Βάσεις Δεδομένων

Αναφορά Εξαμηνιαίας Εργασίας

Music Festival - Pulse University

Φαίδων-Κορνέλις Κουρουνάκης (el20035)

Περιεχόμενα

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, 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 XX
  - **`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.

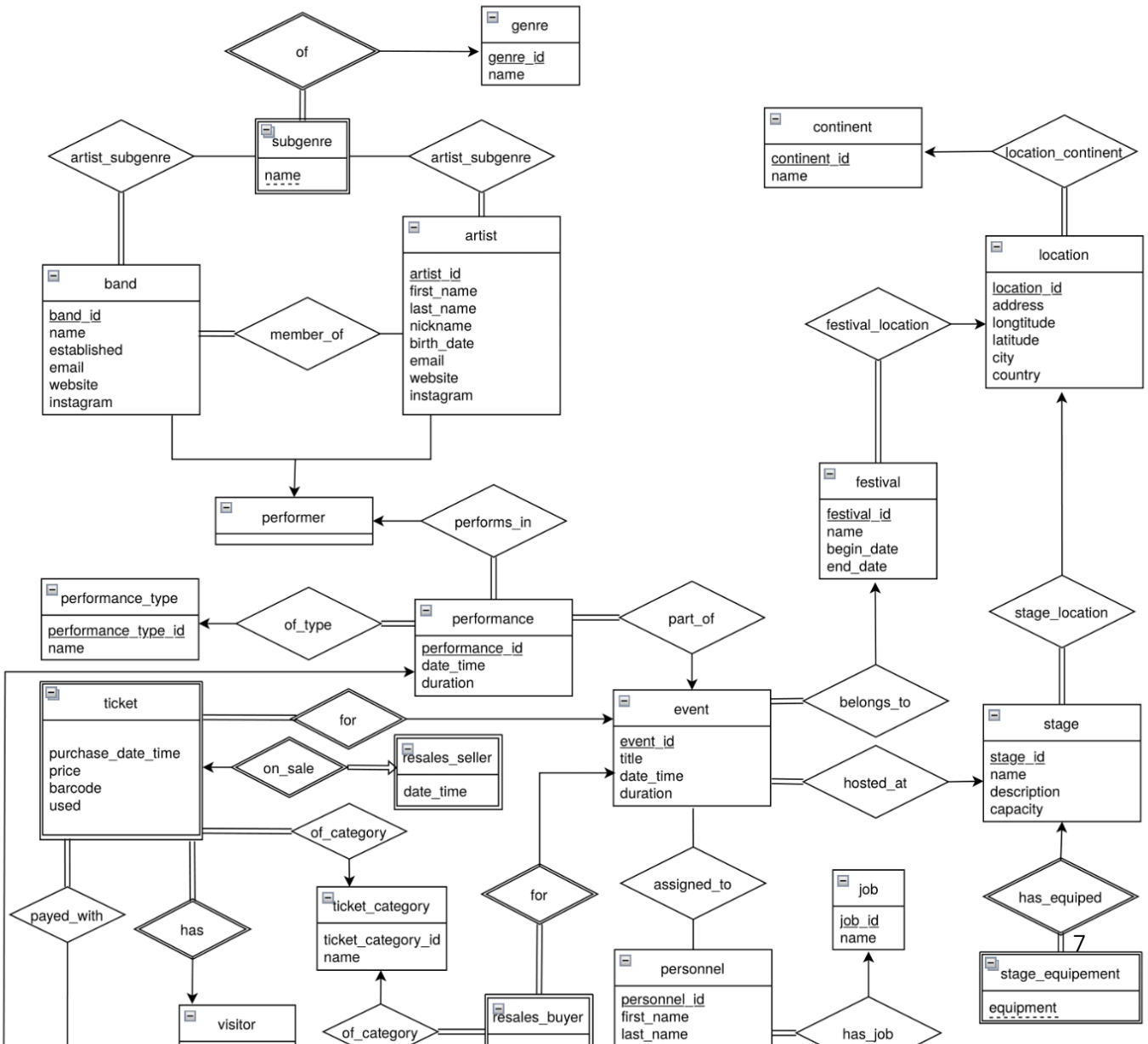
16. Εκτός από τους επισκέπτες, οι οντότητες καλλιτέχνης, συγκρότημα και προσωπικό έχουν όλες διευθύνσεις email.

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).

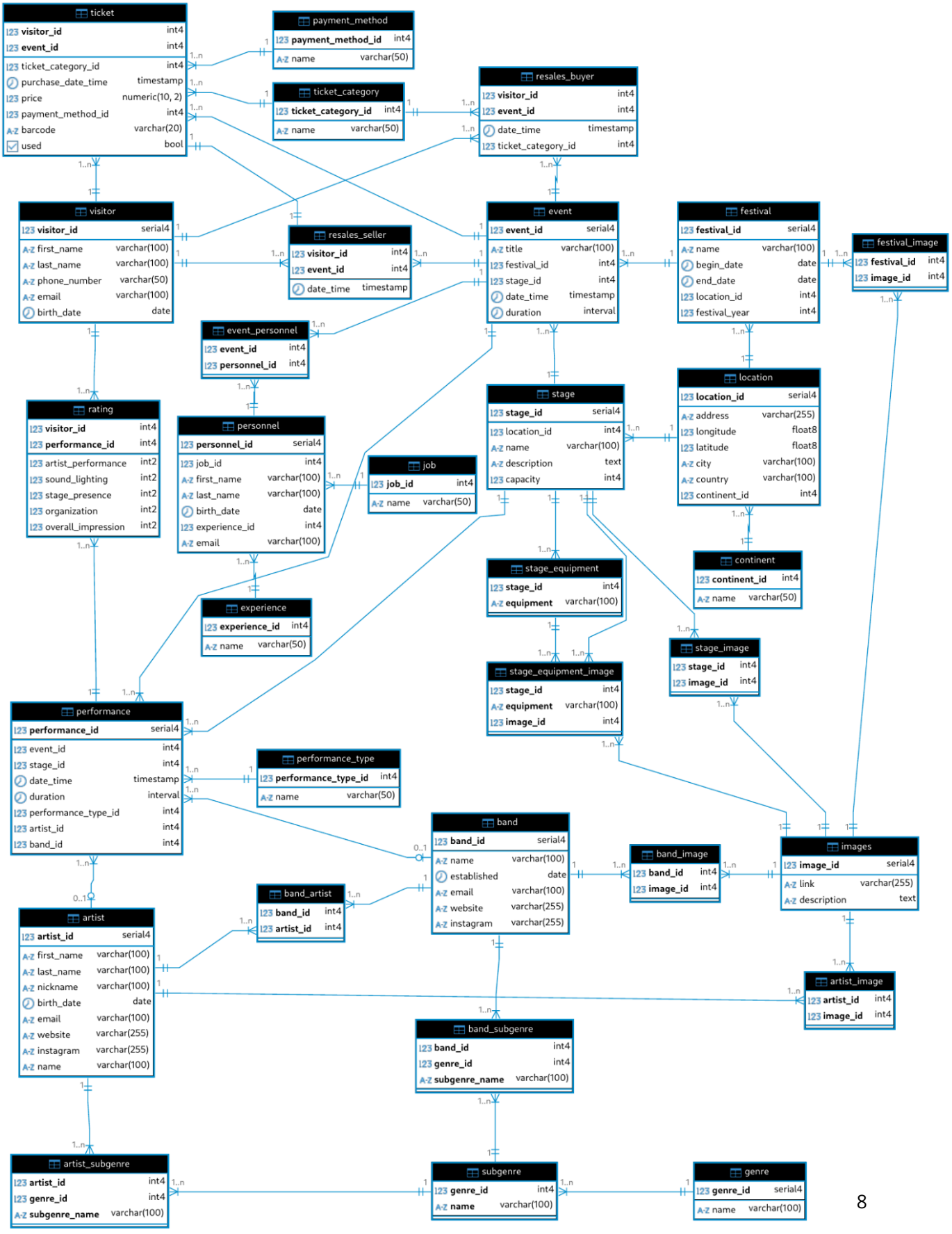
Entity Relationship Diagram

Παρατίθεται το διάγραμμα ER που κατασκευάστηκε για τον σχεδιασμό της βάσης. Χρησιμοποιήθηκε το εργαλείο draw.io (app.diagrams.net). (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.



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 implicit (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
```

```sql
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
```

```sql
-- ...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

```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!          | 2.8000000000000000000
          86 | Sync Incredible Reflections 2019!   |
          84 | Bounce Epic Tones 2019!           |
          237 | Shake Epic Spark 2021!            | 3.8000000000000000000
          223 | Clap Majestic Phases 2021!        | 2.6000000000000000000
          352 | Spin Blazing Strums 2023!         |
          219 | Beat Boisterous Tunes 2021!       |
          320 | Buzz Fiery Ripples 2022!          |
          348 | Harmonize Dynamic Chimes 2023!    | 2.6000000000000000000
          399 | Strum Burning Grooves 2024!       |
          66 | Vibrate Sizzling Pulse 2018!       |         4.0000000000000000
          112 | Beat Majestic Motions 2019!       | 2.2000000000000000000
          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!         | 2.6000000000000000000
          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!     | 2.2000000000000000000
          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