

**Βάσεις Δεδομένων**

Αναφορά Εξαμηνιαίας Εργασίας

Music Festival - Pulse University

Φαίδων-Κορνέλις Κουρουνάκης (el20035)

# Περιεχόμενα

[Περιεχόμενα 2](#_hi32mfu0mba7)

[Εισαγωγή 3](#_y02hp9jrpxb4)

[Τεχνολογίες Υλοποίησης 3](#_32zvg1l9x4uz)

[Δομή Αρχείων 3](#_87otjzrwst7f)

[Οδηγίες Χειρισμού 4](#_72fq7py7et0c)

[Αναγκαίοι Περιορισμοί Εκφώνησης 5](#_m6x0s73xddcb)

[Υποθέσεις Υλοποίησης 5](#_idux95yqk66c)

[Entity Relationship Diagram 7](#_qkbp6jllgval)

[Relational Diagram 8](#_racimnbpvpo8)

[SQL DDL Script 9](#_nbwbyr2m54kn)

[Χρήση Ευρετηρίων και Στρατηγικών Join 21](#_yet5v2wpe6xj)

[Ερώτημα 4. 21](#_wmeho170qj9o)

[Ερώτημα 6. 25](#_4xshtt1mjmey)

[Σύνοψη Αποτελεσμάτων 32](#_dsw3m1ifzrn8)

# 

# Εισαγωγή

Η παραδοτέα βάση δεδομένων μπορεί να φιλοξενήσει δεδομένα για φεστιβάλ μουσικής, τις τοποθεσίες τους, τις σκηνές, τις παραστάσεις και τις εμφανίσεις, τους καλλιτέχνες και τα συγκροτήματα, τα μουσικά είδη και υποείδη τους, τους επισκέπτες, τα εισιτήρια, τις αξιολογήσεις, τις μεταπωλήσεις και σχετικες εικόνες!

Η αναφορά περιέχει τα απαιτούμενα κατά την εκφώνηση: Βήματα υλοποίησης, τα διαγράμματα ER, Relational, το DDL Script (install.sql) και τις αναλύσεις για τα ερωτήματα 4 και 6.

Το αρχείο sql/load.sql ήταν πολύ μεγάλο για να συμπεριληφθεί στο [repository](https://github.com/FaidonKourounakis/ntua-db), καθώς το Github έχει όριο μεγέθους αρχείου τα 100MB. Γι' αυτό χωρίστηκε σε πολλαπλά κομμάτια sql/load\_part\_x.sql. Μπορεί να παραχθεί ντετερμινιστικά με την εντολή $ cd code && ./make\_load\_sql.sh ή εναλλακτικά να ανακατασκευαστεί από τα κομμάτια του με την εντολή $ reconstruct\_load\_sql.sh (σε συστήματα Unix).

# Τεχνολογίες Υλοποίησης

Η βάση γράφτηκε για **Postgresql**, ενώ για την δημιουργία ψεύτικων δεδομένων με σκοπό την δοκιμασία των ερωτημάτων και την επιβεβαίωση της ορθής λειτουργίας γράφτηκε ένα script σε **Python** με την χρήση της βιβλιοθήκης **Faker**. Τέλος γράφτηκαν κάμποσα scripts σε **Bash** για την αυτοματοποίηση αρκετών λειτουργιών, όπως η δημιουργία των ψευδών δεδομένων, το τρέξιμο των queries ή άλλου αρχείου SQL, και την αρχικοποίηση της βάσης.

# Δομή Αρχείων

* **code**
  + **adjust\_data\_for\_Q14.sql** Κώδικας SQL τροποποίησης της βάσης με σκοπό την παραγωγή αποτελεσμάτων στο ερώτημα 14 (δες Ερώτημα 14)
  + **make\_load\_sql.sh** Κώδικας για τη αυτόματη κατασκευή του load.sql
  + **split\_load\_sql.sh** Κώδικας για διαχωρισμό του load.sql σε κομμάτια ~42MB load\_part\_x.sql
  + **reconstruct\_load\_sql.sh** Κώδικας για τη ανακατασκευή του load.sql από τα κομμάτια load\_part\_x.sql
  + **print\_fake\_data.py** Κώδικας παραγωγής πλαστών δεδομένων
  + **run\_queries.sh** Αυτόματη εκτέλεση όλων των ερωτημάτων
  + **run\_sql.sh** Αυτόματη κτέλεση ενός αρχείου sql
  + **setup\_db.sh** Αρχικοποίηση της βάσης (δημιουργία χρήστη, βάσης, τρέξιμο install.sql, load.sql)
  + **delete\_db.sh** Διαγραφή της βάσης (επικίνδυνο)
  + **start\_postgres.sh** Έναρξη εξυπηρετητή Postgresql
  + **stop\_postgres.sh** Τερματισμός εξυπηρετητή Postgresql
* **diagrams**
  + **er.pdf** Το διάγραμμα Entity-Relationship για σχεδιασμό των οντοτήτων
  + **relational.pdf** Το Relational diagram για οπτικοποίηση της βάσης
* **docs**
  + **report.pdf** Η συγκεκριμένη αναφορά.
* **sql**
  + **install.sql** Ορισμός σχήματος (πινάκων), ευρετηρίων, και περιορισμών (σε check & triggers)
  + **load.sql** Παραγόμενο αρχείο με (πολλά) ψεύτικα δεδομένα προς φόρτωση
  + **QXX.sql** Κώδικας υλοποίησης ερωτήματος ΧΧ
  + **QXX\_out.txt** Έξοδος ερωτήματος XX
* **README.md** Περίληψη της εργασίας

# Οδηγίες Χειρισμού

1. Εγκαταστήστε τα **Postgres**, **postgres-contrib**, **Python** και τη βιβλιοθήκη **faker** για Python.
2. Αλλάξτε κατάλογο με την εντολή $ cd code για να εκτελέσετε τα αυτοματοποιημένα σκριπτάκια.
3. Εκτελέστε το ./make\_load\_sql.sh για να δημιουργηθεί το αρχείο load.sql με όλες τις εντολές εισαγωγής (insert). Αυτό ουσιαστικά συνδυάζει την έξοδο του print\_fake\_data.py με το adjust\_data\_for\_Q14.sql σε ένα ενιαίο, τεράστιο αρχείο load.sql. Ανάλογα με τις ρυθμίσεις (ορίζονται στην αρχή του print\_fake\_data.py), αυτή η διαδικασία μπορεί να πάρει χρόνο και το τελικό αρχείο μπορεί να έχει μέγεθος εκατοντάδων megabyte.
4. Εναλλακτικά, μπορείτε να χωρίσετε/ανακατασκευάσετε το ενιαίο load.sql από/σε κομμάτια load\_part\_x.sql χρησιμοποιώντας τα scripts ./split\_load\_sql.sh/./reconstruct\_load\_sql.sh αντίστοιχα.
5. Εκτελέστε το ./setup\_db.sh για να δημιουργηθεί χρήστης, βάση δεδομένων, να εγκατασταθεί το σχήμα (όπως ορίζεται στο sql/install.sql) και να εισαχθούν όλα τα δεδομένα (sql/load.sql).
6. Εκτελέστε το ./run\_queries.sql για να τρέξουν όλα τα ερωτήματα QXX.sql και να δημιουργηθούν τα αρχεία αποτελεσμάτων τους.

# Αναγκαίοι Περιορισμοί Εκφώνησης

Στην εκφώνηση εντοπίσαμε τους εξής περιορισμούς που λάβαμε υπόψιν:

1. Ένα Φεστιβάλ ανά έτος, επομένως το έτος της ημερομηνίας έναρξης και λήξης είναι το ίδιο και μοναδικό. Για τον αποτελεσματικό έλεγχο μοναδικότητας, δημιουργείται μια υπολογιζόμενη στήλη "έτος" με περιορισμό μοναδικότητας (unique).
2. Κάθε σκηνή φιλοξενεί το πολύ ένα γεγονός (event) τη φορά.
3. Προσωπικό εκδηλώσεων: Η Ασφάλεια πρέπει να είναι ≥ 5% της χωρητικότητας της σκηνής, οι Βοηθοί ≥ 2% της χωρητικότητας της σκηνής (υλοποιείται μέσω trigger στους πίνακες events/event\_personnel).
4. Τα χρονικά διαλείμματα ανάμεσα στις εμφανίσεις ενός event πρέπει να είναι μεταξύ 5 και 30 λεπτών. (Ελέγχεται κατά την προσθήκη/τροποποίηση εμφάνισης).
5. Η διάρκεια μιας εμφάνισης (performance) δεν μπορεί να υπερβαίνει τις 3 ώρες (υλοποίηση με check).
6. Τα Events και τα Φεστιβάλ δεν μπορούν να ακυρωθούν (η διαγραφή δεν επιτρέπεται μέσω ON DELETE RESTRICT στα foreign keys προς τους 2 πίνακες event/festival).
7. Οι καλλιτέχνες δεν μπορούν να εμφανίζονται ταυτόχρονα σε πολλαπλές σκηνές. (Trigger που διασφαλίζει πως οι εμφανίσεις των καλλιτεχνών δεν επικαλύπτονται χρονικά).
8. Οι καλλιτέχνες δεν μπορούν να εμφανιστούν σε περισσότερα από 3 συνεχόμενα έτη.
9. Ο αριθμός των εισιτηρίων ανά event δεν μπορεί να υπερβαίνει τη χωρητικότητα της σκηνής.
10. Ο αριθμός VIP εισιτηρίων δεν μπορεί να ξεπερνά το 10% της χωρητικότητας.
11. Η μεταπώληση εισιτηρίων γίνεται με τον πρώτο διαθέσιμο συνδυασμό πωλητή και αγοραστή.

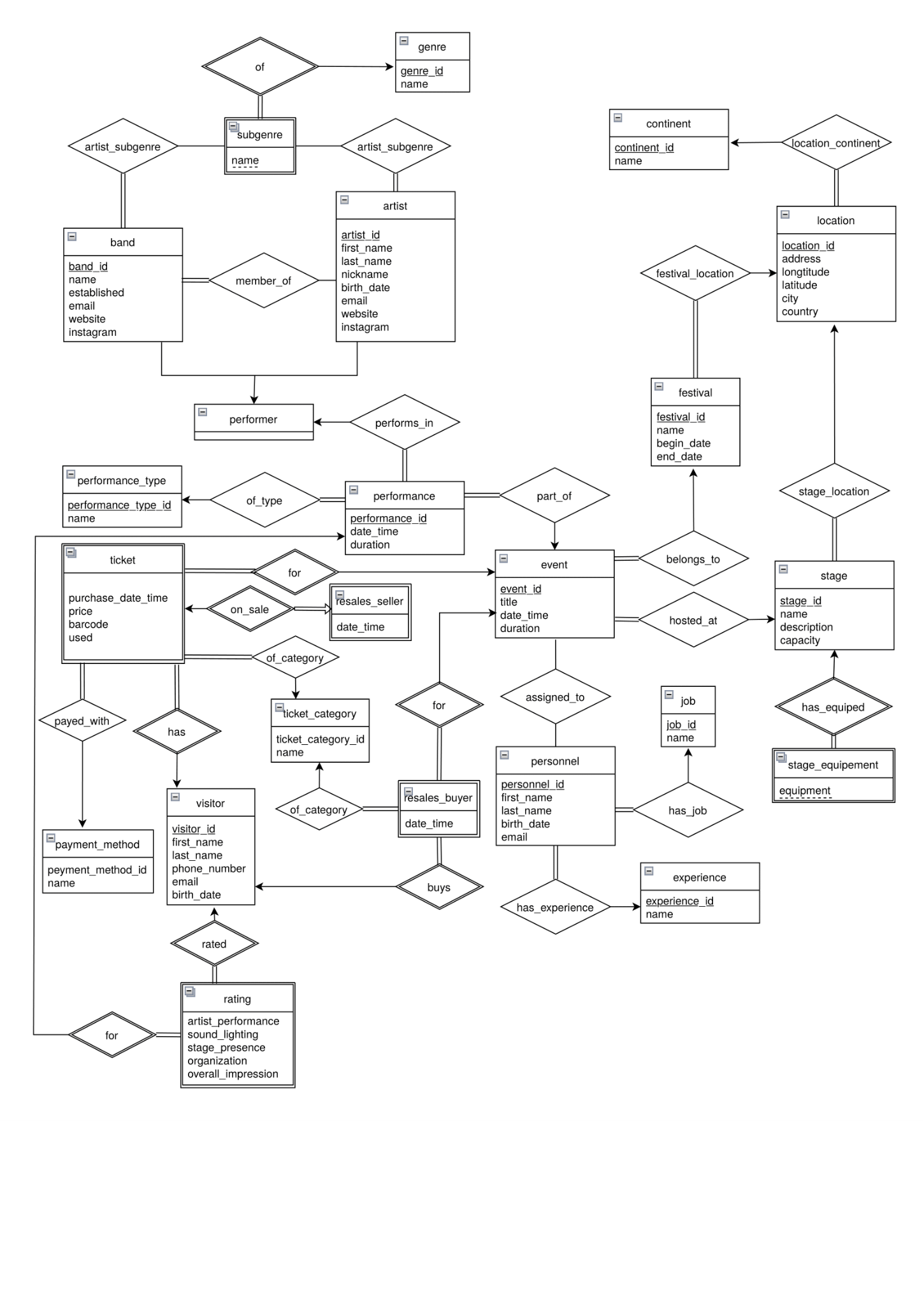
# Υποθέσεις Υλοποίησης

Κατά την δημιουργία του σχήματος, της βάσης και των δεδομένων, λαβαμε τις εξής επιπλέον υποθέσεις:

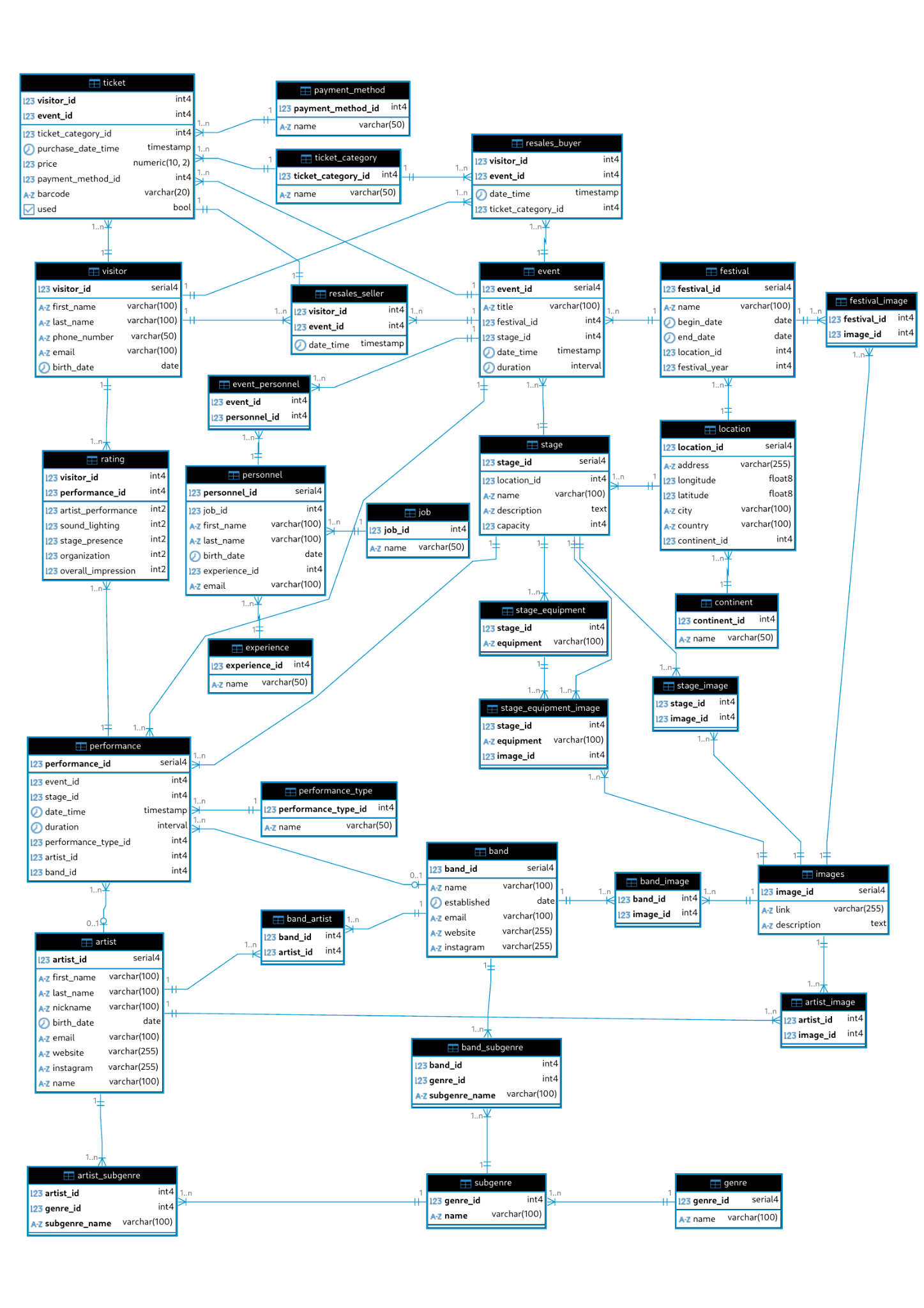
1. Τα αναγνωριστικά (IDs) των οντοτήτων είναι σειριακά (SERIAL Postgresql Type) για απλότητα (υποθέτουμε ένα μη κατανεμημένο σύστημα).
2. Η 'Σκηνή' (Stage) είναι μια νέα οντότητα.
3. Ο ρόλος προσωπικού (Personnel Job) παίρνει (προς το παρόν) τις τιμές Assistant/Security/Technical και αποθηκεύεται σε διαφορετικό πίνακα.
4. Αποθηκεύεται η ημερομηνία γέννησης αντί για ηλικία του προσωπικού.
5. Τα μουσικά είδη (Genres) και τα υποείδη (sub-genres) είναι πολλά, δεν μπορούν να αναπαρασταθούν ως απαριθμήσεις (enums) ή συμβολοσειρές (strings) και επειδή τα πολλά υποείδη ανήκουν σε ένα είδος, τα μοντελοποιούμε ως ξεχωριστές οντότητες. Συγκεκριμένα τα υποείδη είναι Weak Entity.
6. Τα μουσικά είδη ενός καλλιτέχνη ή συγκροτήματος στο οποίο συμμετέχουν είναι ανεξάρτητα.
7. Αποθηκεύονται τα μουσικά υποείδη των καλλιτεχνών και συγκροτημάτων σε πίνακα, ενώ τα μουσικά τους είδη είναι εύκολα υπολογίσιμα από αυτήν την πληροφορία.
8. Μια παράσταση (Event) έχει προκαθορισμένη ημερομηνία/ώρα και διάρκεια.
9. Τα Συγκροτήματα (Bands) είναι ξεχωριστή οντότητα από τους καλλιτέχνες.
10. Στα ερωτήματα, με τον όρο καλλιτέχνες ερμηνεύουμε το σύνολο των artists & bands, ενώ δεν λαμβάνουμε υπόψιν γεγονός πως συχνά οι ίδιοι σολίστες είναι και μέλη συγκροτημάτων.
11. Οι πληροφορίες για την σκηνή (Stage) ενός εισιτηρίου δεν αποθηκεύονται στον πίνακα των εισιτηρίων αλλά μπορεί να βρεθεί έμμεσα βρίσκοντας το stage\_id του αντίστοιχου event του εισιτηρίου.
12. Οι ουρές μεταπώλησης (Resales) υλοποιούνται σε δύο οντότητες, μία με Αγοραστές Μεταπώλησης (Resales Buyers) που θέλουν να αγοράσουν εισιτήρια και μία με Πωλητές Μεταπώλησης (Resales Sellers) (οι προσφορές εισιτηρίων).
13. Οι μεταπωλήσεις έχουν συγκεκριμένη κατηγορία, ως foreign key στον πίνακα κατηγοριών (VIP/General/Backstage).
14. Οι Εικόνες (Images) είναι ξεχωριστή οντότητα, υποθέτουμε πως θα φυλάσσονται σε static server και επομένως στην βάση καταχωρούμε μόνο συνδέσμους προς αυτές.
15. Οι Επισκέπτες (Visitors) είτε βαθμολογούν όλες τις κατηγορίες (Απόδοση Καλλιτέχνη, Ήχος/Φωτισμός κ.λπ.) είτε καμία. Ως εκ τούτου, μια βαθμολογία περιλαμβάνει όλες τις κατηγορίες.
16. Εκτός από τους επισκέπτες, οι οντότητες καλλιτέχνης, συγκρότημα και προσωπικό έχουν όλες διευθύνσεις email.
17. Οντότητες που έχουν εικόνες: Φεστιβάλ, Καλλιτέχνες, Συγκροτήματα, Σκηνές, Εξοπλισμός Σκηνής, οι οποίες σχετίζονται μέσω junction tables.
18. Επιτρέπεται το πολύ μία αξιολόγηση (rating) ανά επισκέπτη και εμφάνιση (Εφαρμόστηκε έμμεσα ορίζοντας τα performance\_id, event\_id ως το primary key).

# Entity Relationship Diagram

Παρατίθεται το διάγραμμα ER που κατασκευάστηκε για τον σχεδιασμό της βάσης. Χρησιμοποιήθηκε το εργαλείο [draw.io (app.diagrams.net)](https://app.diagrams.net/). (Χάριν απλότητος δεν συμπεριλήφθηκε η οντότητα των εικόνων στο διάγραμμα)



# Relational Diagram

Παρατίθεται το σχεσιακό διάγραμμα που οπτικοποιεί την βάση μας. Χρησιμοποιήθηκε το εργαλείο [dbeaver](http://dbeaver.io).

# SQL DDL Script

Παρατίθενται τα περιεχόμενα του install.sql, στο οποίο ορίζονται όλοι οι πίνακες, τα ευρετήρια για την ταχύτερη απάντηση των ερωτημάτων του και ένα σωρό περιορισμοί υλοποιημένοι κατά κύριο λόγο είτε μέσω CHECK εντός των ορισμών των πινάκων, είτε μέσω συναρτήσεων Postgres και αντίστοιχων TRIGGER. Επιπλέον μέσω TRIGGER έχει υλοποιηθεί η αυτόματη αντιστοιχία αγοραστών και πωλητών μεταπωλήσεων, και η μεταφορά του εισιτηρίου στον νέο κάτοχο.

| -- Dependencies  CREATE EXTENSION IF NOT EXISTS btree\_gist;  -- Drop old triggers  DO $$ DECLARE  r RECORD; BEGIN  -- Loop through all triggers in the database  FOR r IN (SELECT trigger\_name, event\_object\_table  FROM information\_schema.triggers  WHERE trigger\_schema = 'public') -- Specify schema if needed (e.g., 'public')  LOOP  -- Drop each trigger dynamically  EXECUTE 'DROP TRIGGER IF EXISTS ' || r.trigger\_name || ' ON ' || r.event\_object\_table;  END LOOP; END $$;  -- Tables Declaration  -- Tables used as enums to restrict values  DROP TABLE IF EXISTS job CASCADE; CREATE TABLE job (  job\_id INT PRIMARY KEY,  name VARCHAR(50) NOT NULL );  DROP TABLE IF EXISTS payment\_method CASCADE; CREATE TABLE payment\_method (  payment\_method\_id INT PRIMARY KEY,  name VARCHAR(50) NOT NULL );  DROP TABLE IF EXISTS ticket\_category CASCADE; CREATE TABLE ticket\_category (  ticket\_category\_id INT PRIMARY KEY,  name VARCHAR(50) NOT NULL );  DROP TABLE IF EXISTS experience CASCADE; CREATE TABLE experience (  experience\_id INT PRIMARY KEY CHECK (experience\_id BETWEEN 1 AND 5),  name VARCHAR(50) NOT NULL );  DROP TABLE IF EXISTS performance\_type CASCADE; CREATE TABLE performance\_type (  performance\_type\_id INT PRIMARY KEY,  name VARCHAR(50) NOT NULL );  DROP TABLE IF EXISTS continent CASCADE; CREATE TABLE continent (  continent\_id INT PRIMARY KEY,  name VARCHAR(50) NOT NULL );  -- Data tables  DROP TABLE IF EXISTS location CASCADE; CREATE TABLE location (  location\_id SERIAL PRIMARY KEY,  address VARCHAR(255) NOT NULL,  longitude DOUBLE PRECISION NOT NULL,  latitude DOUBLE PRECISION NOT NULL,  city VARCHAR(100) NOT NULL,  country VARCHAR(100) NOT NULL,  continent\_id INT NOT NULL REFERENCES continent(continent\_id) ON DELETE RESTRICT );  DROP TABLE IF EXISTS stage CASCADE; CREATE TABLE stage (  stage\_id SERIAL PRIMARY KEY,  location\_id INT NOT NULL REFERENCES location(location\_id) ON DELETE CASCADE,  name VARCHAR(100) NOT NULL,  description TEXT,  capacity INT NOT NULL CHECK (capacity >= 0) );  DROP TABLE IF EXISTS festival CASCADE; CREATE TABLE festival (  festival\_id SERIAL PRIMARY KEY,  name VARCHAR(100) NOT NULL,  begin\_date DATE NOT NULL,  end\_date DATE NOT NULL,  location\_id INT NOT NULL REFERENCES location(location\_id) ON DELETE CASCADE,    festival\_year INT GENERATED ALWAYS AS (EXTRACT(YEAR FROM begin\_date)) STORED,  CONSTRAINT same\_begin\_end\_year CHECK (festival\_year = EXTRACT(YEAR FROM end\_date)),  CONSTRAINT unique\_festival\_per\_year UNIQUE (festival\_year) );  DROP TABLE IF EXISTS event CASCADE; CREATE TABLE event (  event\_id SERIAL PRIMARY KEY,  title VARCHAR(100) NOT NULL,  festival\_id INT NOT NULL REFERENCES festival(festival\_id) ON DELETE RESTRICT,  stage\_id INT NOT NULL REFERENCES stage(stage\_id) ON DELETE CASCADE,  date\_time TIMESTAMP NOT NULL,  duration INTERVAL NOT NULL,    CONSTRAINT stage\_single\_event\_per\_time EXCLUDE USING gist (  stage\_id WITH =,  tsrange(date\_time, date\_time + duration, '[]') WITH &&  ) );  DROP TABLE IF EXISTS stage\_equipment CASCADE; CREATE TABLE stage\_equipment (  stage\_id INT NOT NULL REFERENCES stage(stage\_id) ON DELETE CASCADE,  equipment VARCHAR(100) NOT NULL,  PRIMARY KEY (stage\_id, equipment) );  DROP TABLE IF EXISTS personnel CASCADE; CREATE TABLE personnel (  personnel\_id SERIAL PRIMARY KEY,  job\_id INT NOT NULL REFERENCES job(job\_id) ON DELETE RESTRICT,  first\_name VARCHAR(100) NOT NULL,  last\_name VARCHAR(100) NOT NULL,  birth\_date DATE NOT NULL,  experience\_id INT NOT NULL REFERENCES experience(experience\_id) ON DELETE RESTRICT,  email VARCHAR(100) NOT NULL UNIQUE CHECK (email ~'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,}$') );  DROP TABLE IF EXISTS event\_personnel CASCADE; CREATE TABLE event\_personnel (  event\_id INT NOT NULL REFERENCES event(event\_id) ON DELETE RESTRICT,  personnel\_id INT NOT NULL REFERENCES personnel(personnel\_id) ON DELETE CASCADE,  PRIMARY KEY (event\_id, personnel\_id) );  DROP TABLE IF EXISTS artist CASCADE; CREATE TABLE artist (  artist\_id SERIAL PRIMARY KEY,  first\_name VARCHAR(100) NOT NULL,  last\_name VARCHAR(100) NOT NULL,  nickname VARCHAR(100),  birth\_date DATE NOT NULL,  email VARCHAR(100) NOT NULL UNIQUE CHECK (email ~'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,}$'),  website VARCHAR(255),  instagram VARCHAR(255),   name VARCHAR(100) GENERATED ALWAYS AS (COALESCE(nickname, first\_name || ' ' || last\_name)) STORED );  DROP TABLE IF EXISTS genre CASCADE; CREATE TABLE genre (  genre\_id SERIAL PRIMARY KEY,  name VARCHAR(100) NOT NULL );  DROP TABLE IF EXISTS subgenre CASCADE; CREATE TABLE subgenre (  genre\_id INT NOT NULL REFERENCES genre(genre\_id) ON DELETE CASCADE,  name VARCHAR(100) NOT NULL,  PRIMARY KEY (genre\_id, name) );  DROP TABLE IF EXISTS artist\_subgenre CASCADE; CREATE TABLE artist\_subgenre (  artist\_id INT NOT NULL REFERENCES artist(artist\_id) ON DELETE CASCADE,  genre\_id INT NOT NULL,  subgenre\_name VARCHAR(100) NOT NULL,  PRIMARY KEY (artist\_id, genre\_id, subgenre\_name),  FOREIGN KEY (genre\_id, subgenre\_name) REFERENCES subgenre(genre\_id, name) ON DELETE CASCADE );  DROP TABLE IF EXISTS band CASCADE; CREATE TABLE band (  band\_id SERIAL PRIMARY KEY,  name VARCHAR(100) NOT NULL,  established DATE NOT NULL,  email VARCHAR(100) NOT NULL UNIQUE CHECK (email ~'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,}$'),  website VARCHAR(255),  instagram VARCHAR(255) );  DROP TABLE IF EXISTS band\_artist CASCADE; CREATE TABLE band\_artist (  band\_id INT NOT NULL REFERENCES band(band\_id) ON DELETE CASCADE,  artist\_id INT NOT NULL REFERENCES artist(artist\_id) ON DELETE CASCADE,  PRIMARY KEY (band\_id, artist\_id) );  DROP TABLE IF EXISTS band\_subgenre CASCADE; CREATE TABLE band\_subgenre (  band\_id INT NOT NULL REFERENCES band(band\_id) ON DELETE CASCADE,  genre\_id INT NOT NULL,  subgenre\_name VARCHAR(100) NOT NULL,  PRIMARY KEY (band\_id, genre\_id, subgenre\_name),  FOREIGN KEY (genre\_id, subgenre\_name) REFERENCES subgenre(genre\_id, name) ON DELETE CASCADE );  DROP TABLE IF EXISTS performance CASCADE; CREATE TABLE performance (  performance\_id SERIAL PRIMARY KEY,  event\_id INT NOT NULL REFERENCES event(event\_id) ON DELETE RESTRICT,  stage\_id INT NOT NULL REFERENCES stage(stage\_id) ON DELETE CASCADE,  date\_time TIMESTAMP NOT NULL,  duration INTERVAL NOT NULL CHECK (duration < '3 hours'),  performance\_type\_id INT NOT NULL REFERENCES performance\_type(performance\_type\_id) ON DELETE RESTRICT,  artist\_id INT REFERENCES artist(artist\_id) ON DELETE CASCADE,  band\_id INT REFERENCES band(band\_id) ON DELETE CASCADE CHECK (  (artist\_id IS NOT NULL AND band\_id IS NULL) OR  (artist\_id IS NULL AND band\_id IS NOT NULL)  ) );  DROP TABLE IF EXISTS visitor CASCADE; CREATE TABLE visitor (  visitor\_id SERIAL PRIMARY KEY,  first\_name VARCHAR(100) NOT NULL,  last\_name VARCHAR(100) NOT NULL,  phone\_number VARCHAR(50),  email VARCHAR(100) NOT NULL UNIQUE CHECK (email ~'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,}$'),  birth\_date DATE NOT NULL );  DROP TABLE IF EXISTS ticket CASCADE; CREATE TABLE ticket (  visitor\_id INT NOT NULL REFERENCES visitor(visitor\_id) ON DELETE CASCADE,  event\_id INT NOT NULL REFERENCES event(event\_id) ON DELETE RESTRICT,  ticket\_category\_id INT NOT NULL REFERENCES ticket\_category(ticket\_category\_id) ON DELETE RESTRICT,  purchase\_date\_time TIMESTAMP NOT NULL,  price DECIMAL(10,2) NOT NULL,  payment\_method\_id INT NOT NULL REFERENCES payment\_method(payment\_method\_id) ON DELETE RESTRICT,  barcode VARCHAR(20) NOT NULL,  used BOOLEAN NOT NULL DEFAULT FALSE,  PRIMARY KEY (visitor\_id, event\_id) );  DROP TABLE IF EXISTS resales\_buyer CASCADE; CREATE TABLE resales\_buyer (  visitor\_id INT NOT NULL REFERENCES visitor(visitor\_id) ON DELETE CASCADE,  event\_id INT NOT NULL REFERENCES event(event\_id) ON DELETE RESTRICT,  date\_time TIMESTAMP NOT NULL,  ticket\_category\_id INT NOT NULL REFERENCES ticket\_category(ticket\_category\_id) ON DELETE RESTRICT,  PRIMARY KEY (visitor\_id, event\_id) );  DROP TABLE IF EXISTS resales\_seller CASCADE; CREATE TABLE resales\_seller (  visitor\_id INT NOT NULL REFERENCES visitor(visitor\_id) ON DELETE CASCADE,  event\_id INT NOT NULL REFERENCES event(event\_id) ON DELETE RESTRICT,  date\_time TIMESTAMP NOT NULL,  PRIMARY KEY (visitor\_id, event\_id),  FOREIGN KEY (visitor\_id, event\_id) REFERENCES ticket(visitor\_id, event\_id) ON DELETE CASCADE );  DROP TABLE IF EXISTS rating CASCADE; CREATE TABLE rating (  visitor\_id INT NOT NULL REFERENCES visitor(visitor\_id) ON DELETE CASCADE,  performance\_id INT NOT NULL REFERENCES performance(performance\_id) ON DELETE CASCADE,  artist\_performance SMALLINT NOT NULL CHECK (artist\_performance BETWEEN 1 AND 5),  sound\_lighting SMALLINT NOT NULL CHECK (sound\_lighting BETWEEN 1 AND 5),  stage\_presence SMALLINT NOT NULL CHECK (stage\_presence BETWEEN 1 AND 5),  organization SMALLINT NOT NULL CHECK (organization BETWEEN 1 AND 5),  overall\_impression SMALLINT NOT NULL CHECK (overall\_impression BETWEEN 1 AND 5),  PRIMARY KEY (visitor\_id, performance\_id) );  DROP TABLE IF EXISTS images CASCADE; CREATE TABLE images (  image\_id SERIAL PRIMARY KEY,  link VARCHAR(255) NOT NULL,  description TEXT );  DROP TABLE IF EXISTS festival\_image CASCADE; CREATE TABLE festival\_image (  festival\_id INT NOT NULL REFERENCES festival(festival\_id) ON DELETE RESTRICT,  image\_id INT NOT NULL REFERENCES images(image\_id) ON DELETE CASCADE,  PRIMARY KEY (festival\_id, image\_id) );  DROP TABLE IF EXISTS artist\_image CASCADE; CREATE TABLE artist\_image (  artist\_id INT NOT NULL REFERENCES artist(artist\_id) ON DELETE CASCADE,  image\_id INT NOT NULL REFERENCES images(image\_id) ON DELETE CASCADE,  PRIMARY KEY (artist\_id, image\_id) );  DROP TABLE IF EXISTS band\_image CASCADE; CREATE TABLE band\_image (  band\_id INT NOT NULL REFERENCES band(band\_id) ON DELETE CASCADE,  image\_id INT NOT NULL REFERENCES images(image\_id) ON DELETE CASCADE,  PRIMARY KEY (band\_id, image\_id) );  DROP TABLE IF EXISTS stage\_image CASCADE; CREATE TABLE stage\_image (  stage\_id INT NOT NULL REFERENCES stage(stage\_id) ON DELETE CASCADE,  image\_id INT NOT NULL REFERENCES images(image\_id) ON DELETE CASCADE,  PRIMARY KEY (stage\_id, image\_id) );  DROP TABLE IF EXISTS stage\_equipment\_image CASCADE; CREATE TABLE stage\_equipment\_image (  stage\_id INT NOT NULL REFERENCES stage(stage\_id) ON DELETE CASCADE,  equipment VARCHAR(100) NOT NULL,  image\_id INT NOT NULL REFERENCES images(image\_id) ON DELETE CASCADE,  PRIMARY KEY (stage\_id, equipment, image\_id),  FOREIGN KEY (stage\_id, equipment) REFERENCES stage\_equipment(stage\_id, equipment) ON DELETE CASCADE );   -- Indexes  -- Many indexes are omitted due to being impliciτ (primary keys) -- Also some queries already benefit from indexes for previous queries  -- Speedup resales triggers  DROP INDEX IF EXISTS idx\_resales\_buyer\_event\_category\_date; CREATE INDEX idx\_resales\_buyer\_event\_category\_date ON resales\_buyer(event\_id, ticket\_category\_id, date\_time);  DROP INDEX IF EXISTS idx\_resales\_seller\_event\_date; CREATE INDEX idx\_resales\_seller\_event\_date ON resales\_seller(event\_id, date\_time);  DROP INDEX IF EXISTS idx\_ticket\_event\_category\_used; CREATE INDEX idx\_ticket\_event\_category\_used ON ticket(event\_id, ticket\_category\_id, used);  -- Q01 CREATE INDEX idx\_ticket\_event\_id\_payment\_method\_id ON ticket(event\_id, payment\_method\_id) CREATE INDEX idx\_event\_festival\_id ON event(festival\_id); CREATE INDEX idx\_festival\_year ON festival(festival\_year);  -- Q02 CREATE INDEX idx\_performance\_year\_date\_time ON performance (EXTRACT(YEAR FROM date\_time)); CREATE INDEX idx\_performance\_artist\_id ON performance(artist\_id); CREATE INDEX idx\_performance\_band\_id ON performance(band\_id); CREATE INDEX idx\_performance\_event\_id ON performance(event\_id); CREATE INDEX idx\_artist\_subgenre\_genre\_id ON artist\_subgenre(genre\_id); CREATE INDEX idx\_band\_subgenre\_genre\_id ON band\_subgenre(genre\_id); CREATE INDEX idx\_genre\_name ON genre(name);  -- Q03 CREATE INDEX idx\_performance\_type\_event\_artist\_band ON performance(performance\_type\_id, event\_id, artist\_id, band\_id);  -- Q04 CREATE INDEX idx\_rating\_performance ON rating(performance\_id);  -- Q05 CREATE INDEX idx\_artist\_birth\_date ON artist(birth\_date); CREATE INDEX idx\_band\_established ON band(established);  -- Q07 CREATE INDEX idx\_job\_name ON job(name); CREATE INDEX idx\_personnel\_job\_id ON personnel(job\_id);  -- Q08 CREATE INDEX idx\_event\_date\_time ON event (date\_time);  -- Q09 CREATE INDEX idx\_ticket\_used ON ticket (used, event\_id, visitor\_id);  -- Q  -- Triggers for constraint enforcement  -- (Some of the following constraints are simple, yet putting them as table checks could lead to incosistency, -- as checks are only meant to check values of the new/updated row, not access other rows/tables.)  -- In order to defer constraints/checks: -- BEGIN; -- SET CONSTRAINTS ALL DEFERRED -- ALTER TABLE my\_table DISABLE TRIGGER my\_trigger; -- ... -- ...statements -- ... -- ALTER TABLE my\_table ENABLE TRIGGER my\_trigger; -- END   CREATE OR REPLACE FUNCTION event\_during\_festival() RETURNS TRIGGER AS $$ DECLARE  festival\_row festival; BEGIN  SELECT INTO festival\_row \* FROM festival WHERE festival\_id = NEW.festival\_id;  IF NEW.date\_time < festival\_row.begin\_date OR  (NEW.date\_time + NEW.duration) > (festival\_row.end\_date + INTERVAL '1 day') THEN  RAISE EXCEPTION 'Event dates must fall within festival % (% to %)',  NEW.festival\_id, festival\_row.begin\_date, festival\_row.end\_date;  END IF;  RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_event\_during\_festival BEFORE INSERT OR UPDATE ON event FOR EACH ROW EXECUTE FUNCTION event\_during\_festival();  CREATE OR REPLACE FUNCTION personnel\_requirements() RETURNS TRIGGER AS $$ DECLARE  stage\_capacity INT;  security\_count INT;  assistant\_count INT;  violation\_text TEXT; BEGIN  SELECT string\_agg(  format('Event %s: Security (%s/%s) | Assistants (%s/%s)',  e.event\_id,  pc.security\_count, ceil(st.capacity \* 0.05),  pc.assistant\_count, ceil(st.capacity \* 0.02)),  E'\n'  )  INTO violation\_text  FROM (  SELECT  event\_id,  stage\_id,  COUNT(CASE WHEN job.name = 'Security' THEN 1 END) AS security\_count,  COUNT(CASE WHEN job.name = 'Assistant' THEN 1 END) AS assistant\_count  FROM events e  LEFT JOIN event\_personnel USING (event\_id)  LEFT JOIN personnel USING (personnel\_id)  LEFT JOIN job USING (job\_id)  GROUP BY event\_id, stage\_id  )  JOIN stage USING (stage\_id)  WHERE security\_count < 0.05 \* capacity OR assistant\_count < 0.02 \* st.capacity;    IF violation\_text IS NOT NULL THEN  RAISE EXCEPTION 'Personnel requirements not met:%\n%s',  E'\n-----------------------------',  violation\_text  USING HINT = 'Add Security and/or Assistant personnel to these events';  END IF;   IF TG\_OP = 'DELETE' THEN  RETURN OLD;  ELSE  RETURN NEW;  END IF; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_personnel\_requirements\_on\_personnel AFTER UPDATE OR DELETE ON event\_personnel FOR EACH STATEMENT EXECUTE FUNCTION personnel\_requirements();  CREATE TRIGGER trigger\_personnel\_requirements\_on\_event AFTER INSERT OR UPDATE ON event FOR EACH STATEMENT EXECUTE FUNCTION personnel\_requirements();  CREATE OR REPLACE FUNCTION performance\_duration\_breaks() RETURNS TRIGGER AS $$ DECLARE -- event\_begin TIMESTAMP; -- event\_end TIMESTAMP; -- prev\_end TIMESTAMP; -- next\_start TIMESTAMP; -- gap\_before INTERVAL; -- gap\_after INTERVAL; BEGIN  -- 1. Check if performance is within the festival's dates  IF EXISTS (  SELECT 1  FROM event e  WHERE  e.event\_id = NEW.event\_id AND  (NEW.date\_time < e.date\_time OR NEW.date\_time + NEW.duration > event\_end))  THEN RAISE EXCEPTION 'Performance not within event duration'  END IF;   -- 2. Check breaks between performances  IF EXISTS (  WITH t AS (  SELECT date\_time, (date\_time + duration) AS end\_time  FROM performance  WHERE event\_id = NEW.event\_id AND performance\_id IS DISTINCT FROM NEW.performance\_id  ORDER BY date\_time)  SELECT 1  FROM (  SELECT (NEW.date\_time - MAX(end\_time)) AS break  FROM t  WHERE date\_time < NEW.date\_time  UNION ALL  SELECT (MIN(date\_time) - (NEW.date\_time + NEW.duration)) AS break  FROM t  WHERE date\_time > NEW.date\_time)  WHERE break < '5 minutes' OR break > '30 minutes'  ) THEN RAISE EXCEPTION 'Performance break before or after is out of bounds (5-30 min)'  END IF;   RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_performance\_duration\_breaks BEFORE INSERT OR UPDATE ON performance FOR EACH ROW EXECUTE FUNCTION performance\_duration\_breaks();  -- CREATE OR REPLACE FUNCTION prevent\_delete() -- RETURNS TRIGGER AS $$ -- BEGIN -- RAISE EXCEPTION 'Deletion (Cancellation) of events or festivals is not allowed'; -- END; -- $$ LANGUAGE plpgsql;  -- CREATE TRIGGER trigger\_prevent\_delete\_festival -- BEFORE DELETE ON festival FOR EACH STATEMENT EXECUTE FUNCTION prevent\_delete();  -- CREATE TRIGGER trigger\_prevent\_delete\_event -- BEFORE DELETE ON event FOR EACH STATEMENT EXECUTE FUNCTION prevent\_delete();  CREATE OR REPLACE FUNCTION check\_artist\_overlap() RETURNS TRIGGER AS $$ BEGIN  IF EXISTS (  WITH involved\_artists AS (  SELECT NEW.artist\_id AS artist\_id  WHERE NEW.artist\_id IS NOT NULL  UNION ALL  SELECT artist\_id  FROM band\_artist  WHERE band\_id = NEW.band\_id  )  SELECT 1  FROM performance p  WHERE (  p.artist\_id IN (SELECT artist\_id FROM involved\_artists)  OR EXISTS (  SELECT 1  FROM band\_artist ba  WHERE ba.band\_id = p.band\_id  AND ba.artist\_id IN (SELECT artist\_id FROM involved\_artists)  )  )  AND (p.date\_time, p.date\_time + p.duration)  OVERLAPS (NEW.date\_time, NEW.date\_time + NEW.duration)  AND p.performance\_id IS DISTINCT FROM NEW.performance\_id  ) THEN  RAISE EXCEPTION 'Artist(s) have overlapping performances';  END IF;   RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_check\_artist\_overlap BEFORE INSERT OR UPDATE ON performance FOR EACH ROW EXECUTE FUNCTION check\_artist\_overlap();  CREATE OR REPLACE FUNCTION limit\_consecutive\_artist\_festivals() RETURNS TRIGGER AS $$ BEGIN   IF EXISTS (  WITH involved\_artists AS (  SELECT NEW.artist\_id AS artist\_id  WHERE NEW.artist\_id IS NOT NULL  UNION ALL  SELECT artist\_id FROM band\_artist  WHERE band\_id = NEW.band\_id  ), all\_years AS (  SELECT DISTINCT  ia.artist\_id,  EXTRACT(YEAR FROM p.date\_time)::INT AS performance\_year  FROM involved\_artists ia  LEFT JOIN performance p ON p.artist\_id = ia.artist\_id  OR p.band\_id IN (SELECT band\_id FROM band\_artist WHERE artist\_id = ia.artist\_id)  UNION  SELECT DISTINCT  ia.artist\_id,  EXTRACT(YEAR FROM NEW.date\_time)::INT AS performance\_year  FROM involved\_artists ia  ), consecutive\_check AS (  -- Check for 4+ consecutive years using window functions  SELECT artist\_id, performance\_year - ROW\_NUMBER() OVER (  PARTITION BY artist\_id ORDER BY performance\_year  ) AS consecutive\_group\_id  FROM all\_years  )  SELECT 1  FROM consecutive\_check  GROUP BY artist\_id, consecutive\_group\_id  HAVING COUNT(\*) >= 4  ) THEN  RAISE EXCEPTION 'Artist(s) cannot perform for more than 3 consecutive years';  END IF;  RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_limit\_consecutive\_artist\_festivals BEFORE INSERT OR UPDATE ON performance FOR EACH ROW EXECUTE FUNCTION limit\_consecutive\_artist\_festivals();  CREATE OR REPLACE FUNCTION limit\_ticket\_count\_on\_insert() RETURNS TRIGGER AS $$ DECLARE  max\_capacity INT;  tickets\_sold INT;  vip\_sold INT;  exception\_message TEXT; BEGIN  SELECT s.capacity INTO max\_capacity  FROM event e  JOIN stage s ON e.stage\_id = s.stage\_id  WHERE e.event\_id = NEW.event\_id;   SELECT COUNT(\*), SUM(CASE WHEN ticket\_category.name = 'VIP' THEN 1 ELSE 0 END)  INTO tickets\_sold, vip\_sold  FROM ticket  JOIN ticket\_category USING (ticket\_category\_id)  WHERE event\_id = NEW.event\_id;   IF tickets\_sold >= max\_capacity THEN  exception\_message := format('Ticket count (%s) will exceed stage capacity (%s).', tickets\_sold, max\_capacity);  RAISE EXCEPTION '%', exception\_message;  END IF;  IF vip\_sold >= 0.1 \* max\_capacity THEN  exception\_message := format('VIP Ticket count (%s) will exceed 10%% of stage capacity (%s).', vip\_sold, max\_capacity);  RAISE EXCEPTION '%', exception\_message;  END IF;   RETURN NEW; END; $$ LANGUAGE plpgsql;   CREATE TRIGGER trigger\_limit\_ticket\_count\_on\_insert BEFORE INSERT ON ticket FOR EACH ROW EXECUTE FUNCTION limit\_ticket\_count\_on\_insert();  CREATE OR REPLACE FUNCTION add\_resales\_buyer() RETURNS TRIGGER AS $$ DECLARE  seller\_id INT; BEGIN  -- Find a matching seller offer:  SELECT rs.visitor\_id  INTO seller\_id  FROM resales\_seller rs  JOIN ticket t ON t.visitor\_id = rs.visitor\_id AND t.event\_id = rs.event\_id  WHERE rs.event\_id = NEW.event\_id  AND t.ticket\_category\_id = NEW.ticket\_category\_id  AND t.used = FALSE -- only consider unused tickets  ORDER BY rs.date\_time ASC  LIMIT 1;    IF FOUND THEN  -- Transfer the ticket: change the ticket owner from the seller to the new buyer.  UPDATE ticket  SET visitor\_id = NEW.visitor\_id  WHERE visitor\_id = seller\_id  AND event\_id = NEW.event\_id;    -- Remove the seller's resale offer.  DELETE FROM resales\_seller  WHERE visitor\_id = seller\_id  AND event\_id = NEW.event\_id;    -- Skip inserting the buyer row since the resale is now complete.  RETURN NULL;  ELSE  -- No matching seller found: proceed with inserting the buyer offer.  RETURN NEW;  END IF; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_add\_resales\_buyer BEFORE INSERT ON resales\_buyer FOR EACH ROW EXECUTE FUNCTION add\_resales\_buyer();  CREATE OR REPLACE FUNCTION add\_resales\_seller() RETURNS TRIGGER AS $$ DECLARE  seller\_ticket\_category\_id; INT  buyer\_id INT; -- Specify data type  ticket\_used BOOLEAN; BEGIN  SELECT t.ticket\_category\_id, t.used  INTO seller\_ticket\_category\_id, ticket\_used  FROM ticket t  WHERE t.visitor\_id = NEW.visitor\_id  AND t.event\_id = NEW.event\_id;    IF ticket\_used THEN  RAISE EXCEPTION 'Cannot offer resale: Ticket is already used.';  END IF;    -- Look for a matching buyer offer based on event and category.  SELECT rb.visitor\_id  INTO buyer\_id  FROM resales\_buyer rb  WHERE rb.event\_id = NEW.event\_id  AND rb.ticket\_category\_id = seller\_ticket\_category\_id  ORDER BY rb.date\_time ASC  LIMIT 1;    IF FOUND THEN  -- Transfer ticket ownership from seller to buyer.  UPDATE ticket  SET visitor\_id = buyer\_id  WHERE visitor\_id = NEW.visitor\_id  AND event\_id = NEW.event\_id;    -- Remove the buyer offer since the resale is complete.  DELETE FROM resales\_buyer  WHERE visitor\_id = buyer\_id  AND event\_id = NEW.event\_id;    -- Cancel the insertion of the seller row.  RETURN NULL;  ELSE  -- No matching buyer: allow the seller offer to be inserted.  RETURN NEW;  END IF; END; $$ LANGUAGE plpgsql;  -- Fix trigger syntax CREATE TRIGGER trigger\_add\_resales\_seller BEFORE INSERT ON resales\_seller FOR EACH ROW EXECUTE FUNCTION add\_resales\_seller(); |
| --- |

# 

# Χρήση Ευρετηρίων και Στρατηγικών Join

#### Ερώτημα 4.

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

**Q04.sql**

| **WITH vars AS (SELECT  42 AS target\_artist\_id) SELECT  ANY\_VALUE(artist.name) AS name,  ROUND(AVG(artist\_performance)::numeric, 2) AS average\_artist\_performance,  ROUND(AVG(overall\_impression)::numeric, 2) AS average\_overal\_impression FROM performance JOIN artist USING (artist\_id) JOIN rating USING (performance\_id) JOIN vars ON artist\_id = target\_artist\_id GROUP BY artist\_id;  SELECT 'Sequential Scans + Nested Loop Join' AS "Combination 1"; SET enable\_seqscan = on; SET enable\_indexscan = off; SET enable\_nestloop = on; SET enable\_hashjoin = off; SET enable\_mergejoin = off;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Sequential Scans + Hash Join' AS "Combination 2"; SET enable\_nestloop = off; SET enable\_hashjoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Sequential Scans + Merge Join' AS "Combination 3"; SET enable\_hashjoin = off; SET enable\_mergejoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Force Index Scans + Nested Loop Join' AS "Combination 4"; SET enable\_seqscan = off; SET enable\_indexscan = on; SET enable\_nestloop = on; SET enable\_mergejoin = off;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Force Index Scans + Hash Join' AS "Combination 5"; SET enable\_nestloop = off; SET enable\_hashjoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Force Index Scans + Merge Join' AS "Combination 6";SET enable\_hashjoin = off; SET enable\_mergejoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  -- Reset planner settings to default after testing RESET enable\_seqscan; RESET enable\_indexscan; RESET enable\_nestloop; RESET enable\_hashjoin; RESET enable\_mergejoin;** |
| --- |

**Q04\_out.txt**

name | average\_artist\_performance | average\_overal\_impression

----------------+----------------------------+---------------------------

christiecooper | 3.07 | 3.10

(1 row)

Combination 1

-------------------------------------

Sequential Scans + Nested Loop Join

(1 row)

SET

SET

SET

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=16.84..1415.17 rows=1 width=100) (actual time=0.511..0.515 rows=1 loops=1)

-> Nested Loop (cost=16.84..1412.15 rows=401 width=20) (actual time=0.136..0.389 rows=500 loops=1)

-> Nested Loop (cost=8.57..16.60 rows=1 width=20) (actual time=0.065..0.070 rows=1 loops=1)

-> Bitmap Heap Scan on performance (cost=4.29..8.30 rows=1 width=8) (actual time=0.039..0.041 rows=1 loops=1)

Recheck Cond: (artist\_id = 42)

Heap Blocks: exact=1

-> Bitmap Index Scan on idx\_performance\_artist\_id (cost=0.00..4.29 rows=1 width=0) (actual time=0.025..0.027 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Bitmap Heap Scan on artist (cost=4.28..8.30 rows=1 width=16) (actual time=0.018..0.019 rows=1 loops=1)

Recheck Cond: (artist\_id = 42)

Heap Blocks: exact=1

-> Bitmap Index Scan on artist\_pkey (cost=0.00..4.28 rows=1 width=0) (actual time=0.013..0.013 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Bitmap Heap Scan on rating (cost=8.28..1390.57 rows=497 width=8) (actual time=0.068..0.178 rows=500 loops=1)

Recheck Cond: (performance.performance\_id = performance\_id)

Heap Blocks: exact=4

-> Bitmap Index Scan on idx\_rating\_performance (cost=0.00..8.15 rows=497 width=0) (actual time=0.049..0.049 rows=500 loops=1)

Index Cond: (performance\_id = performance.performance\_id)

Planning Time: 0.525 ms

Execution Time: 0.722 ms

(20 rows)

Combination 2

------------------------------

Sequential Scans + Hash Join

(1 row)

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000012.59..10000011325.20 rows=1 width=100) (actual time=75.416..75.420 rows=1 loops=1)

-> Nested Loop (cost=10000000012.59..10000011322.17 rows=401 width=20) (actual time=54.541..75.377 rows=500 loops=1)

-> Hash Join (cost=8.31..11308.86 rows=401 width=8) (actual time=54.517..75.250 rows=500 loops=1)

Hash Cond: (rating.performance\_id = performance.performance\_id)

-> Seq Scan on rating (cost=0.00..9735.00 rows=594700 width=8) (actual time=0.027..32.056 rows=594700 loops=1)

-> Hash (cost=8.30..8.30 rows=1 width=8) (actual time=0.031..0.032 rows=1 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

-> Bitmap Heap Scan on performance (cost=4.29..8.30 rows=1 width=8) (actual time=0.019..0.020 rows=1 loops=1)

Recheck Cond: (artist\_id = 42)

Heap Blocks: exact=1

-> Bitmap Index Scan on idx\_performance\_artist\_id (cost=0.00..4.29 rows=1 width=0) (actual time=0.011..0.011 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Materialize (cost=4.28..8.30 rows=1 width=16) (actual time=0.000..0.000 rows=1 loops=500)

-> Bitmap Heap Scan on artist (cost=4.28..8.30 rows=1 width=16) (actual time=0.015..0.015 rows=1 loops=1)

Recheck Cond: (artist\_id = 42)

Heap Blocks: exact=1

-> Bitmap Index Scan on artist\_pkey (cost=0.00..4.28 rows=1 width=0) (actual time=0.007..0.007 rows=1 loops=1)

Index Cond: (artist\_id = 42)

Planning Time: 0.379 ms

Execution Time: 75.968 ms

(20 rows)

Combination 3

-------------------------------

Sequential Scans + Merge Join

(1 row)

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000066788.70..10000069769.25 rows=1 width=100) (actual time=146.687..146.689 rows=1 loops=1)

-> Merge Join (cost=10000066788.70..10000069766.22 rows=401 width=20) (actual time=146.594..146.654 rows=500 loops=1)

Merge Cond: (performance.performance\_id = rating.performance\_id)

-> Sort (cost=10000000016.61..10000000016.62 rows=1 width=20) (actual time=0.015..0.017 rows=1 loops=1)

Sort Key: performance.performance\_id

Sort Method: quicksort Memory: 25kB

-> Nested Loop (cost=10000000008.57..10000000016.60 rows=1 width=20) (actual time=0.012..0.013 rows=1 loops=1)

-> Bitmap Heap Scan on performance (cost=4.29..8.30 rows=1 width=8) (actual time=0.006..0.007 rows=1 loops=1)

Recheck Cond: (artist\_id = 42)

Heap Blocks: exact=1

-> Bitmap Index Scan on idx\_performance\_artist\_id (cost=0.00..4.29 rows=1 width=0) (actual time=0.004..0.004 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Bitmap Heap Scan on artist (cost=4.28..8.30 rows=1 width=16) (actual time=0.003..0.004 rows=1 loops=1)

Recheck Cond: (artist\_id = 42)

Heap Blocks: exact=1

-> Bitmap Index Scan on artist\_pkey (cost=0.00..4.28 rows=1 width=0) (actual time=0.002..0.002 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Sort (cost=66772.09..68258.84 rows=594700 width=8) (actual time=114.449..127.044 rows=395629 loops=1)

Sort Key: rating.performance\_id

Sort Method: quicksort Memory: 38515kB

-> Seq Scan on rating (cost=0.00..9735.00 rows=594700 width=8) (actual time=0.003..54.029 rows=594700 loops=1)

Planning Time: 0.196 ms

Execution Time: 148.008 ms

(23 rows)

Combination 4

--------------------------------------

Force Index Scans + Nested Loop Join

(1 row)

SET

SET

SET

SET

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=0.98..44.72 rows=1 width=100) (actual time=0.155..0.156 rows=1 loops=1)

-> Nested Loop (cost=0.98..41.69 rows=401 width=20) (actual time=0.030..0.121 rows=500 loops=1)

-> Nested Loop (cost=0.55..16.60 rows=1 width=20) (actual time=0.020..0.021 rows=1 loops=1)

-> Index Scan using idx\_performance\_artist\_id on performance (cost=0.28..8.29 rows=1 width=8) (actual time=0.015..0.015 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Index Scan using artist\_pkey on artist (cost=0.28..8.29 rows=1 width=16) (actual time=0.004..0.004 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Index Scan using idx\_rating\_performance on rating (cost=0.42..20.12 rows=497 width=8) (actual time=0.009..0.057 rows=500 loops=1)

Index Cond: (performance\_id = performance.performance\_id)

Planning Time: 0.138 ms

Execution Time: 0.178 ms

(11 rows)

Combination 5

-------------------------------

Force Index Scans + Hash Join

(1 row)

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000009.01..10000016706.12 rows=1 width=100) (actual time=93.095..93.097 rows=1 loops=1)

-> Nested Loop (cost=10000000009.01..10000016703.09 rows=401 width=20) (actual time=61.670..93.055 rows=500 loops=1)

-> Hash Join (cost=8.73..16689.78 rows=401 width=8) (actual time=61.655..92.942 rows=500 loops=1)

Hash Cond: (rating.performance\_id = performance.performance\_id)

-> Index Scan using idx\_rating\_performance on rating (cost=0.42..15115.92 rows=594700 width=8) (actual time=0.010..59.676 rows=594700 loops=1)

-> Hash (cost=8.29..8.29 rows=1 width=8) (actual time=0.008..0.009 rows=1 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

-> Index Scan using idx\_performance\_artist\_id on performance (cost=0.28..8.29 rows=1 width=8) (actual time=0.006..0.006 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Materialize (cost=0.28..8.30 rows=1 width=16) (actual time=0.000..0.000 rows=1 loops=500)

-> Index Scan using artist\_pkey on artist (cost=0.28..8.29 rows=1 width=16) (actual time=0.008..0.008 rows=1 loops=1)

Index Cond: (artist\_id = 42)

Planning Time: 0.113 ms

Execution Time: 93.121 ms

(14 rows)

Combination 6

--------------------------------

Force Index Scans + Merge Join

(1 row)

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000000.98..10000016685.25 rows=1 width=100) (actual time=60.417..60.418 rows=1 loops=1)

-> Merge Join (cost=10000000000.98..10000016682.22 rows=401 width=20) (actual time=60.174..60.378 rows=500 loops=1)

Merge Cond: (performance.performance\_id = rating.performance\_id)

-> Nested Loop (cost=10000000000.55..10000000075.53 rows=1 width=20) (actual time=0.135..0.232 rows=1 loops=1)

-> Index Scan using performance\_pkey on performance (cost=0.28..67.23 rows=1 width=8) (actual time=0.130..0.225 rows=1 loops=1)

Filter: (artist\_id = 42)

Rows Removed by Filter: 1482

-> Index Scan using artist\_pkey on artist (cost=0.28..8.29 rows=1 width=16) (actual time=0.004..0.005 rows=1 loops=1)

Index Cond: (artist\_id = 42)

-> Index Scan using idx\_rating\_performance on rating (cost=0.42..15115.92 rows=594700 width=8) (actual time=0.004..39.813 rows=395629 loops=1)

Planning Time: 0.222 ms

Execution Time: 60.444 ms

(12 rows)

RESET

RESET

RESET

RESET

RESET

#### 

#### Ερώτημα 6.

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

**Q06.sql**

| **WITH vars AS (  SELECT 1234 AS target\_visitor\_id ) SELECT  event\_id,  event.title AS event\_title,  (AVG(artist\_performance)  + AVG(sound\_lighting)  + AVG(stage\_presence)  + AVG(organization)  + AVG(overall\_impression)) / 5 AS average\_event\_rating FROM visitor JOIN vars ON visitor\_id = target\_visitor\_id JOIN ticket USING (visitor\_id) JOIN performance USING (event\_id) JOIN event USING (event\_id) LEFT JOIN rating USING (visitor\_id, performance\_id) WHERE ticket.used = TRUE GROUP BY (event\_id, event.title);  SELECT 'Sequential Scans + Nested Loop Join' AS "Combination 1"; SET enable\_seqscan = on; SET enable\_indexscan = off; SET enable\_nestloop = on; SET enable\_hashjoin = off; SET enable\_mergejoin = off;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Sequential Scans + Hash Join' AS "Combination 2"; SET enable\_nestloop = off; SET enable\_hashjoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Sequential Scans + Merge Join' AS "Combination 3"; SET enable\_hashjoin = off; SET enable\_mergejoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Force Index Scans + Nested Loop Join' AS "Combination 4"; SET enable\_seqscan = off; SET enable\_indexscan = on; SET enable\_nestloop = on; SET enable\_mergejoin = off;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Force Index Scans + Hash Join' AS "Combination 5"; SET enable\_nestloop = off; SET enable\_hashjoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  SELECT 'Force Index Scans + Merge Join' AS "Combination 6";SET enable\_hashjoin = off; SET enable\_mergejoin = on;  EXPLAIN ANALYZE -- ...the query repeated...  -- Reset planner settings to default after testing RESET enable\_seqscan; RESET enable\_indexscan; RESET enable\_nestloop; RESET enable\_hashjoin; RESET enable\_mergejoin;** |
| --- |

**Q06\_out.txt**

event\_id | event\_title | average\_event\_rating

----------+-------------------------------------+------------------------

229 | Whirl Mysterious Rumbles 2021! |

297 | Clap Intense Frequencies 2022! |

321 | Spin Heavy Reflections 2022! |

105 | Flicker Epic Flashes 2019! |

386 | Echo Epic Chords 2024! |

450 | Groove Fiery Rumbles 2024! | 4.2000000000000000

424 | Jam Surreal Basses 2024! | 2.80000000000000000000

86 | Sync Incredible Reflections 2019! |

84 | Bounce Epic Tones 2019! |

237 | Shake Epic Spark 2021! | 3.80000000000000000000

223 | Clap Majestic Phases 2021! | 2.60000000000000000000

352 | Spin Blazing Strums 2023! |

219 | Beat Boisterous Tunes 2021! |

320 | Buzz Fiery Ripples 2022! |

348 | Harmonize Dynamic Chimes 2023! | 2.60000000000000000000

399 | Strum Burning Grooves 2024! |

66 | Vibrate Sizzling Pulse 2018! | 4.0000000000000000

112 | Beat Majestic Motions 2019! | 2.20000000000000000000

59 | Strum Mystic Grooves 2018! |

356 | Rush Adventurous Chords 2023! |

290 | Groove Wild Lights 2022! |

460 | Shake Intoxicating Sounds 2024! |

289 | Vibe Burning Explosions 2022! |

135 | Buzz Cloudy Notes 2019! |

300 | Spin Raucous Chimes 2022! | 2.60000000000000000000

265 | Strum Free Colors 2021! |

2 | Vibrate Fierce Ripples 2017! | 3.3000000000000000

124 | Whirl Wild Flares 2019! |

227 | Riff Dizzying Waves 2021! |

372 | Rock Intoxicating Tunes 2023! | 2.20000000000000000000

339 | Flow Groovy Basses 2023! |

261 | Drop Free Rays 2021! |

344 | Surge Bright Ripples 2023! |

385 | Sync Neon Explosions 2024! |

99 | Spin Dynamic Spectra 2019! |

367 | Groove Sonic Phases 2023! |

235 | Buzz Fierce Basses 2021! |

335 | Spin Magnetic Breezes 2023! | 3.0000000000000000

439 | Jammin Loud Thunder 2024! |

53 | Flicker Fiery Notes 2018! | 4.0000000000000000

260 | Rave Sonic Grooves 2021! | 3.6000000000000000

412 | Shout Loud Storms 2024! | 2.8000000000000000

359 | Rush Epic Rays 2023! | 3.4000000000000000

139 | Twist Neon Flares 2019! |

226 | Groove Surreal Beats 2021! |

293 | Rave Funky Echoes 2022! |

77 | Bop Rebellious Storms 2019! |

232 | Riff Epic Colors 2021! | 3.0000000000000000

21 | Chime Blazing Shakes 2017! |

316 | Strum Explosive Vibes 2022! |

324 | Jammin Adventurous Explosions 2022! |

177 | Harmonize Cloudy Beats 2020! |

256 | Vibrate Intoxicating Melodies 2021! |

310 | Strum Funky Shakes 2022! |

5 | Vibe Neon Storms 2017! |

264 | Rush Neon Sounds 2021! |

353 | Buzz Blazing Rumbles 2023! |

(57 rows)

Combination 1

-------------------------------------

Sequential Scans + Nested Loop Join

(1 row)

SET

SET

SET

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=855.28..859.65 rows=50 width=68) (actual time=1.088..1.199 rows=57 loops=1)

Group Key: ticket.event\_id

-> Sort (cost=855.28..855.60 rows=131 width=40) (actual time=1.072..1.082 rows=148 loops=1)

Sort Key: ticket.event\_id

Sort Method: quicksort Memory: 32kB

-> Nested Loop Left Join (cost=14.77..850.67 rows=131 width=40) (actual time=0.106..1.009 rows=148 loops=1)

Join Filter: (performance.performance\_id = rating.performance\_id)

Rows Removed by Join Filter: 2493

-> Nested Loop (cost=10.22..753.56 rows=131 width=38) (actual time=0.061..0.495 rows=148 loops=1)

Join Filter: (ticket.event\_id = performance.event\_id)

-> Nested Loop (cost=9.88..524.23 rows=52 width=38) (actual time=0.053..0.318 rows=57 loops=1)

-> Nested Loop (cost=9.28..292.61 rows=52 width=8) (actual time=0.042..0.149 rows=57 loops=1)

-> Bitmap Heap Scan on visitor (cost=4.30..8.31 rows=1 width=4) (actual time=0.011..0.012 rows=1 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=1

-> Bitmap Index Scan on visitor\_pkey (cost=0.00..4.30 rows=1 width=0) (actual time=0.005..0.005 rows=1 loops=1)

Index Cond: (visitor\_id = 1234)

-> Bitmap Heap Scan on ticket (cost=4.98..283.78 rows=52 width=8) (actual time=0.028..0.128 rows=57 loops=1)

Recheck Cond: (visitor\_id = 1234)

Filter: used

Rows Removed by Filter: 20

Heap Blocks: exact=77

-> Bitmap Index Scan on ticket\_pkey (cost=0.00..4.97 rows=72 width=0) (actual time=0.014..0.015 rows=77 loops=1)

Index Cond: (visitor\_id = 1234)

-> Memoize (cost=0.60..4.61 rows=1 width=30) (actual time=0.003..0.003 rows=1 loops=57)

Cache Key: ticket.event\_id

Cache Mode: logical

Hits: 0 Misses: 57 Evictions: 0 Overflows: 0 Memory Usage: 8kB

-> Bitmap Heap Scan on event (cost=0.59..4.60 rows=1 width=30) (actual time=0.002..0.002 rows=1 loops=57)

Recheck Cond: (ticket.event\_id = event\_id)

Heap Blocks: exact=57

-> Bitmap Index Scan on event\_pkey (cost=0.00..0.59 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=57)

Index Cond: (event\_id = ticket.event\_id)

-> Bitmap Heap Scan on performance (cost=0.34..4.37 rows=3 width=8) (actual time=0.002..0.002 rows=3 loops=57)

Recheck Cond: (event\_id = event.event\_id)

Heap Blocks: exact=58

-> Bitmap Index Scan on idx\_performance\_event\_id (cost=0.00..0.33 rows=3 width=0) (actual time=0.001..0.001 rows=3 loops=57)

Index Cond: (event\_id = event.event\_id)

-> Materialize (cost=4.55..65.71 rows=16 width=18) (actual time=0.000..0.001 rows=17 loops=148)

-> Bitmap Heap Scan on rating (cost=4.55..65.63 rows=16 width=18) (actual time=0.019..0.042 rows=18 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=18

-> Bitmap Index Scan on rating\_pkey (cost=0.00..4.54 rows=16 width=0) (actual time=0.015..0.015 rows=18 loops=1)

Index Cond: (visitor\_id = 1234)

Planning Time: 0.587 ms

Execution Time: 1.326 ms

(46 rows)

Combination 2

------------------------------

Sequential Scans + Hash Join

(1 row)

SET

SET

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000418.31..10000000422.68 rows=50 width=68) (actual time=0.988..1.077 rows=57 loops=1)

Group Key: ticket.event\_id

-> Sort (cost=10000000418.31..10000000418.64 rows=131 width=40) (actual time=0.971..0.982 rows=148 loops=1)

Sort Key: ticket.event\_id

Sort Method: quicksort Memory: 32kB

-> Hash Left Join (cost=10000000374.60..10000000413.70 rows=131 width=40) (actual time=0.545..0.933 rows=148 loops=1)

Hash Cond: (performance.performance\_id = rating.performance\_id)

-> Hash Join (cost=10000000308.78..10000000347.53 rows=131 width=38) (actual time=0.445..0.805 rows=148 loops=1)

Hash Cond: (ticket.event\_id = event.event\_id)

-> Nested Loop (cost=10000000288.73..10000000327.14 rows=131 width=16) (actual time=0.221..0.544 rows=148 loops=1)

-> Hash Join (cost=284.43..317.19 rows=131 width=16) (actual time=0.203..0.477 rows=148 loops=1)

Hash Cond: (performance.event\_id = ticket.event\_id)

-> Seq Scan on performance (cost=0.00..28.83 rows=1483 width=8) (actual time=0.008..0.121 rows=1483 loops=1)

-> Hash (cost=283.78..283.78 rows=52 width=8) (actual time=0.187..0.188 rows=57 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 11kB

-> Bitmap Heap Scan on ticket (cost=4.98..283.78 rows=52 width=8) (actual time=0.034..0.173 rows=57 loops=1)

Recheck Cond: (visitor\_id = 1234)

Filter: used

Rows Removed by Filter: 20

Heap Blocks: exact=77

-> Bitmap Index Scan on ticket\_pkey (cost=0.00..4.97 rows=72 width=0) (actual time=0.017..0.017 rows=77 loops=1)

Index Cond: (visitor\_id = 1234)

-> Materialize (cost=4.30..8.32 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=148)

-> Bitmap Heap Scan on visitor (cost=4.30..8.31 rows=1 width=4) (actual time=0.009..0.010 rows=1 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=1

-> Bitmap Index Scan on visitor\_pkey (cost=0.00..4.30 rows=1 width=0) (actual time=0.006..0.006 rows=1 loops=1)

Index Cond: (visitor\_id = 1234)

-> Hash (cost=12.80..12.80 rows=580 width=30) (actual time=0.216..0.216 rows=580 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 45kB

-> Seq Scan on event (cost=0.00..12.80 rows=580 width=30) (actual time=0.021..0.103 rows=580 loops=1)

-> Hash (cost=65.63..65.63 rows=16 width=18) (actual time=0.073..0.074 rows=18 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

-> Bitmap Heap Scan on rating (cost=4.55..65.63 rows=16 width=18) (actual time=0.023..0.061 rows=18 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=18

-> Bitmap Index Scan on rating\_pkey (cost=0.00..4.54 rows=16 width=0) (actual time=0.014..0.014 rows=18 loops=1)

Index Cond: (visitor\_id = 1234)

Planning Time: 0.734 ms

Execution Time: 1.203 ms

(40 rows)

Combination 3

-------------------------------

Sequential Scans + Merge Join

(1 row)

SET

SET

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000606.48..10000000624.04 rows=50 width=68) (actual time=1.683..2.015 rows=57 loops=1)

Group Key: ticket.event\_id

-> Merge Join (cost=10000000606.48..10000000620.00 rows=131 width=40) (actual time=1.656..1.918 rows=148 loops=1)

Merge Cond: (ticket.event\_id = event.event\_id)

-> Merge Join (cost=10000000567.06..10000000576.04 rows=131 width=18) (actual time=1.431..1.612 rows=148 loops=1)

Merge Cond: (performance.event\_id = ticket.event\_id)

-> Sort (cost=10000000281.80..10000000285.51 rows=1483 width=18) (actual time=1.211..1.274 rows=1197 loops=1)

Sort Key: performance.event\_id

Sort Method: quicksort Memory: 84kB

-> Merge Left Join (cost=10000000196.03..10000000203.69 rows=1483 width=18) (actual time=0.597..0.920 rows=1483 loops=1)

Merge Cond: (performance.performance\_id = rating.performance\_id)

-> Sort (cost=10000000130.08..10000000133.79 rows=1483 width=12) (actual time=0.540..0.623 rows=1483 loops=1)

Sort Key: performance.performance\_id

Sort Method: quicksort Memory: 95kB

-> Nested Loop (cost=10000000004.30..10000000051.97 rows=1483 width=12) (actual time=0.048..0.353 rows=1483 loops=1)

-> Bitmap Heap Scan on visitor (cost=4.30..8.31 rows=1 width=4) (actual time=0.035..0.037 rows=1 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=1

-> Bitmap Index Scan on visitor\_pkey (cost=0.00..4.30 rows=1 width=0) (actual time=0.010..0.011 rows=1 loops=1)

Index Cond: (visitor\_id = 1234)

-> Seq Scan on performance (cost=0.00..28.83 rows=1483 width=8) (actual time=0.007..0.113 rows=1483 loops=1)

-> Sort (cost=65.95..65.99 rows=16 width=18) (actual time=0.052..0.054 rows=18 loops=1)

Sort Key: rating.performance\_id

Sort Method: quicksort Memory: 25kB

-> Bitmap Heap Scan on rating (cost=4.55..65.63 rows=16 width=18) (actual time=0.019..0.045 rows=18 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=18

-> Bitmap Index Scan on rating\_pkey (cost=0.00..4.54 rows=16 width=0) (actual time=0.011..0.011 rows=18 loops=1)

Index Cond: (visitor\_id = 1234)

-> Sort (cost=285.26..285.39 rows=52 width=8) (actual time=0.211..0.214 rows=57 loops=1)

Sort Key: ticket.event\_id

Sort Method: quicksort Memory: 26kB

-> Bitmap Heap Scan on ticket (cost=4.98..283.78 rows=52 width=8) (actual time=0.044..0.198 rows=57 loops=1)

Recheck Cond: (visitor\_id = 1234)

Filter: used

Rows Removed by Filter: 20

Heap Blocks: exact=77

-> Bitmap Index Scan on ticket\_pkey (cost=0.00..4.97 rows=72 width=0) (actual time=0.025..0.026 rows=77 loops=1)

Index Cond: (visitor\_id = 1234)

-> Sort (cost=39.42..40.87 rows=580 width=30) (actual time=0.220..0.241 rows=460 loops=1)

Sort Key: event.event\_id

Sort Method: quicksort Memory: 53kB

-> Seq Scan on event (cost=0.00..12.80 rows=580 width=30) (actual time=0.013..0.103 rows=580 loops=1)

Planning Time: 1.113 ms

Execution Time: 2.112 ms

(45 rows)

Combination 4

--------------------------------------

Force Index Scans + Nested Loop Join

(1 row)

SET

SET

SET

SET

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------------------------

HashAggregate (cost=472.20..473.95 rows=50 width=68) (actual time=1.380..1.428 rows=57 loops=1)

Group Key: ticket.event\_id

Batches: 1 Memory Usage: 88kB

-> Nested Loop Left Join (cost=10.38..469.91 rows=131 width=40) (actual time=0.097..1.282 rows=148 loops=1)

Join Filter: (performance.performance\_id = rating.performance\_id)

Rows Removed by Join Filter: 2493

-> Nested Loop (cost=5.84..372.80 rows=131 width=38) (actual time=0.075..0.567 rows=148 loops=1)

Join Filter: (ticket.event\_id = performance.event\_id)

-> Nested Loop (cost=5.56..346.88 rows=52 width=38) (actual time=0.068..0.388 rows=57 loops=1)

-> Nested Loop (cost=5.27..288.60 rows=52 width=8) (actual time=0.051..0.171 rows=57 loops=1)

-> Index Only Scan using visitor\_pkey on visitor (cost=0.29..4.31 rows=1 width=4) (actual time=0.014..0.015 rows=1 loops=1)

Index Cond: (visitor\_id = 1234)

Heap Fetches: 0

-> Bitmap Heap Scan on ticket (cost=4.98..283.78 rows=52 width=8) (actual time=0.035..0.146 rows=57 loops=1)

Recheck Cond: (visitor\_id = 1234)

Filter: used

Rows Removed by Filter: 20

Heap Blocks: exact=77

-> Bitmap Index Scan on ticket\_pkey (cost=0.00..4.97 rows=72 width=0) (actual time=0.014..0.014 rows=77 loops=1)

Index Cond: (visitor\_id = 1234)

-> Memoize (cost=0.29..1.15 rows=1 width=30) (actual time=0.003..0.003 rows=1 loops=57)

Cache Key: ticket.event\_id

Cache Mode: logical

Hits: 0 Misses: 57 Evictions: 0 Overflows: 0 Memory Usage: 8kB

-> Index Scan using event\_pkey on event (cost=0.28..1.14 rows=1 width=30) (actual time=0.003..0.003 rows=1 loops=57)

Index Cond: (event\_id = ticket.event\_id)

-> Index Scan using idx\_performance\_event\_id on performance (cost=0.28..0.46 rows=3 width=8) (actual time=0.002..0.002 rows=3 loops=57)

Index Cond: (event\_id = event.event\_id)

-> Materialize (cost=4.55..65.71 rows=16 width=18) (actual time=0.000..0.002 rows=17 loops=148)

-> Bitmap Heap Scan on rating (cost=4.55..65.63 rows=16 width=18) (actual time=0.017..0.044 rows=18 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=18

-> Bitmap Index Scan on rating\_pkey (cost=0.00..4.54 rows=16 width=0) (actual time=0.011..0.011 rows=18 loops=1)

Index Cond: (visitor\_id = 1234)

Planning Time: 0.826 ms

Execution Time: 1.542 ms

(36 rows)

Combination 5

-------------------------------

Force Index Scans + Hash Join

(1 row)

SET

SET

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000467.18..10000000471.55 rows=50 width=68) (actual time=2.449..2.631 rows=57 loops=1)

Group Key: ticket.event\_id

-> Sort (cost=10000000467.18..10000000467.50 rows=131 width=40) (actual time=2.405..2.454 rows=148 loops=1)

Sort Key: ticket.event\_id

Sort Method: quicksort Memory: 32kB

-> Hash Left Join (cost=10000000393.05..10000000462.57 rows=131 width=40) (actual time=1.077..2.353 rows=148 loops=1)

Hash Cond: (performance.performance\_id = rating.performance\_id)

-> Hash Join (cost=10000000327.22..10000000396.40 rows=131 width=38) (actual time=0.957..2.163 rows=148 loops=1)

Hash Cond: (ticket.event\_id = event.event\_id)

-> Nested Loop (cost=10000000284.99..10000000353.83 rows=131 width=16) (actual time=0.489..1.603 rows=148 loops=1)

-> Hash Join (cost=284.70..347.88 rows=131 width=16) (actual time=0.454..1.461 rows=148 loops=1)

Hash Cond: (performance.event\_id = ticket.event\_id)

-> Index Scan using idx\_performance\_event\_id on performance (cost=0.28..59.52 rows=1483 width=8) (actual time=0.041..0.653 rows=1483 loops=1)

-> Hash (cost=283.78..283.78 rows=52 width=8) (actual time=0.366..0.368 rows=57 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 11kB

-> Bitmap Heap Scan on ticket (cost=4.98..283.78 rows=52 width=8) (actual time=0.110..0.346 rows=57 loops=1)

Recheck Cond: (visitor\_id = 1234)

Filter: used

Rows Removed by Filter: 20

Heap Blocks: exact=77

-> Bitmap Index Scan on ticket\_pkey (cost=0.00..4.97 rows=72 width=0) (actual time=0.046..0.046 rows=77 loops=1)

Index Cond: (visitor\_id = 1234)

-> Materialize (cost=0.29..4.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=148)

-> Index Only Scan using visitor\_pkey on visitor (cost=0.29..4.31 rows=1 width=4) (actual time=0.022..0.024 rows=1 loops=1)

Index Cond: (visitor\_id = 1234)

Heap Fetches: 0

-> Hash (cost=34.97..34.97 rows=580 width=30) (actual time=0.451..0.452 rows=580 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 45kB

-> Index Scan using event\_pkey on event (cost=0.28..34.97 rows=580 width=30) (actual time=0.058..0.325 rows=580 loops=1)

-> Hash (cost=65.63..65.63 rows=16 width=18) (actual time=0.104..0.106 rows=18 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

-> Bitmap Heap Scan on rating (cost=4.55..65.63 rows=16 width=18) (actual time=0.040..0.095 rows=18 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=18

-> Bitmap Index Scan on rating\_pkey (cost=0.00..4.54 rows=16 width=0) (actual time=0.025..0.025 rows=18 loops=1)

Index Cond: (visitor\_id = 1234)

Planning Time: 0.942 ms

Execution Time: 2.813 ms

(38 rows)

Combination 6

--------------------------------

Force Index Scans + Merge Join

(1 row)

SET

SET

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

GroupAggregate (cost=10000000519.91..10000000570.73 rows=50 width=68) (actual time=1.536..1.862 rows=57 loops=1)

Group Key: ticket.event\_id

-> Merge Join (cost=10000000519.91..10000000566.69 rows=131 width=40) (actual time=1.522..1.780 rows=148 loops=1)

Merge Cond: (ticket.event\_id = event.event\_id)

-> Merge Join (cost=10000000519.64..10000000528.62 rows=131 width=18) (actual time=1.513..1.662 rows=148 loops=1)

Merge Cond: (performance.event\_id = ticket.event\_id)

-> Sort (cost=10000000234.38..10000000238.09 rows=1483 width=18) (actual time=1.409..1.459 rows=1197 loops=1)

Sort Key: performance.event\_id

Sort Method: quicksort Memory: 84kB

-> Merge Left Join (cost=10000000066.52..10000000156.27 rows=1483 width=18) (actual time=0.102..1.151 rows=1483 loops=1)

Merge Cond: (performance.performance\_id = rating.performance\_id)

-> Nested Loop (cost=10000000000.57..10000000086.37 rows=1483 width=12) (actual time=0.040..0.878 rows=1483 loops=1)

-> Index Scan using performance\_pkey on performance (cost=0.28..63.52 rows=1483 width=8) (actual time=0.026..0.320 rows=1483 loops=1)

-> Materialize (cost=0.29..4.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=1483)

-> Index Only Scan using visitor\_pkey on visitor (cost=0.29..4.31 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)

Index Cond: (visitor\_id = 1234)

Heap Fetches: 0

-> Sort (cost=65.95..65.99 rows=16 width=18) (actual time=0.058..0.060 rows=18 loops=1)

Sort Key: rating.performance\_id

Sort Method: quicksort Memory: 25kB

-> Bitmap Heap Scan on rating (cost=4.55..65.63 rows=16 width=18) (actual time=0.021..0.049 rows=18 loops=1)

Recheck Cond: (visitor\_id = 1234)

Heap Blocks: exact=18

-> Bitmap Index Scan on rating\_pkey (cost=0.00..4.54 rows=16 width=0) (actual time=0.011..0.011 rows=18 loops=1)

Index Cond: (visitor\_id = 1234)

-> Sort (cost=285.26..285.39 rows=52 width=8) (actual time=0.101..0.105 rows=57 loops=1)

Sort Key: ticket.event\_id

Sort Method: quicksort Memory: 26kB

-> Bitmap Heap Scan on ticket (cost=4.98..283.78 rows=52 width=8) (actual time=0.021..0.094 rows=57 loops=1)

Recheck Cond: (visitor\_id = 1234)

Filter: used

Rows Removed by Filter: 20

Heap Blocks: exact=77

-> Bitmap Index Scan on ticket\_pkey (cost=0.00..4.97 rows=72 width=0) (actual time=0.010..0.010 rows=77 loops=1)

Index Cond: (visitor\_id = 1234)

-> Index Scan using event\_pkey on event (cost=0.28..34.97 rows=580 width=30) (actual time=0.008..0.068 rows=460 loops=1)

Planning Time: 1.430 ms

Execution Time: 1.939 ms

(38 rows)

RESET

RESET

RESET

RESET

RESET

#### Σύνοψη Αποτελεσμάτων

Ο χρόνος εκτέλεσης με τις αλλαγές:

|  | **Query 4** | | **Query 6** | |
| --- | --- | --- | --- | --- |
| **Sequential Scan** | **Index Scan** | **Sequential Scan** | **Index Scan** |
| **Nested Loop Join** | 0.722 ms | 0.178 ms | 1.326 ms | 1.542 ms |
| **Hash Join** | 75.968 ms | 93.121 ms | 1.203 ms | 2.813 ms |
| **Sort Merge Join** | 148.008 ms | 60.444 ms | 2.112 ms | 1.939 ms |

#### 