμένη κατάσταση όπου ένας επισκέπτης προσπαθεί να πουλήσει ένα ήδη χρησιμοποιημένο εισιτήριο, κάτι που προφανώς δεν επιτρέπεται.

Επιπλέον, γίνεται αντίστροφος έλεγχος: αν είτε το παλιό είτε το νέο for_sale είναι 1, και ταυτόχρονα επιχειρείται η ενεργοποίηση του εισιτηρίου (activated = 1), τότε η λειτουργία απαγορεύεται. Αυτό εξασφαλίζει ότι ένα εισιτήριο που βρίσκεται σε κατάσταση προς πώληση δεν μπορεί να ενεργοποιηθεί για είσοδο, έως ότου αποσυρθεί από την ουρά μεταπώλησης. Είναι σημαντικό να τονίσουμε, εδώ, πως κάποιος μπορεί να θέσει το εισιτήριο προς πώληση και να το μετανιώσει και να θέλει να το χρησιμοποιήσει, υπό την προϋπόθεση ότι το έχει βγάλει από την ουρά μεταπώλησης

```

CREATE TRIGGER ticket_update_restrictions
BEFORE UPDATE ON ticket
FOR EACH ROW
BEGIN
    DECLARE stage_capacity INT;
    DECLARE total_tickets INT;
    DECLARE event_end DATETIME;

    -- Απαγόρευση αλλαγής activated από 1 σε 0
    IF OLD.activated = 1 AND NEW.activated = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Activated tickets cannot be deactivated.';
    END IF;

    -- Απαγόρευση να τεθεί for_sale = 1 σε activated εισιτήριο
    IF (OLD.activated = 1 OR NEW.activated = 1) AND NEW.for_sale = 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Activated tickets cannot be put for sale.';
    END IF;

    -- Απαγόρευση να τεθεί activated = 1 σε for_sale εισιτήριο
    IF (OLD.for_sale = 1 OR NEW.for_sale = 1) AND NEW.activated = 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'For sale tickets cannot be activated.';
    END IF;

```

Το πιο σημαντικό κομμάτι του trigger αφορά τη διαχείριση της μεταπώλησης εισιτηρίων. Όταν γίνεται προσπάθεια να τεθεί for_sale = 1, τότε ο trigger ελέγχει αν η σκηνή στην οποία ανήκει το event είναι ήδη γεμάτη. Για να γίνει αυτό, ο trigger βρίσκει τη χωρητικότητα της σκηνής (s.capacity) κάνοντας JOIN του πίνακα event με τον πίνακα stage μέσω stage_id, και στη συνέχεια μετρά πόσα εισιτήρια έχουν εκδοθεί για το συγκεκριμένο event. Αν το πλήθος των εκδοθέντων εισιτηρίων είναι μικρότερο από τη χωρητικότητα της σκηνής, τότε δεν επιτρέπεται η μεταπώληση.

Τέλος, όταν πληρούνται οι παραπάνω προϋποθέσεις και η σκηνή είναι γεμάτη, ο trigger εισάγει αυτόματα το εισιτήριο στην ουρά μεταπώλησης, μέσω INSERT INTO resell_queue. Η ενέργεια αυτή συνδέει λειτουργικά τον πίνακα ticket με τον πίνακα resell_queue, δημιουργώντας μια έμμεση σχέση μεταξύ τους, όπου το code_ean13 λειτουργεί ως συνδετικός κρίκος.

```
-- Αν προσπαθεί να βάλει for_sale = 1 → επιτρέπεται ΜΟΝΟ αν η σκηνή είναι γεμάτη
IF NEW.for_sale = 1 AND OLD.for_sale = 0 THEN
    -- Πάρε χωρητικότητα σκηνής
    SELECT s.capacity INTO stage_capacity
    FROM event e
    JOIN stage s ON e.stage_id = s.stage_id
    WHERE e.event_id = NEW.event_id;

    -- Μέτρησε πόσα εισιτήρια έχουν εκδοθεί για το event
    SELECT COUNT(*) INTO total_tickets
    FROM ticket
    WHERE event_id = NEW.event_id;

    -- Αν η σκηνή δεν είναι γεμάτη → απαγόρευση
    IF total_tickets < stage_capacity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Tickets can only be marked as for sale after the stage is full.';
    END IF;

    -- Εισαγωγή στην ουρά μεταπώλησης
    INSERT INTO resell_queue (code_ean13, timestamp)
    VALUES (NEW.code_ean13, NOW());
END IF;
END;
//
```

12. Το trigger prevent_delete_activated_ticket ενεργοποιείται πριν από κάθε διαγραφή (BEFORE DELETE) εγγραφής από τον πίνακα ticket, και έχει στόχο να αποτρέψει τη διαγραφή εισιτηρίων που έχουν ήδη ενεργοποιηθεί. Πιο συγκεκριμένα, ελέγχεται αν το πεδίο activated της εγγραφής που πρόκειται να διαγραφεί είναι ίσο με 1. Αν ισχύει αυτό, τότε εγείρεται σφάλμα μέσω SIGNAL SQLSTATE και η διαγραφή ακυρώνεται. Η ενεργοποίηση ενός εισιτηρίου σηματοδοτεί ότι έχει ήδη χρησιμοποιηθεί για είσοδο στο φεστιβάλ και η διαγραφή του θα αλλοίωνε το ιστορικό χρήσης και την ακρίβεια

των δεδομένων. Με τον έλεγχο αυτό, διασφαλίζουμε την ακεραιότητα των εγγραφών.

```

CREATE TRIGGER prevent_delete_activated_ticket
BEFORE DELETE ON ticket
FOR EACH ROW
    BEGIN
        IF OLD.activated = 1 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Activated tickets cannot be deleted.';
        END IF;
    END;
    //

```

13. Το procedure `match_resale()` υλοποιεί τον πλήρη μηχανισμό μεταπώλησης εισιτηρίων, με βάση τις ουρές `resell_queue` και `buyer_queue`. Ο στόχος της είναι να ταιριάξει αγοραστές με διαθέσιμα προς μεταπώληση εισιτήρια, με σειρά προτεραιότητας (FIFO), και να διαχειριστεί τη μεταβίβαση του εισιτηρίου σε νέο επισκέπτη.

Αρχικά, δηλώνονται μεταβλητές ελέγχου για την ανάγνωση του cursor (`done`) και για την αποθήκευση των ενδιάμεσων τιμών εισιτηρίου, αγοραστή και προηγούμενου κατόχου (`matched_ticket`, `matched_buyer_id`, `previous_visitor_id` κ.λπ.). Στη συνέχεια, ορίζεται cursor που ανακτά τα υποψήφια `matches` μεταξύ `resell_queue` και `buyer_queue`. Το ταίριασμα γίνεται είτε με βάση τον ακριβή `code_ean13` (συγκεκριμένο εισιτήριο) είτε με βάση τον συνδυασμό `event_id` και `ticket_category` (γενική προτίμηση). Η επιλογή ταξινομείται χρονικά με βάση τη στιγμή που το εισιτήριο μπήκε στη `resell_queue` και τη στιγμή που ο αγοραστής μπήκε στην `buyer_queue`, ώστε να τηρείται αυστηρά σειρά FIFO.

```

    BEGIN
        DECLARE done INT DEFAULT 0;
        DECLARE matched_ticket CHAR(13);
        DECLARE matched_buyer_id INT;
        DECLARE matched_buyer_queue_id INT;
        DECLARE previous_visitor_id INT;
        DECLARE existing_visitor_id INT;
        DECLARE buyer_requests_remaining INT;
        DECLARE tickets_remaining INT;

        -- Cursor: match κατά FIFO, με έλεγχο στις ανά code_ean13 στις ανά event & category
        DECLARE cur CURSOR FOR
            SELECT rq.code_ean13, b.buyer_id, bq.buyer_queue_id
            FROM resell_queue rq
            JOIN ticket t ON rq.code_ean13 = t.code_ean13
            JOIN buyer_queue bq
            ON (
                (bq.code_ean13 IS NOT NULL AND bq.code_ean13 = rq.code_ean13)
                OR
                (bq.code_ean13 IS NULL AND bq.event_id = t.event_id AND bq.id_ticket_category = t.id_ticket_category)
            )
            JOIN buyer b ON b.buyer_id = bq.buyer_id
            ORDER BY rq.timestamp, bq.timestamp;

        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    
```

Η διαδικασία διατρέχει για το πρώτο ζεύγος που βρίσκει εκτελεί τα εξής βήματα:

Αρχικά, αναζητείται ο αρχικός επισκέπτης που κατείχε το εισιτήριο (previous_visitor_id) από τον πίνακα ticket. Έπειτα, ελέγχεται αν τα στοιχεία του αγοραστή (name, last_name, email) αντιστοιχούν ήδη σε υπάρχοντα επισκέπτη (visitor). Αν υπάρχει τέτοιος επισκέπτης (existing_visitor_id), τότε το εισιτήριο μεταβιβάζεται σε αυτόν με ενημέρωση του id_visitor στον πίνακα ticket, ενώ τα πεδία for_sale και activated μηδενίζονται για να δηλώσουν ότι το εισιτήριο δεν είναι πλέον προς πώληση και δεν έχει χρησιμοποιηθεί.

```

OPEN cur;

FETCH cur INTO matched_ticket, matched_buyer_id, matched_buyer_queue_id;

-- Πάρε τον προηγούμενο visitor που κατείχε το εισιτήριο
SELECT id_visitor INTO previous_visitor_id
FROM ticket
WHERE code_ean13 = matched_ticket;

-- Δες αν υπάρχει ήδη visitor με τα στοιχεία του buyer
SELECT v.id_visitor INTO existing_visitor_id
FROM visitor v
JOIN buyer b ON b.buyer_id = matched_buyer_id
WHERE v.first_name = b.name AND v.last_name = b.last_name AND v.email = b.email
LIMIT 1;

IF existing_visitor_id IS NOT NULL THEN
    -- Αν υπάρχει, το εισιτήριο περνάει σε αυτόν
    UPDATE ticket
    SET id_visitor = existing_visitor_id,
        for_sale = 0,
        activated = 0
    WHERE code_ean13 = matched_ticket;
ELSE

```

Αν δεν υπάρχει αντιστοιχία με υπάρχοντα επισκέπτη, τότε δημιουργείται νέα εγγραφή στον πίνακα visitor με τα στοιχεία του αγοραστή. Η εισαγωγή γίνεται με INSERT INTO visitor ... SELECT FROM buyer, ώστε να αντιγραφούν τα δεδομένα. Στη συνέχεια, με χρήση του LAST_INSERT_ID(), το εισιτήριο μεταφέρεται στον νέο επισκέπτη και δηλώνεται ανενεργό και μη προς πώληση, όπως και στην προηγούμενη περίπτωση.

```

-- Αν όχι, δημιουργούμε νέο visitor με τα στοιχεία του buyer
INSERT INTO visitor (first_name, last_name, phone, email, age)
SELECT name, last_name, phone, email, age
FROM buyer
WHERE buyer_id = matched_buyer_id;

SET existing_visitor_id = LAST_INSERT_ID();

-- Εκχωρούμε το εισιτήριο στον νέο visitor
UPDATE ticket
SET id_visitor = existing_visitor_id,
for_sale = 0,
activated = 0
WHERE code_ean13 = matched_ticket;
END IF;

-- Αφαιρεση από τις ουρές
DELETE FROM buyer_queue WHERE buyer_queue_id = matched_buyer_queue_id;
DELETE FROM resell_queue WHERE code_ean13 = matched_ticket;

```

Μετά την επιτυχή μεταφορά του εισιτηρίου, το αίτημα αγοράς του αγοραστή αφαιρείται από την buyer_queue και το αντίστοιχο εισιτήριο αφαιρείται από τη resell_queue. Ακολουθεί έλεγχος για να διαπιστωθεί αν ο αγοραστής έχει άλλες ενεργές αιτήσεις. Αν όχι, τότε διαγράφεται και από τον πίνακα buyer, καθαρίζοντας πλήρως το σύστημα από ανενεργούς αγοραστές.

Τέλος, ελέγχεται αν ο αρχικός κάτοχος (visitor) έχει πλέον άλλα εισιτήρια. Αν δεν έχει, τότε διαγράφεται και από τον πίνακα visitor, ώστε να διατηρείται το σύστημα καθαρό από ανενεργές εγγραφές.

```

-- Αν ο buyer δεν έχει άλλα αιτήματα, διαγράφεται
SELECT COUNT(*) INTO buyer_requests_remaining
FROM buyer_queue
WHERE buyer_id = matched_buyer_id;

IF buyer_requests_remaining = 0 THEN
    DELETE FROM buyer WHERE buyer_id = matched_buyer_id;
END IF;

-- Αν ο προηγούμενος visitor δεν έχει πλέον εισιτήρια, διαγράφεται
IF previous_visitor_id IS NOT NULL AND previous_visitor_id != existing_visitor_id THEN
    SELECT COUNT(*) INTO tickets_remaining
    FROM ticket
    WHERE id_visitor = previous_visitor_id;

    IF tickets_remaining = 0 THEN
        DELETE FROM visitor WHERE id_visitor = previous_visitor_id;
    END IF;
END IF;

CLOSE cur;
END;
//
```

14. Με σκοπό την αυτόματη εκτέλεση της `match_resale()`, έχουμε φτιάξει τα 2 triggers και ένα event.

Πιο συγκεκριμένα, το trigger `trigger_resell_queue_flag` ενεργοποιείται μετά την εισαγωγή εγγραφής στον πίνακα `resell_queue` και αυξάνει το `is_pending` κατά 1. Το ίδιο ακριβώς συμβαίνει και στο trigger `trigger_buyer_queue_flag`, που παρακολουθεί τον πίνακα `buyer_queue`. Κάθε νέα εισαγωγή συνεπάγεται και μία πιθανή ανάγκη για εκτέλεση της διαδικασίας `match_resale()`. Η χρήση αυτής της προσέγγισης εξασφαλίζει ότι καλείται η `match_resale()` μόνο όταν πραγματικά υπάρχουν νέα δεδομένα στις ουρές, και όχι συνεχώς χωρίς λόγο, βελτιώνοντας έτσι σημαντικά την αποδοτικότητα της βάσης μας.

```

CREATE TRIGGER trigger_resell_queue_flag
AFTER INSERT ON resell_queue
FOR EACH ROW
    BEGIN
        UPDATE match_pending
        SET is_pending = is_pending + 1
        WHERE id = 1;
    END;
    //

CREATE TRIGGER trigger_buyer_queue_flag
AFTER INSERT ON buyer_queue
FOR EACH ROW
    BEGIN
        UPDATE match_pending
        SET is_pending = is_pending + 1
        WHERE id = 1;
    END;
CREATE EVENT event_call_match_resale
ON SCHEDULE EVERY 1 MINUTE
DO
    BEGIN
        DECLARE i INT DEFAULT 0;
        DECLARE pending_count INT;

        SELECT is_pending INTO pending_count FROM match_pending WHERE id = 1;

        WHILE i < pending_count DO
            CALL match_resale();
            SET i = i + 1;
        END WHILE;

        UPDATE match_pending
        SET is_pending = 0, last_checked = NOW()
        WHERE id = 1;
    END;
    //

```

Ακολούθως, έχουμε δημιουργήσει ένα event, το event_call_match_resale, το οποίο έχει οριστεί να εκτελείται ανά ένα λεπτό. Το event διαβάζει την τιμή του is_pending από τον πίνακα match_pending, η οποία δείχνει πόσες ενέργειες εισαγωγής έγιναν συνολικά στις ουρές από το προηγούμενο λεπτό. Στη συνέχεια, καλεί τη διαδικασία match_resale() ακριβώς τόσες φορές. Αυτό είναι απαραίτητο επειδή, όπως έχει υλοποιηθεί, η match_resale() διαχειρίζεται μόνο μία αντιστοίχιση κάθε φορά, επομένως χρειάζεται να εκτελεστεί επαναληπτικά για να επεξεργαστεί πολλαπλά αιτήματα. Τέλος, το event μηδενίζει το is_pending και ενημερώνει το last_checked ώστε να καταγράφεται πότε έγινε ο τελευταίος κύκλος ελέγχου.

Αξίζει να σημειωθεί ότι η αρχική μας ιδέα ήταν να καλέσουμε απευθείας τη match_resale() μέσα από trigger, κατά την εισαγωγή σε resell_queue ή buyer_queue. Ωστόσο, αυτό δεν επιτρέπεται από τη MySQL, και συγκεκριμένα προκαλεί το σφάλμα ERROR 1442 (HY000): Can't update table 'ticket' in stored function/trigger because it is already used by the statement which invoked this stored function/trigger. Το σφάλμα αυτό συμβαίνει επειδή η MySQL δεν επιτρέπει η ίδια η ενέργεια ενός trigger να μεταβάλλει πίνακα που επηρέασε την κλήση του, για λόγους αποφυγής κυκλικών εξαρτήσεων και αβεβαιότητας στην εκτέλεση. Γι' αυτόν τον λόγο επιλέξαμε τη λύση με χρήση event και εξωτερικής διαδικασίας, η οποία διασφαλίζει την εκτελεστικότητα και τη σταθερότητα του συστήματος.

15. Δημιουργούμε με το trigger check_buyer_queue_constraints το οποίο ενεργοποιείται πριν από κάθε εισαγωγή (BEFORE INSERT) στον πίνακα buyer_queue.

Ο πρώτος έλεγχος αφορά τη δομή της αίτησης. Ελέγχεται αν ο αγοραστής έχει δηλώσει είτε συγκεκριμένο code_ean13 (δηλαδή συγκεκριμένο εισιτήριο που επιθυμεί να αγοράσει), είτε συνδυαστικά τα πεδία event_id και id_ticket_category, που δηλώνουν γενικό ενδιαφέρον για τύπο εισιτηρίου σε συγκεκριμένο event. Αν δεν έχει συμπληρωθεί καμία από τις δύο περιπτώσεις, το trigger αποτρέπει την εισαγωγή με σχετικό μήνυμα σφάλματος.

```

CREATE TRIGGER check_buyer_queue_constraints
BEFORE INSERT ON buyer_queue
FOR EACH ROW
BEGIN
    DECLARE existing_ticket_count INT;
    DECLARE buyer_visitor_id INT;
    DECLARE target_event_id INT;
    DECLARE resale_exists INT;

    -- 1. Ο buyer πρέπει να έχει δηλώσει είτε code_ean13 είτε (event_id και id_ticket_category)
    IF NEW.code_ean13 IS NULL AND (NEW.event_id IS NULL OR NEW.id_ticket_category IS NULL) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You must either specify a specific ticket code or both event and ticket category.';
    END IF;

    -- 2. Αν έχει δηλώσει code_ean13, τότε αυτό πρέπει να υπάρχει ήδη στο resell_queue
    IF NEW.code_ean13 IS NOT NULL THEN
        IF NOT EXISTS (
            SELECT 1
            FROM resell_queue
            WHERE code_ean13 = NEW.code_ean13
        ) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'The selected ticket code is not currently available for resale.';
        END IF;
    END IF;
END IF;

```

Στη συνέχεια, εφόσον έχει δηλωθεί code_ean13, o trigger ελέγχει αν το εισιτήριο αυτό υπάρχει ήδη στον πίνακα resell_queue. Αυτό διασφαλίζει ότι ο αγοραστής δεν μπορεί να αιτηθεί εισιτήριο το οποίο δεν έχει ακόμη τεθεί προς μεταπώληση.

```
-- 4. Αν υπάρχει αντίστοιχος visitor, έλεγξε αν έχει ήδη εισιτήριο για το ζητούμενο event
IF buyer_visitor_id IS NOT NULL THEN
    IF NEW.code_ean13 IS NOT NULL THEN
        SELECT event_id INTO target_event_id
        FROM ticket
        WHERE code_ean13 = NEW.code_ean13
        LIMIT 1;
    ELSE
        SET target_event_id = NEW.event_id;
    END IF;

    IF EXISTS (
        SELECT 1
        FROM ticket
        WHERE id_visitor = buyer_visitor_id AND event_id = target_event_id
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You already have a ticket for the selected event.';
    END IF;

END IF;
END;
//
```

Ο τρίτος έλεγχος αφορά το ενδεχόμενο ένας visitor να ζητήσει εισιτήριο για παράσταση, για την οποία ήδη έχει εισιτήριο, προσπαθώντας να παρακάμψει μία από τις βασικές απαιτήσεις της άσκησης που δηλώνει πως ένας visitor έχει το πολύ ένα εισιτήριο για μία παράσταση. Το trigger ελέγχει αν υπάρχει εγγραφή στον πίνακα visitor με ίδιο first_name, last_name και email με τον αγοραστή (buyer). Αν βρεθεί τέτοιος επισκέπτης (buyer_visitor_id), τότε εξετάζεται αν ήδη διαθέτει εισιτήριο για το ζητούμενο event. Για τον σκοπό αυτό, αν έχει δηλωθεί code_ean13, o trigger αναζητά το αντίστοιχο event_id από τον πίνακα ticket. Σε διαφορετική περίπτωση, χρησιμοποιεί απευθείας το NEW.event_id που δηλώθηκε στην αίτηση. Αν δει πως έχει ήδη εισιτήριο για αυτό το event, τότε αποτρέπει την εγγραφή.

16. To trigger check_evaluation_validity ενεργοποιείται πριν από κάθε εισαγωγή (BEFORE INSERT) στον πίνακα evaluation και έχει σκοπό να διασφαλίσει ότι η υποβολή μιας αξιολόγησης είναι έγκυρη και επιτρεπτή χρονικά.

Αρχικά, ανακτάται το event_id της εμφάνισης (performance) που αφορά η αξιολόγηση, μέσω αναζήτησης στον πίνακα performance με βάση το performance_id. Η τιμή αποθηκεύεται στη μεταβλητή event_id_of_perf.

Στη συνέχεια, ελέγχουμε αν ο επισκέπτης (visitor) έχει έγκυρο εισιτήριο για το event στο οποίο ανήκει η εμφάνιση. Αυτό επιτυγχάνεται μέσω EXISTS, αναζητώντας αν υπάρχει εγγραφή στον πίνακα ticket που να αντιστοιχεί στο συγκεκριμένο id_visitor και event_id. Αν δεν υπάρχει, o trigger απορρίπτει την εισαγωγή.

```

CREATE TRIGGER check_evaluation_validity
BEFORE INSERT ON evaluation
FOR EACH ROW
BEGIN
    DECLARE event_id_of_perf INT;
    DECLARE ticket_activated TINYINT;
    DECLARE performance_end DATETIME;

    -- 1. Πάρε το event_id της performance
    SELECT event_id INTO event_id_of_perf
    FROM performance
    WHERE performance_id = NEW.performance_id;

    -- 2. Έλεγχε αν ο visitor έχει εισιτήριο για το συγκεκριμένο event
    IF NOT EXISTS (
        SELECT 1 FROM ticket
        WHERE id_visitor = NEW.id_visitor AND event_id = event_id_of_perf
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Visitor must have a ticket for the event of the performance.';
    END IF;

```

Ακολουθεί ο δεύτερος έλεγχος, όπου εξετάζεται αν το εισιτήριο του επισκέπτη είναι ενεργοποιημένο (activated = 1). Αυτό σημαίνει ότι ο επισκέπτης πραγματικά παρευρέθηκε στο φεστιβάλ και εισήλθε στη σκηνή. Η πληροφορία αντλείται από το ίδιο εισιτήριο που ταιριάζει στον πίνακα ticket, και αν δεν είναι ενεργοποιημένο, η αξιολόγηση απορρίπτεται.

```

    -- 3. Έλεγχε αν το εισιτήριο είναι activated
    SELECT activated INTO ticket_activated
    FROM ticket
    WHERE id_visitor = NEW.id_visitor AND event_id = event_id_of_perf;

    IF ticket_activated != 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Ticket must be activated to submit an evaluation.';
    END IF;

    -- 4. Υπολόγισε την ώρα λήξης της performance
    SELECT TIMESTAMP(performance_date, performance_end_time)
    INTO performance_end
    FROM performance
    WHERE performance_id = NEW.performance_id;

    -- 5. Έλεγχε αν η ώρα τώρα είναι μετά την ώρα λήξης
    IF NOW() < performance_end THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Evaluation can only be submitted after the performance has ended.';
    END IF;
END;
//
```

Ο τρίτος έλεγχος αφορά το πότε επιτρέπεται να υποβληθεί η αξιολόγηση. Υπολογίζεται η ακριβής ώρα λήξης της εμφάνισης (performance_end) μέσω συνένωσης της performance_date και της performance_end_time από τον πίνακα performance. Έπειτα,

συγκρίνεται με την τρέχουσα ώρα του συστήματος (NOW()). Αν η υποβολή γίνεται πριν τη λήξη της εμφάνισης, η εισαγωγή απορρίπτεται.

17. Δημιουργούμε το view evaluation_likert_view που έχει ως σκοπό να μεταφράσει τις αριθμητικές τιμές αξιολόγησης που υπάρχουν στον πίνακα evaluation σε λεκτικές κατηγορίες, όπως αυτές ορίζονται στον πίνακα Likert. Με τον τρόπο αυτό, κάθε εγγραφή αξιολόγησης παρουσιάζεται σε πιο κατανοητή και αναγνώσιμη μορφή, π.χ. «Very Dissatisfied», «Disatisfied», «Neutral», «Satisfied» και «Very Satisfied»

Για την υλοποίηση της προβολής, χρησιμοποιούνται πέντε JOIN εντολές, μία για κάθε πεδίο αξιολόγησης: artist_performance_rating, sound_and_light_rating, stage_presentation_rating, organization_rating και overall_impression_rating. Κάθε πεδίο αριθμητικής κλίμακας (TINYINT) αντιστοιχίζεται με το αντίστοιχο id_likert στον πίνακα Likert, ώστε να επιστραφεί η περιγραφή likert_category.

```
CREATE VIEW evaluation_likert_view AS
SELECT
    e.id_visitor,
    e.performance_id,
    11.likert_category AS artist_performance,
    12.likert_category AS sound_and_light,
    13.likert_category AS stage_presentation,
    14.likert_category AS organization,
    15.likert_category AS overall_impression
FROM evaluation e
JOIN Likert 11 ON e.artist_performance_rating = 11.id_likert
JOIN Likert 12 ON e.sound_and_light_rating = 12.id_likert
JOIN Likert 13 ON e.stage_presentation_rating = 13.id_likert
JOIN Likert 14 ON e.organization_rating = 14.id_likert
JOIN Likert 15 ON e.overall_impression_rating = 15.id_likert;
```

Η προβολή επιστρέφει το id_visitor, το performance_id και πέντε στήλες με λεκτικές περιγραφές αξιολόγησης, τις οποίες μετονομάζουμε σε artist_performance, sound_and_light, stage_presentation, organization, και overall_impression αντίστοιχα.

18. Ελέγχουμε αν υπάρχει performance, το οποίο πραγματοποιείται την ίδια ώρα να μην πραγματοποιείται στο ίδιο stage.

```

DELIMITER //
-- αυτό σίναι ώστε κάθε performance που γίνεται την ίδια ώρα να μην πραγματοποιείται στην ίδια σκηνή
CREATE TRIGGER check_performance_stage_overlap
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1 FROM performance
        WHERE stage_id = NEW.stage_id
        AND performance_date = NEW.performance_date
        AND (
            (NEW.performance_start_time BETWEEN performance_start_time AND performance_end_time)
            OR
            (NEW.performance_end_time BETWEEN performance_start_time AND performance_end_time)
            OR
            (performance_start_time BETWEEN NEW.performance_start_time AND NEW.performance_end_time)
            OR
            (performance_end_time BETWEEN NEW.performance_start_time AND NEW.performance_end_time)
        )
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Time slot overlaps with existing performance on this stage.' ;
    END IF;
END;
//
```

Δημιουργούμε ένα trigger στο οποίο κάθε φορά πριν πραγματοποιήσουμε insert στο performance, ελέγχουμε αν το stage του performance που πάμε να καταχωρήσουμε είναι ήδη καταχωριμένο σε άλλο performance, ώρα η οποία συμπίπτει με οποιοδήποτε τρόπο με την ώρα του performance που πάμε να καταχωρήσουμε (4 περιπτώσεις). Αν πάμε να καταχωρήσουμε stage, το οποίο την ίδια ώρα έχει ήδη καταχωρηθεί πετάμε το μήνυμα error. Έτσι, διακόπτεται η insert και το νέο performance δε θα αποθηκευτεί στη βάση.

19. Ελέγχουμε αν ανάμεσα σε διαφορετικά performances που συμβαίνουν στην ίδια σκηνή υπάρχει η σωστή πρόβλεψη για διάλειμμα.

```

DELIMITER //
CREATE TRIGGER check_performance_break
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
    DECLARE end_time_of_previous TIME;
    DECLARE break_duration INT;
    DECLARE err_msg TEXT;

    SELECT performance_end_time INTO end_time_of_previous FROM performance
    WHERE stage_id = NEW.stage_id
        AND performance_date = NEW.performance_date
        AND performance_end_time <= NEW.performance_start_time
    ORDER BY performance_end_time DESC
    LIMIT 1;

    IF end_time_of_previous IS NOT NULL THEN
        SET break_duration = (NEW.performance_start_time - end_time_of_previous)/100;

        IF break_duration < 5 OR break_duration > 30 THEN
            SET err_msg = CONCAT('break between performances is', break_duration, ' so it is not allowed');
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
        END IF;
    END IF;
END;
//
```

Δημιουργούμε ένα trigger στο οποίο κάθε φορά πριν πραγματοποιήσουμε insert στο performance, ελέγχουμε αν η ακριβώς προηγούμενη εμφάνιση επιτρέπει χρονικά ένα διάλειμμα. Αρχικά, με ένα select βρίσκουμε την ακριβώς προηγούμενη εμφάνιση που συμβαίνει στο ίδιο stage και την ίδια μέρα πριν το performance που θέλουμε να κάνουμε insert. Αυτό το κάνουμε επιλέγοντας όλα τα performances (τα οποία έχουν ήδη γίνει inserted), τα οποία τελειώνουν πριν αρχίσει το νέο performance (end_time<=NEW.start_time) και ταξινομώντας τα από το performance με το μεγαλύτερο end_time στο μικρότερο κρατώντας τελικά 1. Άρα κρατάμε το performance που τελειώνει πριν αρχίσει το νέο performance που θέλουμε να κάνουμε insert. Έπειτα υπολογίζουμε τη χρονική απόσταση από το να τελειώσει το προηγούμενο performance μέχρι να αρχίσει το νέο και αν δεν είναι ανάμεσα σε 5-30 λεπτά πετάμε error και δεν γίνεται το insert.

20. Ελέγχουμε ότι κάθε performance δεν διαρκεί πάνω από 3 ώρες (μέγιστη διάρκεια εμφάνισης).

```

DELIMITER //
CREATE TRIGGER check_max_duration_performance
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
    IF NEW.performance_duration > 180 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Performance duration cannot exceed 3 hours';
    END IF;
END//
```

Για να το ελέγξουμε αυτό, δημιουργούμε ένα trigger το οποίο πριν από κάθε insert μιας παράστασης ελέγχει αν το duration είναι πάνω από 3 ώρες (μετράμε σε λεπτά) και αν είναι πετάει error και δεν γίνεται το insert.

21. Θέλουμε να φτιάξουμε μια λίστα με τα συγκροτήματα και τα μέλη του κάθε συγκροτήματος. Για να το πετύχουμε αυτό φτιάχνουμε ένα view, το οποίο είναι ένας εικονικός πίνακας (δεν αποθηκεύει δηλαδή δεδομένα από μόνο του).

```
CREATE OR REPLACE VIEW list_band_members AS
SELECT
    b.band_name,
    GROUP_CONCAT(CONCAT(a.artist_name, ' ', a.artist_surname) SEPARATOR ', ')
FROM
    band b
JOIN artist_band ab ON b.band_id = ab.band_id
JOIN artist a ON a.artist_id = ab.artist_id
GROUP BY b.band_id, b.band_name;
```

Για κάθε συγκρότημα του πίνακα band βρίσκει κάθε artist που ανήκει σε αυτό και κάνει concat το όνομα και το επίθετο του. Ο τρόπος που βρίσκει τους artists είναι μέσω του πίνακα artist_band, αφού συγκρίνει ποιοι καλλιτέχνες είναι συνδεδεμένοι με το αντίστοιχο κάθε φορά band_id. Τα ονοματεπώνυμα των καλλιτεχνών τα βρίσκει μέσω του πίνακα artist, ταυτίζοντας τα artist_id του πίνακα αυτού με του artist_band.

Το αποτέλεσμα που παίρνουμε είναι της μορφής:

	band_name	Name_exp_2
▶	The Rockers	Zoe Stathopoulou, Panagiotis Marinou, Nikos Karamitsos
	Jazz Ensemble	Stavros Kosta, Alex Lazarou, Petros Alexiou, Zoe Andreou, Eleni Andreou, Maria Dimitriou, Kostas Papadopoulos
	Electro Kings	Christos Kosta, Giorgos Nikolaidis, Anna Marinou, Nikos Lazarou, Stavros Marinou, Katerina Nikolaidis
	Samba Stars	Stavros Kosta, Anna Marinou, Christos Tsilios, Christos Karamitsos, Panagiotis Vlachos, Giorgos Pavlidis
	Pop Queens	Babis Heotis, Christos Lazarou
	The Vinyls	Irene Alexiou, Sofia Vlachos
	Neon Shadows	Panagiotis Nikolaidis, Anna Tsilios, Sofia Stathopoulou, Irene Petrakis, Panagiotis Vardis
	Moonlight Riot	Zoe Tsouroufli, Christos Vardis, Panagiotis Kalogeropoulos, Giorgos Kosta, Olga Vardis
	Funk Syndicate	Efi Tsilios, Dimitris Stathopoulou, Petros Karamitsos, Kostas Petrakis, Alex Marinou
	The Noisers	Katerina Nikolaidis, Stavros Andreou
	Groove Mecha...	Alex Ioannou, Irene Xenou, Maria Dimitriou
	Echo Hunters	Alexandra Karali, Eleni Andreou, Vasilis Petrou, Katerina Marinou, Stavros Andreou, Panagiotis Vlachos, Sofia Andreou
	Sunset Rebels	Kostas Stathopoulou, Giorgos Ioannou, Nikos Karamitsos, Panagiotis Vlachos, Zoe Marinou
	Wave Makers	Sofia Vardis
	The Frequencies	Giorgos Pavlidis

22. Ελέγχουμε αν κάποιος καλλιτέχνης είναι ήδη booked για ένα performance.

```

CREATE TRIGGER check_artist_overlap
BEFORE INSERT ON performance_artist
FOR EACH ROW
BEGIN
    DECLARE p_date DATE;
    DECLARE p_start TIME;
    DECLARE p_end TIME;
    DECLARE p_stage INT;
    DECLARE err_msg TEXT;

    SELECT performance_date, performance_start_time, performance_end_time
    INTO p_date, p_start, p_end
    FROM performance
    WHERE performance_id = NEW.performance_id;

    IF EXISTS (
        SELECT 1
        FROM performance_artist pa
        JOIN performance p ON p.performance_id = pa.performance_id
        WHERE pa.artist_id = NEW.artist_id
        AND p.performance_date = p_date
        AND (
            (p_start BETWEEN p.performance_start_time AND p.performance_end_time)
            OR
            (p_end BETWEEN p.performance_start_time AND p.performance_end_time)
            OR
            (p.performance_start_time BETWEEN p_start AND p_end)
            OR
            (p.performance_end_time BETWEEN p_start AND p_end)
        )
    ) THEN
        SET err_msg = CONCAT('The artist ',
        (SELECT CONCAT(a.artist_name, ' ', a.artist_surname)
        FROM artist a
        WHERE a.artist_id = NEW.artist_id),
        ' is already booked at the day ', p_date, ' and time ', p_start, ' - ', p_end, '');
    );
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = err_msg;
END IF;

```

Στόχος μας είναι να ελέξγουμε αν υπάρχει καλλιτέχνης, ο οποίος πάει να γίνει double booked, δηλαδή πάει να συνδεθεί με νέο performance που έχει χρονικό overlap με κάποιο performance που έχει συνδεθεί ήδη. Αυτό που κάνουμε είναι ότι πριν από μια εισαγωγή στον πίνακα performance_artist ελέγχουμε κάθε φορά αν υπάρχει κάποια εισαγωγή ήδη στον πίνακα αυτόν που κάποιο performance, στο οποίο συμμετέχει ο ίδιος artist με αυτόν που πάμε να κάνουμε insert, να είναι ίδια ημερομηνία και να συμπίπτει χρονικά (4 τρόποι) με το performance που πάμε να κάνουμε insert.

Αν βρεθεί τέτοιο performance τότε πέτα error και μην κάνεις insert.

Αυτό το trigger δεν χρειάζεται να το επαναλάβουμε και για τα συγκροτήματα, την περίπτωση δηλαδή του double-booking, γιατί ένας καλλιτέχνης δεν μπορεί να γίνει double-booked ára αυτομάτως ούτε και ένα συγκρότημα που περιέχει καλλιτέχνες.

23. Δημιουργούμε ένα view το οποίο περιέχει όλους τους artists και ποιες χρονιές έχει συμμετάσχει ο καθένας σε φεστιβάλ. Θα μας χρειαστεί στη συνέχεια.

```

CREATE OR REPLACE VIEW artist_participation_years AS
SELECT
    a.artist_id,
    a.artist_name,
    a.artist_surname,
    YEAR(p.performance_date) AS year
FROM artist a
JOIN performance_artist pa ON a.artist_id = pa.artist_id
JOIN performance p ON pa.performance_id = p.performance_id
ORDER BY a.artist_id, year;

```

Επιλέγουμε τη χρονία από το performance_date με το YEAR(), για κάθε artist φτάνοντας στον πίνακα performance που περιέχει το attribute performance_date, μέσω του πίνακα performance_artist.

24. Ελέγχουμε αν κάποιος καλλιτέχνης έχει συμμετάσχει στο φεστιβάλ πάνω από 3 συνεχόμενα χρόνια.

```

CREATE TRIGGER check_artist_consecutive_years
BEFORE INSERT ON performance_artist
FOR EACH ROW
BEGIN
    DECLARE new_year INT;      -- η χρονιά που τώρα γίνεται inserted και διερευνούμε
    DECLARE problem INT;       -- πάνω από 3 συνεχόμενα χρόνια

    SELECT YEAR(performance_date)
    INTO new_year
    FROM performance
    WHERE performance_id=NEW.performance_id;

    SELECT COUNT(*)
    INTO problem
    FROM (
        SELECT DISTINCT YEAR(p.performance_date) AS yearss
        FROM performance_artist pa
        JOIN performance p ON pa.performance_id=p.performance_id
        WHERE pa.artist_id=NEW.artist_id
        UNION
        SELECT new_year
    ) AS year
    WHERE yearss BETWEEN (new_year-3) AND new_year;

    IF problem = 4 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist 3 consecutive years';
    END IF;
END;

```

Δημιουργούμε ένα trigger το οποίο πριν από κάθε insert στον πίνακα performance_artist εκτελεί τον εξής έλεγχο. Φτιάχνουμε έναν μετρητή που τον ονομάζουμε problem πόσες φορές συμμετέχει συνεχόμενα ένας καλλιτέχνης στο φεστιβάλ. Βρίσκουμε πριν το insert που θέλουμε να κάνουμε πόσες φορές έχει ήδη συμμετάσχει ο καλλιτέχνης που θέλουμε να κάνουμε insert στο φεστιβάλ και το κάνουμε UNION με το νέο έτος που θέλουμε να εισάγουμε τώρα. Φυσικά δεν κρατάμε όλες τις χρονιές που έχει συμμετάσχει ο καλλιτέχνης αλλά μόνο τις χρονιές που απέχουν 3 χρόνια από τη νέα χρονιά, γιατί θέλουμε συνεχόμενες

χρονιές. Στο τέλος κάνουμε έναν έλεγχο αν ο μετρητής μας που είναι οι συνολικές συνεχόμενες χρονιές έχει γίνει 4 (πάνω από 3) και αν έχει συμβεί αυτό πετάμε error και δεν γίνεται ποτέ το insert.

25. Ελέγχουμε αν ένα performance που ανήκει σε ένα event είναι όντως μέσα στα χρονικά πλαίσια του event.

```

DELIMITER //
CREATE TRIGGER check_performance_event_time
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
    DECLARE e_date DATE;
    DECLARE e_start TIME;
    DECLARE e_end TIME;

    SELECT event_date, start_time, end_time
    INTO e_date, e_start, e_end
    FROM event
    WHERE event_id = NEW.event_id;

    IF NEW.performance_date <> e_date THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Performance date does not match event date';
    END IF;

    IF NEW.performance_start_time < e_start OR NEW.performance_end_time > e_end THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Performance time is outside event time range';
    END IF;

END;
// 
DELIMITER ;

```

Δημιουργούμε ένα trigger στο οποίο ελέγχουμε πριν κάνουμε νέο insert στον πίνακα performance αν έχει διαφορετικό date με το αντίστοιχο event με το οποίο συνδέεται, αν αρχίζει πριν το αντίστοιχο event ή τελειώνει μετά από αυτό. Σε κάθε μία από αυτές τις περιπτώσεις πετάει error και δεν γίνεται το insert.

26. Θέλουμε να δηλώσουμε το total participation του event με το performance, ότι δηλαδή ένα event δεν μπορεί να μείνει χωρίς performance (θέλουμε τουλάχιστον 1).

```

DELIMITER $$

CREATE PROCEDURE event_total_participation_performance()
BEGIN
    IF EXISTS (
        SELECT 1 FROM event e
        WHERE NOT EXISTS (
            SELECT 1 FROM performance p
            WHERE e.event_id = p.event_id
        )
    ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Every Event must have at least one performance.';
    END IF;
END$$

DELIMITER ;

```

Δημιουργούμε ένα procedure το οποίο και καλούμε και ελέγχουμε για κάθε event, αν υπάρχει event του οποίου το id δεν βρίσκεται σε κανένα performance. Αν συμβαίνει αυτό τότε πέτα error και το αντίστοιχο μήνυμα.

27. Ελέγχουμε ομοίως το total participation του band με τον artist, δηλαδή ότι κάθε band πρέπει να έχει τουλάχιστον έναν artist.

```

DELIMITER $$

CREATE PROCEDURE artist_total_participation_band()
BEGIN
    IF EXISTS (
        SELECT 1 FROM band b
        WHERE NOT EXISTS (
            SELECT 1 FROM artist_band ab
            WHERE ab.band_id = b.band_id
        )
    ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Every Band must have at least one artist.';
    END IF;
END$$

DELIMITER ;

```

28. Θέλουμε κάθε φορά το performance να παίρνει σαν δεδομένο και το stage στο οποίο πραγματοποιείται. Όμως η πληροφορία αυτή υπάρχει στον πίνακα event και όχι performance.

```

DELIMITER $$

CREATE TRIGGER trg_set_stage_id
BEFORE INSERT ON performance
FOR EACH ROW
BEGIN
    DECLARE stg_id INT;
    SELECT stage_id INTO stg_id FROM event WHERE event_id = NEW.event_id;
    SET NEW.stage_id = stg_id;
END$$

DELIMITER ;

```

Άρα δημιουργούμε ένα trigger, με το οποίο γεμίζουμε αυτόματα το stage_id σε κάθε insert του performance, το οποίο παίρνουμε από το event.

Μέρος Β: QUERIES ΕΡΩΤΗΜΑΤΩΝ

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

Για την επίλυση του ερωτήματος χρησιμοποιούμε JOIN στους πίνακες ticket, event, festival και payment_method, ώστε να φτάσουμε από το εισιτήριο μέχρι το αντίστοιχο φεστιβάλ και να γνωρίζουμε ποιος τρόπος πληρωμής χρησιμοποιήθηκε. Ο υπολογισμός του έτους γίνεται με τη συνάρτηση YEAR() εφαρμοσμένη στο start του φεστιβάλ. Στη συνέχεια, με GROUP BY δημιουργούμε συνδυασμούς ανά έτος και τρόπο πληρωμής, και με SUM(t.cost) υπολογίζουμε τα συνολικά χρήματα που συγκεντρώθηκαν σε κάθε περίπτωση.

```

SELECT
    YEAR(f.start) AS festival_year,
    pm.method_name AS payment_method,
    SUM(t.cost) AS total_revenue
FROM
    ticket t
    JOIN event e ON t.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
    JOIN payment_method pm ON t.id_payment_method = pm.id_payment_method
GROUP BY
    YEAR(f.start), pm.method_name
ORDER BY
    festival_year, payment_method;

```

Το τελικό αποτέλεσμα, απόσπασμα του οποίου φαίνεται και παρακάτω, εμφανίζει ανά γραμμή: το έτος, τον τρόπο πληρωμής και το συνολικό ποσό εσόδων που συγκεντρώθηκε. Το ORDER BY φροντίζει ώστε οι εγγραφές να εμφανίζονται ταξινομημένες πρώτα χρονικά και μετά αλφαριθμητικά με βάση τον τρόπο πληρωμής, για καλύτερη αναγνωσιμότητα.

festival_year	payment_method	total_revenue
2018	Bank Transfer	2237.71
2018	Cash	3062.59
2018	Credit Card	3265.05
2018	Debit Card	3664.40
2019	Bank Transfer	1937.18
2019	Cash	2265.55
2019	Credit Card	2436.93
2019	Debit Card	2764.96
2020	Bank Transfer	3371.89
2020	Cash	2974.11
2020	Credit Card	2987.90
2020	Debit Card	2697.71
2021	Bank Transfer	2204.93
2021	Cash	2758.78
2021	Credit Card	2645.32
2021	Debit Card	3422.36
2022	Bank Transfer	2252.82
2022	Cash	2177.10
2022	Credit Card	2334.68
2022	Debit Card	2581.44
2023	Bank Transfer	2116.22
2023	Cash	2638.41
2023	Credit Card	2969.52
2023	Debit Card	2444.65
2024	Bank Transfer	1084.24

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 επιλέγουμε ένα τυχαίο έτος (2024) και ένα τυχαίο μουσικό είδος (Rock) και βλέπουμε πόσοι από τους καλλιτέχνες που συμμετείχαν στο φεστιβάλ το έτος αυτό ανήκουν στο είδος αυτό.

```

SELECT
    a.artist_id,
    a.artist_name,
    a.artist_surname,
    mg.music_genre,
    CASE
        WHEN EXISTS (
            SELECT 1
            FROM performance_artist pa
            JOIN performance p ON pa.performance_id = p.performance_id
            JOIN event e ON p.event_id = e.event_id
            JOIN festival f ON e.festival_id = f.festival_id
            WHERE pa.artist_id = a.artist_id AND YEAR(f.start) = 2024
        )
        THEN 'Yes'
        ELSE 'No'
    END AS participated_in_2024
FROM artist a
JOIN artist_genre ag ON a.artist_id = ag.artist_id
JOIN music_genre mg ON ag.music_genre_id = mg.music_genre_id
WHERE mg.music_genre = 'Rock';

```

Για να το πετύχουμε αυτό κάνουμε select το ονοματεπώνυμο και το είδος μουσικής, όπως και επίσης το αν έχει συμμετάσχει στο festival ο καλλιτέχνης εκείνη τη χρονιά ή όχι. Για να ελέγχουμε το τελευταίο πάμε από τον performance_artist πίνακα, στον performance, στο event και τελικά στο φεστιβάλ όπου και παίρνουμε τη χρονιά του φεστιβάλ για τον καλλιτέχνη

που θέλουμε. Αν έχει πράγματι συμμετάσχει στη στήλη participated_in_2024 εμφανίζεται το αντίστοιχο μήνυμα.

	artist_id	artist_name	artist_surname	music_genre	participated_in_2024
▶	19	Irene	Petrakis	Rock	No
	27	Nikos	Lazarou	Rock	Yes
	41	Stavros	Andreou	Rock	No
	46	Panagiotis	Vlachos	Rock	No
	48	Olga	Vardis	Rock	No

3. Βρείτε πτοιοι καλλιτέχνες έχουν εμφανιστεί ως warm up περισσότερες από 2 φορές στο ίδιο φεστιβάλ;

► **SELECT**

```

a.artist_id,
a.artist_name,
a.artist_surname,
f.festival_id,
COUNT(*) AS warmup_count
FROM performance p
JOIN performance_kind pk ON p.performance_kind_id = pk.performance_kind_id
JOIN performance_artist pa ON p.performance_id = pa.performance_id
JOIN artist a ON pa.artist_id = a.artist_id
JOIN event e ON p.event_id = e.event_id
JOIN festival f ON e.festival_id = f.festival_id
WHERE pk.performance_kind_id = 1
GROUP BY a.artist_id, f.festival_id
HAVING COUNT(*) > 2;

```

Εμφανίζουμε το id του καλλιτέχνη, το όνομα του, το επίθετο του, το id του φεστιβάλ και τον αριθμό που ήταν warm up στο συγκεκριμένο φεστιβάλ.

Από τον πίνακα performance κάνουμε join για να πάρουμε το είδος του performance, το id του καλλιτέχνη, τα στοιχεία του, το event και τελικά το festival στο οποίο βρίσκεται.

Παίρνουμε μόνο όσους έχουν performance_kind_id ίσο με 1 (warm up) με το where. Έπειτα, με το group by ομαδοποιούμε τα στοιχεία ανά καλλιτέχνη και φεστιβάλ. Έτσι δεν παίρνουμε τα warm ups όλων των ετών, αλλά τα warm ups ανα έτος. Τέλος, αφού θέλουμε πάνω από 2 φορές στο ίδιο φεστιβάλ βάζουμε HAVING COUNT(>2).

artist_id	artist_name	artist_surname	festival_id	warmup_count
29	Stavros	Marinou	9	3
33	Maria	Dimitriou	1	3

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

Για τον σκοπό αυτό, χρησιμοποιείται JOIN των πινάκων artist, performance_artist και evaluation, ώστε να εντοπιστούν οι αξιολογήσεις που συνδέονται με τις εμφανίσεις του συγκεκριμένου καλλιτέχνη.

Ο υπολογισμός των μέσων τιμών γίνεται μέσω της AVG(), τόσο για το πεδίο artist_performance_rating (σχετικό με την καλλιτεχνική του παρουσία), όσο και για το overall_impression_rating (σχετικό με τη συνολική εμπειρία του κοινού). Η χρήση του φίλτρου WHERE a.artist_id = 4 περιορίζει το αποτέλεσμα σε έναν μόνο καλλιτέχνη.

```
SELECT
    a.artist_id,
    a.artist_name,
    a.artist_surname,
    AVG(e.artist_performance_rating) AS avg_artist_rating,
    AVG(e.overall_impression_rating) AS avg_overall_rating
FROM artist a
JOIN performance_artist pa ON a.artist_id = pa.artist_id
JOIN evaluation e ON pa.performance_id = e.performance_id
WHERE a.artist_id = 4
```

Το τελικό αποτέλεσμα επιστρέφει τον artist_id, το όνομα και το επώνυμό του, καθώς και τις δύο ζητούμενες μέσες τιμές.

	artist_id	artist_name	artist_surname	avg_artist_rating	avg_overall_rating
▶	4	Panagiotis	Nikolaidis	3.1000	2.8500

Για την εκτέλεση ερωτημάτων που περιλαμβάνουν JOIN μεταξύ πινάκων, η MySQL μπορεί να επιλέξει διαφορετικές στρατηγικές συνένωσης, ανάλογα με τη δομή των πινάκων, τα indexes, τον αριθμό των εγγραφών και τη στατιστική πληροφορία. Παρακάτω παρουσιάζονται οι τρεις κύριες τεχνικές συνένωσης:

1. Nested Loop Join

Σύμφωνα με αυτή τη τεχνική, για κάθε γραμμή του εξωτερικού πίνακα (outer table), ο μηχανισμός ελέγχει όλες τις γραμμές του εσωτερικού πίνακα (inner table) για να βρει τις αντιστοιχίες. Αν υπάρχει ευρετήριο στον inner πίνακα, ο έλεγχος μπορεί να επιταχυνθεί σημαντικά.

2. Hash Join

Σε αυτή τη στρατηγική, ο ένας πίνακας (συνήθως ο μικρότερος) φορτώνεται πρώτα στη μνήμη ως hash table, βάσει του κλειδιού συνένωσης. Στη συνέχεια, για κάθε γραμμή του άλλου πίνακα, αναζητείται αντιστοίχιση στον hash πίνακα.

Η MySQL υποστηρίζει Hash Join από την έκδοση 8.0.18 και μετά, και ενεργοποιείται μόνο αν οριστεί η επιλογή SET optimizer_switch='hash_join=on';. Είναι πιο αποδοτική από το nested loop για μεγάλους πίνακες χωρίς καλούς δείκτες, αλλά απαιτεί περισσότερη μνήμη.

3. Merge Join

Αυτή η μέθοδος προϋποθέτει ότι οι δύο πίνακες είναι προ-ταξινομημένοι με βάση τα πεδία συνένωσης. Η συνένωση υλοποιείται σαν "συγχώνευση δύο ταξινομημένων λιστών", με γραμμική πολυπλοκότητα. Αν και είναι θεωρητικά πολύ αποδοτική, η MySQL δεν την υποστηρίζει ακόμη εγγενώς όπως άλλες βάσεις (π.χ. PostgreSQL).

Για τις ανάγκες της άσκησης, θα πειραματιστούμε με την τεχνική Nested Loop Join με force index και χωρίς.

Αρχικά, εκτελέστηκε το ερώτημα χωρίς καμία παρέμβαση, με τη χρήση της εντολής EXPLAIN, ώστε να παρατηρηθεί το default execution plan. Όπως αναμενόταν, η MySQL επέλεξε Nested Loop Join για τις συνενώσεις μεταξύ artist, performance_artist και evaluation, λόγω του σχετικά μικρού όγκου δεδομένων και των υπαρχόντων πρωτευουσών και ξένων κλειδιών που εξυπηρετούν την πλοήγηση μεταξύ των πινάκων.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	pa	ref	PRIMARY,artist_id	artist_id	4	const	3	Using index
1	SIMPLE	e	ref	performance_id	performance_id	4	festivals.pa.performance_id	1	

Θέτωντας "SET profiling = 1;" και εκτελώντας "Show profiles" βλέπουμε ότι το query μας εκτελέστηκε πολύ γρήγορα:

	Query_ID	Duration	Query
▶	1	0.00070810	EXPLAIN SELECT a.artist_id, a.artist_name, ...

Στη συνέχεια, εφαρμόστηκε το ίδιο ερώτημα, αλλά με χρήση της εντολής FORCE INDEX, ώστε να επιβληθεί η χρήση συγκεκριμένων ευρετηρίων στις συνενώσεις:

```
EXPLAIN SELECT
    a.artist_id,
    a.artist_name,
    a.artist_surname,
    AVG(e.artist_performance_rating) AS avg_artist_rating,
    AVG(e.overall_impression_rating) AS avg_overall_rating
FROM artist a
FORCE INDEX (PRIMARY)
JOIN performance_artist pa FORCE INDEX (PRIMARY) ON a.artist_id = pa.artist_id
JOIN evaluation e FORCE INDEX (PRIMARY) ON pa.performance_id = e.performance_id
WHERE a.artist_id = 4;
```

Με χρήση της εντολής explain βλέπουμε

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	e	ALL	HULL	HULL	HULL	HULL	494	
1	SIMPLE	pa	eq_ref	PRIMARY	PRIMARY	8	festivals.e.performance_id,const	1	Using index

Συνολικά, η MySQL εφαρμόζει εδώ την τεχνική του Nested Loop Join, με το artist ως βασικό πίνακα και τα performance_artist και evaluation ως joined tables. Ο μηχανισμός εκμεταλλεύεται πλήρως τα primary keys όπου υπάρχουν φίλτρα, ενώ επιλέγει σάρωση

πίνακα (ALL) μόνο στο evaluation, όπου αυτό είναι αναμενόμενο. Η χρήση του FORCE INDEX επιβεβαιώνει ότι οι δείκτες που ζητήθηκαν χρησιμοποιήθηκαν και δεν παρακάμφθηκαν από τον βελτιστοποιητή. Στη συγκεκριμένη περίπτωση η διαφορά στην απόδοση δεν είναι αισθητή λόγω μικρού όγκου δεδομένων.

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

Για την υλοποίησή του, συσχετίζουμε πέντε πίνακες με JOIN: artist, performance_artist, performance, event, festival. Η αλυσίδα των συσχετίσεων μας επιτρέπει να καταγράψουμε τις εμφανίσεις κάθε καλλιτέχνη και να εντοπίσουμε μέσω των εμφανίσεων (performance) τα φεστιβάλ στα οποία συμμετείχε.

Η ηλικία κάθε καλλιτέχνη υπολογίζεται δυναμικά με χρήση της συνάρτησης `TIMESTAMPDIFF(YEAR, artist_birthdate, CURDATE())`, η οποία επιστρέφει τη διαφορά ετών μεταξύ της ημερομηνίας γέννησης του καλλιτέχνη και της τρέχουσας ημερομηνίας. Στη συνέχεια, εφαρμόζουμε φίλτρο WHERE `age < 30` ώστε να εξετάσουμε μόνο καλλιτέχνες κάτω των 30 ετών.

Για να μετρήσουμε το πλήθος φεστιβάλ, χρησιμοποιούμε `COUNT(DISTINCT f.festival_id)`, ώστε να υπολογίσουμε σε πόσα διαφορετικά φεστιβάλ έχει εμφανιστεί ο κάθε καλλιτέχνης. Αυτό καταγράφεται ως `festivals_participated`.

Τέλος, για να εντοπίσουμε τον καλλιτέχνη με τις περισσότερες συμμετοχές, χρησιμοποιούμε υποερώτηση στο HAVING όπου συγκρίνουμε τον αριθμό συμμετοχών κάθε καλλιτέχνη με τη μέγιστη τιμή που προκύπτει από ξεχωριστό SELECT. Έτσι διασφαλίζεται ότι επιστρέφεται μόνο ο καλλιτέχνης (ή οι καλλιτέχνες, αν υπάρχει ισοβαθμία) με τις περισσότερες συμμετοχές από όλους τους κάτω των 30 ετών.

Η επιστροφή των αποτελεσμάτων περιλαμβάνει τα πεδία: `artist_id`, `artist_name`, `artist_surname`, `age` και `festivals_participated`, ταξινομημένα αλφαριθμητικά για ευανάγνωση παρουσίαση.

```

SELECT
    a.artist_id,
    a.artist_name,
    a.artist_surname,
    TIMESTAMPDIFF(YEAR, a.artist_birthdate, CURDATE()) AS age,
    COUNT(DISTINCT f.festival_id) AS festivals_participated
FROM
    artist a
    JOIN performance_artist pa ON a.artist_id = pa.artist_id
    JOIN performance p ON pa.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
WHERE
    TIMESTAMPDIFF(YEAR, a.artist_birthdate, CURDATE()) < 30
GROUP BY
    a.artist_id, a.artist_name, a.artist_surname, a.artist_birthdate

HAVING
    COUNT(DISTINCT f.festival_id) = (
        SELECT MAX(festival_count) FROM (
            SELECT
                a2.artist_id,
                COUNT(DISTINCT f2.festival_id) AS festival_count
            FROM
                artist a2
                JOIN performance_artist pa2 ON a2.artist_id = pa2.artist_id
                JOIN performance p2 ON pa2.performance_id = p2.performance_id
                JOIN event e2 ON p2.event_id = e2.event_id
                JOIN festival f2 ON e2.festival_id = f2.festival_id
            WHERE
                TIMESTAMPDIFF(YEAR, a2.artist_birthdate, CURDATE()) < 30
            GROUP BY a2.artist_id
        ) AS counts
    )
ORDER BY a.artist_name;

```

	artist_id	artist_name	artist_surname	age	festivals_participated
	46	Panagiotis	Vlachos	26	6
	29	Stavros	Marinou	29	6

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

Το ερώτημα αυτό επιστρέφει για έναν συγκεκριμένο επισκέπτη (στην προκειμένη περίπτωση με id_visitor = 16) τις εκδηλώσεις στις οποίες έχει υποβάλει αξιολογήσεις, μαζί με τον μέσο όρο δύο βασικών κατηγοριών: της καλλιτεχνικής απόδοσης και της συνολικής εντύπωσης. Μέσω συνδέσεων (JOIN) μεταξύ των πινάκων evaluation, performance και event, εντοπίζονται οι εκδηλώσεις στις οποίες ανήκουν οι εμφανίσεις που έχει αξιολογήσει ο επισκέπτης. Η ομαδοποίηση πραγματοποιείται ανά εκδήλωση και με τη χρήση της AVG() υπολογίζονται οι μέσοι όροι των αντίστοιχων αξιολογήσεων.

```

SELECT
    ev.event_id,
    ev.event_name,
    ev.event_date,
    AVG(e.artist_performance_rating) AS avg_artist_rating,
    AVG(e.overall_impression_rating) AS avg_overall_rating
FROM evaluation e
JOIN performance p ON e.performance_id = p.performance_id
JOIN event ev ON p.event_id = ev.event_id
WHERE e.id_visitor = 16
GROUP BY ev.event_id, ev.event_name, ev.event_date;

```

Όπως έχει εξηγηθεί και στο ερώτημα 4, πειραματιζόμαστε με το Nested Loop Join, χρησιμοποιώντας τόσο το default όσο και το forced index.

Αρχικά με χρήση explain βλέπουμε το παρακάτω:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	e	ref	PRIMARY,performance_id	PRIMARY	4	const	3	Using temporary; Using filesort
1	SIMPLE	p	eq_ref	PRIMARY,event_id	PRIMARY	4	festivals.e.performance_id	1	
1	SIMPLE	ev	eq_ref	PRIMARY	PRIMARY	4	festivals.p.event_id	1	

Ειδικότερα, ο πίνακας e (evaluation) είναι ο πρώτος που προσπελαύνεται, με type = ref, γεγονός που δείχνει ότι το φίλτρο WHERE e.id_visitor = 16 αξιοποιεί ευρετήριο. Χρησιμοποιείται το PRIMARY index στο id_visitor, και επιστρέφονται περίπου 3 σχετικές εγγραφές. Στο Extra εμφανίζεται το μήνυμα Using temporary; Using filesort, το οποίο δηλώνει ότι απαιτείται προσωρινός πίνακας και ταξινόμηση κατά την εκτέλεση του GROUP BY. Στη συνέχεια, ο πίνακας p (performance) προσπελαύνεται με eq_ref, χρησιμοποιώντας το PRIMARY, μέσω της συσχέτισης p.performance_id = e.performance_id. Η αναζήτηση είναι άμεση και επιστρέφει 1 γραμμή ανά εμφάνιση. Τέλος, ο πίνακας ev (event) προσπελαύνεται επίσης με eq_ref, πάλι μέσω PRIMARY, καθώς συσχετίζεται με το p.event_id. Και εδώ επιστρέφεται μία εγγραφή ανά εμφάνιση.

Εφαρμόζουμε τώρα την τεχνική των Force Indexes, όπως φαίνεται παρακάτω:

```

EXPLAIN SELECT
    ev.event_id,
    ev.event_name,
    ev.event_date,
    AVG(e.artist_performance_rating) AS avg_artist_rating,
    AVG(e.overall_impression_rating) AS avg_overall_rating
FROM evaluation e FORCE INDEX (PRIMARY)
JOIN performance p FORCE INDEX (PRIMARY) ON e.performance_id = p.performance_id
JOIN event ev FORCE INDEX (PRIMARY) ON p.event_id = ev.event_id
WHERE e.id_visitor = 16
GROUP BY ev.event_id, ev.event_name, ev.event_date;

```

Και εκτελώντας την explain βλέπουμε:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	e	ref	PRIMARY	PRIMARY	4	const	3	Using temporary; Using filesort
1	SIMPLE	p	eq_ref	PRIMARY	PRIMARY	4	festivals.e.performance_id	1	
1	SIMPLE	ev	eq_ref	PRIMARY	PRIMARY	4	festivals.p.event_id	1	

Είναι προφανές ότι δεν υπάρχει ουσιαστική διαφορά καθώς και στις δύο περιπτώσεις, η MySQL επέλεξε την τεχνική Nested Loop Join και προσπέλασε τους πίνακες evaluation, performance και event μέσω των αντίστοιχων primary κλειδιών. Η χρήση του FORCE INDEX απλώς επιβεβαίωσε τις επιλογές που ήδη έκανε σωστά ο optimizer. Αυτό είναι απόλυτα λογικό, καθώς τα δεδομένα είναι περιορισμένα, οι σχέσεις είναι primary-to-foreign key, και δεν υπάρχει εναλλακτικός δείκτης που να μπορεί να προκαλέσει σύγχυση ή εσφαλμένη επιλογή. Ως εκ τούτου, δεν παρατηρήθηκε κάποια διαφορά στην απόδοση.

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

Για το συγκεκριμένο χρησιμοποιούμε τα experience id για να μετρήσουμε τον μέσο όρο εμπειρίας. Experience_id = 1 είναι ειδικευόμενος και 5 πολύ έμπειρος. Έτσι, όσο χαμηλότερος μέσος όρος, τόσο χαμηλότερη είναι η εμπειρία.

```
• SELECT
    f.name,
    ROUND(AVG(exp.experience_id), 2) AS avg_experience
FROM personnel p
JOIN experience exp ON p.experience_id = exp.experience_id
JOIN role r ON p.role_id = r.role_id
JOIN stage_personnel sp ON p.personel_id = sp.personel_id
JOIN stage s ON sp.stage_id = s.stage_id
JOIN event ev ON s.stage_id = ev.stage_id
JOIN festival f ON ev.festival_id = f.festival_id
WHERE r.role= 'Technical'
GROUP BY f.festival_id
ORDER BY avg_experience ASC
LIMIT 1;
```

Επιλέγουμε το όνομα του φεστιβάλ και τον μέσο όρο του experience id ως avg_experience. Βάλαμε ακρίβεια 2 δεκαδικών ψηφίων.

Μετά, από τον πίνακα personnel με join experience και role παίρνουμε την εμπειρία και το ρόλο του κάθε υπαλλήλου. Έπειτα βρίσκουμε σε ποια σκηνή βρίσκεται, σε ποιο event και τελικά σε ποιο φεστιβάλ. Κρατάμε μόνο όσους είναι τεχνικό προσωπικό και ομαδοποιούμε ανά φεστιβάλ. Υπολογίζει τον μέσο όρο (στο select φυσικά), κάνει κατανομή με αύξουσα σειρά και τυπώνει μόνο 1, αφού θέλει το φεστιβάλ με τον χαμηλότερο μέσο όρο

	name	avg_experience
	NextWave	2.50

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

Το ερώτημα αυτό επιστρέφει τα στοιχεία του υποστηρικτικού προσωπικού (role = 'Support') που δεν εργάστηκε σε καμία σκηνή κατά την ημερομηνία 27/07/2019. Αρχικά, γίνεται σύνδεση του πίνακα personnel με τον πίνακα role ώστε να εντοπιστεί ποιο προσωπικό ανήκει στον ρόλο "Support". Στη συνέχεια, με χρήση της υποερώτησης και του NOT IN, αποκλείονται όσοι εμφανίζονται να έχουν ανατεθεί μέσω του πίνακα stage_personnel σε σκηνή (stage_id) που συμμετείχε σε εκδήλωση (event) εκείνη την ημέρα. Έτσι επιστρέφονται μόνο όσοι υποστηρικτικοί εργαζόμενοι δεν είχαν καμία ανάθεση σε event της συγκεκριμένης ημερομηνίας.

```

SELECT
    p.personel_id,
    p.name,
    p.last_name
FROM
    personnel p
    JOIN role r ON p.role_id = r.role_id
WHERE
    r.role = 'Support'
    AND p.personel_id NOT IN (
        SELECT DISTINCT sp.personel_id
        FROM
            stage_personnel sp
            JOIN event e ON sp.stage_id = e.stage_id
        WHERE
            e.event_date = '2019-07-27'
    );

```

personel_id	name	last_name
1	Βέρο	Σταυρόπουλος
5	Σαμουήλ	Κυρίτσης
7	Τρυφωνία	Δερλώνη
11	Μενέλαος	Ζαννίκου
14	Θεόκλητος	Μηραέσα
17	Θεοδόπη	Καραντζόπουλος
18	Δάφνη	Λώλος
19	Οδυσσέας	Κουτσονίκα
20	Ησαΐας	Κακοτρίχη
21	Αναξαγόρας	Μαγκαφοπούλου
24	Ελπίδα	Γκάγκα
26	Ροδαμάνθη	Αβραμιδης
30	Αντύπας	Λυμπέρης
32	Μιχαήλ	Ζενεμπίσης
41	Ιπποκράτης	Κιπιός
45	Θρασύβουλος	Κυριάκου
49	Προμηθέας	Βούκας
50	Μαρίνος	Βούρας

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

```

1 • ⓧ WITH visitor_counts AS (
2     SELECT
3         t.id_visitor,
4         YEAR(e.event_date) AS year,
5         COUNT(*) AS num_attended
6     FROM ticket t
7     JOIN event e ON t.event_id = e.event_id
8     GROUP BY t.id_visitor, YEAR(e.event_date)
9     HAVING COUNT(*) > 3
10 )
11
12     SELECT vc1.id_visitor, vc1.year, vc1.num_attended
13     FROM visitor_counts vc1
14     JOIN visitor_counts vc2
15     ON vc1.year = vc2.year
16     AND vc1.num_attended = vc2.num_attended
17     AND vc1.id_visitor <> vc2.id_visitor
18     ORDER BY vc1.year, vc1.num_attended;
19

```

Εκτελείται πρώτα το visitor_counts. Συγκεκριμένα, ενώνουμε τα εισιτήρια με τα events και ομαδοποιούμε με βάση το visitor id και την χρονιά των event. Με το count παίρνουμε το

άθροισμα των παρακολουθήσεων του κάθε επισκέπτη σε μια χρονιά και με το having count(*)>3 κρατάμε μόνο όσους έχουν πάνω από 3 παρακολουθήσεις εκείνη τη χρονιά.

Έπειτα, επιλέγουμε visitor id, year και αριθμό παρακολουθήσεων. Παίρνουμε κάθε συνδυασμό δύο επισκεπτών από το visitor_counts και κρατάμε μόνο όσους είναι την ίδια χρονιά, έχουν ίδιο αριθμό παρακολουθήσεων και δεν είναι το ίδιο άτομο. Ταξινομούμε με σειρά ανά έτος και αριθμό συμμετοχών.

id_visitor	year	num_attended
251	2019	4
485	2019	4

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.

Ένας artist έχει πάνω από ένα είδος μουσικής, άρα κατά τα insert του artist δημιουργούνται πολλοί συνδυασμοί από ζεύγη μουσικών ειδών που εμφανίζονται στους artists. Θέλουμε να μετρήσουμε τις εμφανίσεις του κάθε ζεύγους και να βρούμε τα 3 με τις περισσότερες εμφανίσεις.

```

> WITH artist_pairs AS (
    SELECT
        a1.artist_id,
        LEAST(a1.music_genre_id, a2.music_genre_id) AS genre1,
        GREATEST(a1.music_genre_id, a2.music_genre_id) AS genre2
    FROM artist_genre a1
    JOIN artist_genre a2 ON a1.artist_id = a2.artist_id AND a1.music_genre_id < a2.music_genre_id
),
artists_in_festival AS (
    SELECT DISTINCT pa.artist_id
    FROM performance_artist pa
    JOIN performance p ON pa.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
),
SELECT
    genre1,
    genre2,
    COUNT(DISTINCT ap.artist_id) AS num_artists
FROM artist_pairs ap
JOIN artists_in_festival af ON ap.artist_id = af.artist_id
GROUP BY genre1, genre2
ORDER BY num_artists DESC
LIMIT 3;

```

Αρχικά θέλουμε να βρούμε όλα τα δυνατά ζεύγη μουσικών ειδών που εμφανίζονται σε όλους τους artists. Για να το κάνουμε αυτό κάνουμε self-join τον πίνακα artist_genre, παίρνοντας κάθε συνδυασμό από ζεύγη ειδών για τον κάθε καλλιτέχνη. Αποφεύγουμε επίσης να αποθηκεύσουμε πχ το (Rock, Jazz) και το (Jazz, Rock) σαν ξεχωριστά ζεύγη αποθηκεύοντας πάντα σαν genre1 το μουσικό είδος με το μικρότερο music_genre_id, (χρησιμοποιούμε δηλαδή τους ελέγχους LEAST, GREATEST).

Στη συνέχεια επιλέγουμε όλους τους διαφορετικούς καλλιτέχνες που έχουν συμμετάσχει σε ένα τουλάχιστον φεστιβάλ, με τον έλεγχο distinct κάνοντας join από τον πίνακα artists μέχρι και τον festival.

Στο τέλος κάνουμε count τους διαφορετικούς (DISTINCT) artists που εμφανίζονται στο φεστιβάλ και παρουσιάζουν ένα ζεύγος μουσικών ειδών. Μετράμε δηλαδή πόσοι μοναδικοί καλλιτέχνες έχουν και τα δύο genres. Κατατάσσει τα ζεύγη με βάση το πλήθος των μοναδικών καλλιτεχνών σε φθίνουσα σειρά (DESC) και επιλέγει 3 (LIMIT 3), άρα επιλέγει τα top3.

	genre1	genre2	num_artists
▶	17	19	3
	4	14	2
	5	11	2

Βλέπουμε πράγματι ότι πρώτη θέση έχει το ζεύγος με 3 εμφανίσεις και μετά οι εμφανίσεις σε artists μειώνονται.

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

Μας ζητείται να βρούμε τους καλλιτέχνες που εμφανίστηκαν λιγότερο από τον πιο πολυ εμφανιζόμενο καλλιτέχνη στο φεστιβάλ. Για να το πετύχουμε αυτό, θα πρέπει να βρούμε αρχικά τον καλλιτέχνη με τις περισσότερες συμμετοχές σε φεστιβάλ. Μετά από διευκρίνιση θα θεωρήσουμε τις περισσότερες συμμετοχές σε φεστιβάλ, ως περισσότερα performances ενός καλλιτέχνη σε φεστιβάλ. Στη συνέχεια, αφού εντοπίσουμε αυτό θα πρέπει να βρούμε πόσες φορές έχει εμφανιστεί ο κάθε artist στο φεστιβάλ. Τέλος, θα κάνουμε τις συγκρίσεις και θα εμφανίσουμε με το UNION ALL και τον καλλιτέχνη με τις περισσότερες εμφανίσεις και τους καλλιτέχνες με τουλάχιστον 5 λιγότερες εμφανίσεις από το πλήθος των εμφανίσεων του πιο πολυεμφανιζόμενου καλλιτέχνη.

Γενικά το UNION ALL ενώνει τα αποτελέσματα από δύο ή περισσότερα SELECT και τα επιστρέφει μαζί κοινά.

```
WITH number_performances_artists AS (
    SELECT
        a.artist_id,
        a.artist_name,
        a.artist_surname,
        COUNT(*) AS total.Performances
    FROM artist a
    JOIN performance_artist pa ON a.artist_id = pa.artist_id
    JOIN performance p ON pa.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
    GROUP BY a.artist_id, a.artist_name, a.artist_surname
)
SELECT
    artist_id,
    artist_name,
    artist_surname,
    total.Performances,
    'Most Performances' AS status
FROM number_performances_artists np
WHERE total.Performances = (SELECT MAX(total.Performances) FROM number_performances_artists)

UNION ALL

SELECT
    np.artist_id,
    np.artist_name,
    np.artist_surname,
    np.total.Performances,
    'Performed at least 5 times less than the maximum' AS status
FROM number_performances_artists np
WHERE total.Performances <= (SELECT MAX(total.Performances) FROM number_performances_artists) - 5

ORDER BY total.Performances DESC;
```

Ορίζουμε τις στήλες που θέλουμε να δούμε στο select δηλαδή το id, το ονοματεπώνυμο του καλλιτέχνη, τον αριθμό των performances του και το status δηλαδή αν πρόκειται για τον artist

με τα περισσότερα performances ή τους υπόλοιπους. Κάνουμε select και μετράμε τα total_performances του κάθε artist με τον COUNT(*), κάνοντας join τους πίνακες από το performance_artist μέχρι και το festival. Έπειτα με τον έλεγχο MAX διαλέγουμε τον καλλιτέχνη με την μεγαλύτερη τιμή total performances και εμφανίζουμε το αντίστοιχο μήνυμα status και στη συνέχεια παίρνουμε όλους τους καλλιτέχνες που εμφανίζονται σε αριθμό performances μικρότερο ή ίσο των συνολικών performances του πιο πολυεμφανιζόμενου καλλιτέχνη, μειωμένων κατά 5. Κατατάσσουμε σε φθίνουσα σειρά τους καλλιτέχνες ανάλογα με τον αριθμό εμφανίσεων τους. Το αποτέλεσμα που παίρνουμε είναι το εξής:

artist_id	artist_name	artist_surname	total_performances	status
24	Anna	Marinou	11	Most Performances
49	Sofia	Andreou	6	Performed at least 5 times less than the maximum
27	Nikos	Lazarou	6	Performed at least 5 times less than the maximum
34	Vasilis	Petrou	6	Performed at least 5 times less than the maximum
18	Giorgos	Nikolaidis	6	Performed at least 5 times less than the maximum
16	Christos	Kosta	6	Performed at least 5 times less than the maximum
35	Petros	Karamitsos	6	Performed at least 5 times less than the maximum
2	Alexandra	Karali	6	Performed at least 5 times less than the maximum
42	Christos	Karamitsos	6	Performed at least 5 times less than the maximum
39	Katerina	Marinou	6	Performed at least 5 times less than the maximum
30	Katerina	Nikolaidis	6	Performed at least 5 times less than the maximum
43	Kostas	Papadopoulos	5	Performed at least 5 times less than the maximum
40	Christos	Tsiolis	5	Performed at least 5 times less than the maximum
47	Zoe	Marinou	5	Performed at least 5 times less than the maximum
31	Irene	Xenou	5	Performed at least 5 times less than the maximum
11	Petros	Alexiou	5	Performed at least 5 times less than the maximum
50	Giorgos	Pavlidis	5	Performed at least 5 times less than the maximum
21	Panagiotis	Vardis	5	Performed at least 5 times less than the maximum
45	Alex	Marinou	5	Performed at least 5 times less than the maximum
17	Alex	Ioannou	5	Performed at least 5 times less than the maximum
23	Giorgos	Ioannou	5	Performed at least 5 times less than the maximum
36	Kostas	Petrakis	5	Performed at least 5 times less than the maximum
20	Zoe	Andreou	4	Performed at least 5 times less than the maximum
8	Alex	Lazarou	4	Performed at least 5 times less than the maximum
25	Dimitris	Stathopoulou	4	Performed at least 5 times less than the maximum
14	Efi	Tsiolis	4	Performed at least 5 times less than the maximum
10	Panagiotis	Kalogeropoulos	3	Performed at least 5 times less than the maximum
15	Irene	Alexiou	3	Performed at least 5 times less than the maximum
4	Panagiotis	Nikolaidis	3	Performed at least 5 times less than the maximum
44	Sofia	Vlachos	3	Performed at least 5 times less than the maximum
26	Christos	Lazarou	3	Performed at least 5 times less than the maximum
32	Panagiotis	Marinou	3	Performed at least 5 times less than the maximum
12	Anna	Tsiolis	3	Performed at least 5 times less than the maximum
3	Zoe	Tsouroufli	2	Performed at least 5 times less than the maximum
1	Babis	Heotis	2	Performed at least 5 times less than the maximum
13	Sofia	Stathopoulou	2	Performed at least 5 times less than the maximum
37	Giorgos	Kosta	2	Performed at least 5 times less than the maximum
9	Zoe	Stathopoulou	2	Performed at least 5 times less than the maximum
48	Olga	Vardis	2	Performed at least 5 times less than the maximum

19	Irene	Petrakis	2	Performed at least 5 times less than the maximum
5	Christos	Vardis	2	Performed at least 5 times less than the maximum
28	Sofia	Vardis	1	Performed at least 5 times less than the maximum

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

Βρίσκουμε με βάση το constraint για τα capacities πόσοι τουλάχιστον απαιτούνται για κάθε ημέρα του φεστιβάλ. Για το τεχνικό προσωπικό θεωρούμε ότι χρειάζεται τουλάχιστον ένα άτομο ανά σκηνή. Επίσης στρογγυλοποιούμε προς τα πάνω για λόγους ασφαλείας. πχ. αν το capacity ειναι 56 το 2% ειναι 1.12 άρα βάζουμε 2 support.

```

• ⊖ WITH RECURSIVE seq AS (
    SELECT 0 AS n
    UNION ALL
    SELECT n + 1 FROM seq WHERE n + 1 <= 30
),

```

Εδώ έχουμε ένα αναδρομικό πίνακα ακολουθίας (CTE) που δημιουργεί ακέραιους αριθμούς από 0 έως 30. Τον χρειαζόμαστε για να αποθηκεύσουμε μετά τις μέρες του φεστιβαλ, καθώς έχουμε μόνο το πότε αρχίζει και τελειώνει, όχι τις ενδιάμεσες μέρες.

```

⊖ festival_days AS (
  SELECT
    f.festival_id,
    f.name AS festival_name,
    DATE_ADD(f.start, INTERVAL s.n DAY) AS festival_date
  FROM festival f
  JOIN seq s ON s.n <= DATEDIFF(f.end, f.start)
),

```

Εδώ παίρνουμε κάθε φεστιβάλ, υπολογίζουμε με το DATEDIFF πόσες μέρες διαρκεί και χρησιμοποιούμε την seq για να πάρουμε αριθμούς από το 0 έως το πόσο διαρκεί. πχ αν ειναι 07-05-2023 εως 10-05- 2023 παίρνουμε s.n=3. Μετά με το date_add του λέμε πάρε την ημερομηνία έναρξης του φεστιβάλ (f.start) και πρόσθεσε η ημέρες σε αυτήν, όπου η είναι από το seq. Έτσι, έχουμε κάθε μέρα ξεχωριστά.

```

    active_stages AS (
        SELECT
            fd.festival_name,
            fd.festival_date,
            s.stage_id,
            s.stage_name,
            s.capacity
        FROM festival_days fd
        JOIN event e ON e.festival_id = fd.festival_id AND e.event_date = fd.festival_date
        JOIN stage s ON s.stage_id = e.stage_id
    )
)

```

Έπειτα, βλέπουμε ποια stages είναι ενεργά. Επιλέγουμε το όνομα του φεστιβαλ και την ημερομηνία του από το festival days που φτιάξαμε πριν. Επίσης επιλέγουμε το stage id, stage name και το capacity από το stage. Μετά κάνουμε στο festival days join με τα events και τις σκηνές. Έτσι, παίρνουμε ανά ημέρα πόσες σκηνές είναι ενεργές.

```

SELECT
    festival_name,
    festival_date,
    SUM(CEIL(capacity * 0.05)) AS total_security_needed,
    SUM(CEIL(capacity * 0.02)) AS total_support_needed,
    SUM(1) AS total_technical_needed
FROM active_stages
GROUP BY festival_name, festival_date
ORDER BY festival_name, festival_date;

```

Τέλος, από το active stages παίρνουμε το όνομα του φεστιβαλ, την ημέρα, το stage id, το stage name και το capacity. Υπολογίζει πόσοι χρειάζονται σε κάθε σκηνή, με στρογγυλοποίηση προς τα πάνω (ceil). Έπειτα τα αθροίζουμε για να πάρουμε τον συνολικό αριθμό προσωπικού από κάθε ρόλο. Γιαυτό χρειάζεται και η ομαδοποίηση ανα φεστιβαλ και ανα ημέρα.

	festival_name	festival_date	total_security_need...	total_support_nee...	total_technical_nee...	
	Beats	2019-07-25	2	1	1	
	Beats	2019-07-26	2	1	1	
	Beats	2019-07-27	3	2	2	
	Future	2026-05-26	2	1	1	
	Future	2026-05-27	2	1	1	
	Future	2026-05-28	4	2	2	
	NextWave	2027-05-19	6	4	2	
	NextWave	2027-05-20	5	2	2	
	Pulse	2018-05-20	7	3	3	
	Pulse	2018-05-21	2	1	1	
	Pulse	2023-05-17	4	2	2	
	Pulse	2023-05-18	4	2	2	
	Rhythm	2020-07-02	2	1	1	
	Rhythm	2020-07-04	7	4	3	
	Sound Fest	2025-07-29	6	3	3	
	Sound Fest	2025-07-30	2	1	1	
	Sounds	2021-06-13	4	2	2	
	Sounds	2021-06-15	5	3	2	
	Stage	2024-07-28	3	2	1	
	Stage	2024-07-29	7	3	3	
	Vibes	2022-05-25	4	2	2	
	Vibes	2022-05-27	5	3	2	

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

```

SELECT
    a.artist_id,
    a.artist_name,
    a.artist_surname,
    COUNT(DISTINCT c.continent_id) AS num_of_continents
FROM artist a
JOIN performance_artist pa ON a.artist_id = pa.artist_id
JOIN performance p ON pa.performance_id = p.performance_id
JOIN event e ON p.event_id = e.event_id
JOIN festival f ON e.festival_id = f.festival_id
JOIN location l ON f.location_id = l.location_id
JOIN continent c ON l.continent_id = c.continent_id
GROUP BY a.artist_id
HAVING COUNT(DISTINCT c.continent_id) >= 3;

```

Από τον πίνακα artist με join με το performance_artist παίρνουμε τα performance id. Με join με το performance παίρνουμε τα event id. Με join με τα event παίρνουμε τα festival id. Με join με το location παίρνουμε το continent id και τέλος με join με το continent παίρνουμε την ήπειρο. Κάνουμε ομαδοποίηση ανά καλλιτέχνη. Κάνουμε count DISTINCT continent id, για να εξασφαλίσουμε ότι είναι σε διαφορετικές ηπείρους και βάζουμε having count > =3, αφού θέλουμε τουλάχιστον 3 διαφορετικές ηπείρους. Τέλος, κάνουμε select το artist id, artist name artist surname και το count το οποίο αποθηκεύσαμε στο num_of_continents

	artist_id	artist_name	artist_surna...	num_of_continue...
	2	Alexandra	Karali	4
	4	Panagiotis	Nikolaidis	3
	6	Stavros	Kosta	5
	7	Kostas	Stathopoulou	4
	8	Alex	Lazarou	3
	11	Petros	Alexiou	3
	12	Anna	Tsiolis	3
	15	Irene	Alexiou	3
	16	Christos	Kosta	4
	17	Alex	Ioannou	3
	18	Giorgos	Nikolaidis	4
	20	Zoe	Andreou	3
	21	Panagiotis	Vardis	3
	22	Eleni	Andreou	4
	23	Giorgos	Ioannou	4
	24	Anna	Marinou	4
	26	Christos	Lazarou	3
	27	Nikos	Lazarou	4
	29	Stavros	Marinou	4
	30	Katerina	Nikolaidis	4
	31	Irene	Xenou	3
	32	Panagiotis	Marinou	3
	33	Maria	Dimitriou	4
	34	Vasilis	Petrou	4
	38	Nikos	Karamitsos	5
	39	Katerina	Marinou	4
	40	Christos	Tsiolis	3
	41	Stavros	Andreou	4
	42	Christos	Karamitsos	4
	43	Kostas	Papadopoulos	3
	46	Panagiotis	Vlachos	4
	47	Zoe	Marinou	4
	49	Sofia	Andreou	4
	50	Giorgos	Pavlidis	3

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

Στο query αυτό ψάχνουμε να βρούμε τα μουσικά είδη ποια από τα μουσικά είδη είχαν ίδιο αριθμό performances δύο συνεχόμενα έτη, με πάνω από 3 εμφανίσεις ανά έτος. Δηλαδή μια αναμενόμενη απάντηση θα ήταν “Jazz, 2019-2020, 4 εμφανίσεις κάθε χρόνο”.

```

> WITH genre_year_counts AS (
  SELECT
    mg.music_genre,
    YEAR(p.performance_date) AS year,
    COUNT(*) AS num_appearances
  FROM music_genre mg
  JOIN artist_genre ag ON mg.music_genre_id = ag.music_genre_id
  JOIN artist a ON ag.artist_id = a.artist_id
  JOIN performance_artist pa ON a.artist_id = pa.artist_id
  JOIN performance p ON pa.performance_id = p.performance_id
  GROUP BY mg.music_genre, YEAR(p.performance_date)
),
genre_consecutive_years AS (
  SELECT
    g1.music_genre,
    g1.year AS year1,
    g2.year AS year2,
    g1.num_appearances
  FROM genre_year_counts g1
  JOIN genre_year_counts g2
  ON g1.music_genre = g2.music_genre
  AND (g2.year = g1.year + 1)
  AND g1.num_appearances = g2.num_appearances
  AND g1.num_appearances >= 3
),
SELECT DISTINCT
  music_genre,
  year1,
  year2,
  num_appearances
FROM genre_consecutive_years
ORDER BY music_genre, year1;

```

Μετράμε το πλήθος των εμφανίσεων κάθε genre για κάθε έτος με το genre_year_counts, παίρνοντας τη χρονιά από το performance_date και κάνοντας count. Στη συνέχεια πάμε να συγκρίνουμε ποια από αυτά τα είδη είχαν συνεχόμενες χρονιές ίδιες εμφανίσεις με το genre_consecutive_years. Κάνουμε inner join στον πίνακα genre_year_counts, παίρνουμε λοιπόν δύο χρονιές και συγκρίνουμε για το ίδιο music_genre (δηλαδή με ίδιο music_genre_id), αν ο αριθμός των appearances είναι ίδιος, αν οι χρονιές είναι συνεχόμενες ($g2.year=g1.year+1$) και αν ο αριθμός των εμφανίσεων είναι μεγαλύτερος του 3. Αν ισχύουν τα κριτήρια αυτά κάνουμε select και εμφανίζουμε τα στοιχεία των genres και των ετών για τα οποία ισχύουν τα παραπάνω.

Το αποτέλεσμα μας:

	music_genre	year1	year2	num_appearances
▶	Ambient	2021	2022	3
	Blues	2021	2022	4
	Funk	2018	2019	4
	Reggae	2022	2023	3
	Reggae	2023	2024	3
	Reggae	2025	2026	5

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

Το συνολικό σκορ βαθμολόγησης θεωρήθηκε, σύμφωνα με διευκρινήσεις, το άθροισμα της βαθμολόγησης του καλλιτέχνη με την συνολική εντύπωση για όλες τις αξιολογήσεις που έχει δώσει ο επισκέπτης στον καλλιτέχνη αυτόν.

SELECT

```
v.first_name AS visitor_first_name,
v.last_name AS visitor_last_name,
a.artist_name,
a.artist_surname,
AVG(e.artist_performance_rating + e.overall_impression_rating) AS avg_score
FROM evaluation e
JOIN performance_artist pa ON e.performance_id = pa.performance_id
JOIN artist a ON pa.artist_id = a.artist_id
JOIN visitor v ON e.id_visitor = v.id_visitor
GROUP BY e.id_visitor, pa.artist_id
ORDER BY avg_score DESC
LIMIT 5;
```

Από τον πίνακα evaluation κάνουμε join με το performance_artist και τον artist για να πάρουμε το όνομα και το επίθετο του καλλιτέχνη. Έπειτα κάνουμε join με τους visitor για να δούμε τις αξιολογήσεις που έχουν δώσει στους καλλιτέχνες. Κάνουμε group ανά visitor και καλλιτέχνη. Αθροίζουμε τις τιμές των πεδίων artist_performance_rating και overall_impression rating και κάνουμε AVG, καθώς ένας επισκέπτης μπορεί να έχει ξαναπαρακολουθήσει τον καλλιτέχνη και να έχει παραπάνω από μια αξιολόγηση. Κάνουμε order σε φθίνουσα σειρά και κρατάμε 5, αφού θέλουμε τους top 5 επισκέπτες. Τέλος, κάνουμε select το όνομα και επίθετο του visitor και του artist, καθώς και το συνολικό σκορ

visitor_first_na...	visitor_last_na...	artist_name	artist_surna...	avg_score
Kenneth	Moore	Christos	Karamitsos	10.0000
George	Foster	Giorgos	Nikolaidis	10.0000
George	Foster	Katerina	Nikolaidis	10.0000
Tony	Daugherty	Petros	Karamitsos	10.0000
Kristen	Brown	Christos	Kosta	10.0000

Dummy data

Για να κάνουμε dummy data generation χρησιμοποιήσαμε chat gpt, όπου δημιουργούσε json files με τα επιθυμητά δεδομένα. Για να τα κάνουμε import στην βάση χρησιμοποιήσαμε

python script το οποίο κάνει connect στη βάση, διαβάζει τα json files και κάνει τα δεδομένα insert into στα tables της βάσης μας με loop. Αυτό είναι πολύ πιο γρήγορο από το να γράψουμε χειροκίνητα κάθε εντολή INSERT INTO.

Κάναμε generate τα εξής : 50 καλλιτέχνες/συγκροτήματα, 30 σκηνές, 100 εμφανίσεις, 10 φεστιβάλ (εκ των οποίων 2 μελλοντικά) και αντί για 200 φτιάξαμε 972 εισιτήρια, ώστε να είναι πιο ρεαλιστικά τα νούμερα παρακολούθησης κάθε σκηνής.

Επίσης, φτιάξαμε 40 events, 7 equipments, 50 personnel, 15 bands, 20 genres, 20 subgenres, 800 visitors, 20 buyers, 23 στο buyer_queue και 494 evaluations.

Όλα προφανώς τηρούν τα constraints της εργασίας.