Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Βάσεις Δεδομένων (2024–2025)

Pulse University Database

https://github.com/ntua-el21026/DB25_137.git

Στοιχεία Ομάδας - DB137

- **Πέππας Μιχαήλ-Αθανάσιος**, 03121026
- **Τσουβαλάς Ανδρέας**, 03121022
- **Μάντζαρης Κωνσταντίνος**, 03122406

Στοιχεία Μαθήματος

- **Διδάσκοντες**: Δ. Τσουμάκος, Μ. Κόνιαρης
- **Εργαστήριο**: Συστημάτων Βάσεων Γνώσεων και Δεδομένων
- **Εξάμηνο**: Εαρινό 2025

Αθήνα, Μάιος 2025

Περιεχόμενα

1 Εισαγωγή

Το έγγραφο αυτό περιγράφει τον σχεδιασμό, την υλοποίηση και τις λειτουργίες της βάσης δεδομένων και του συστήματος **Pulse University**, το οποίο αναπτύχθηκε στο πλαίσιο της εξαμηνιαίας εργασίας για το μάθημα *Βάσεις Δεδομένων*, κατά το ακαδημαϊκό έτος 2024–2025.

Στόχος του έργου είναι η δημιουργία μιας πλήρους και ρεαλιστικής πληροφοριακής υποδομής για τη διαχείριση μουσικών φεστιβάλ, καλλιτεχνών, διοργανώσεων, επισκεπτών και στατιστικών αξιολογήσεων. Το σύστημα υποστηρίζει πολλαπλά φεστιβάλ ανά έτος και ήπειρο, διαχείριση χωρητικότητας, προγραμματισμό εμφανίσεων και δυναμικό μηχανισμό εισιτηρίων με αγοραπωλησίες, ουρές, και αξιολογήσεις.

Η ανάπτυξη καλύπτει:

- Τον σχεδιασμό και την υλοποίηση ενός σχεσιακού σχήματος σε MySQL, με αυστηρούς περιορισμούς ακεραιότητας και triggers για την επιβολή επιχειρησιακών κανόνων.
- Την ανάπτυξη εφαρμογής γραμμής εντολών (`CLI`) για την πλήρη διαχείριση του σχήματος, των χρηστών, και των ερωτημάτων.
- Την κατασκευή μιας σύγχρονης frontend SPA εφαρμογής (React + Tailwind), που επιτρέπει τη διεπαφή με τη βάση μέσω γραφικών καρτελών, επεξεργαστή SQL, και εργαλείων CLI.
- Την αυτοματοποιημένη παραγωγή συνθετικών δεδομένων μέσω Python scripts που σέβονται όλα τα triggers και τους περιορισμούς της βάσης.
- Την αξιολόγηση της απόδοσης και της κάλυψης ερωτημάτων μέσω indexing, views, και σχετικών metrics.

Η προσέγγιση που ακολουθήθηκε βασίζεται στην επεκτασιμότητα, την αναπαραγωγιμότητα και την αυστηρή διασφάλιση ακεραιότητας δεδομένων, με στόχο τη δημιουργία ενός συστήματος που μπορεί να δοκιμαστεί, να επεκταθεί και να χρησιμοποιηθεί παραγωγικά σε οποιδήποτε περιβάλλον.

1.1 Δομή του Project

Η δομή του project ακολουθεί μία καθαρή και κατανοητή οργάνωση, βασισμένη στη διάκριση λειτουργικών ενοτήτων του συστήματος. Κάθε φάκελος εξυπηρετεί συγκεκριμένο σκοπό, ενώ διατηρείται συνοχή μεταξύ τεκμηρίωσης, υλοποίησης και δοκιμών.
Ριζική Δομή Φακέλων:

```
.
├── cli/              # Διεπαφή γραμμής εντολών (CLI) – διαχείριση DB και χρηστών
│   ├── db137.py       # Κύριο CLI script με τις εντολές
│   └── users/         # Υποσύστημα διαχείρισης χρηστών/endpoint
├── code/          # Βοηθητικά scripts για παραγωγή δεδομένων, οργάνωση, και εργαλεία κώδικα
│   ├── data_generation/   # Συνθετικά δεδομένα (faker.py, faker_sql.py)
│   ├── code_utils/     # Εργαλεία για queries και consistency checks (qgen.py κ.ά.)
│   └── organization/      # Ανάλυση/δομή αρχείων έργου (struct.py)
├── diagrams/          # ER και relational διαγράμματα (PDF)
├── docs/              # Αναφορά, εκφώνηση και εσωτερική τεκμηρίωση
├── frontend/       # Περιβάλλον web (React/Vite) με tabs: Schema, Browse, Query, CLI
│   ├── public/            # Δημόσια assets (π.χ. logo)
│   ├── src/
│   │   ├── api/           # Διαχείριση επικοινωνίας με Flask backend
│   │   ├── auth/          # Ενότητες authentication (login, session storage)
│   │   ├── components/    # Επαναχρησιμοποιήσιμα στοιχεία UI (π.χ. session timer)
│   │   ├── hooks/         # Custom React hooks (π.χ. useSchema)
│   │   ├── pages/         # Δομή σελίδων (π.χ. LoginPage, MainPage)
│   │   ├── tabs/       # Περιεχόμενο των καρτελών του UI (Schema, CLI, Query, Browse)
│   │   ├── App.jsx        # Root component
│   │   ├── main.jsx       # Entry point εφαρμογής
│   │   └── index.css      # Tailwind styling
│   ├── index.html         # HTML πρότυπο της εφαρμογής
│   ├── tailwind.config.js # Ρυθμίσεις Tailwind
│   └── vite.config.js     # Ρυθμίσεις Vite
├── sql/             # SQL scripts (install, triggers, indexes, views, procedures)
```

```
│   └── queries/       # Υλοποιήσεις Q01–Q15 και αποτελέσματα (QXX_out.txt, plans)
├── test/               # Scripts αυτόματων δοκιμών CLI (test_cli.sh)
└── .envrc              # Περιβάλλον χρήστη (DB credentials, path setup)
```

Επεξήγηση βασικών φακέλων:

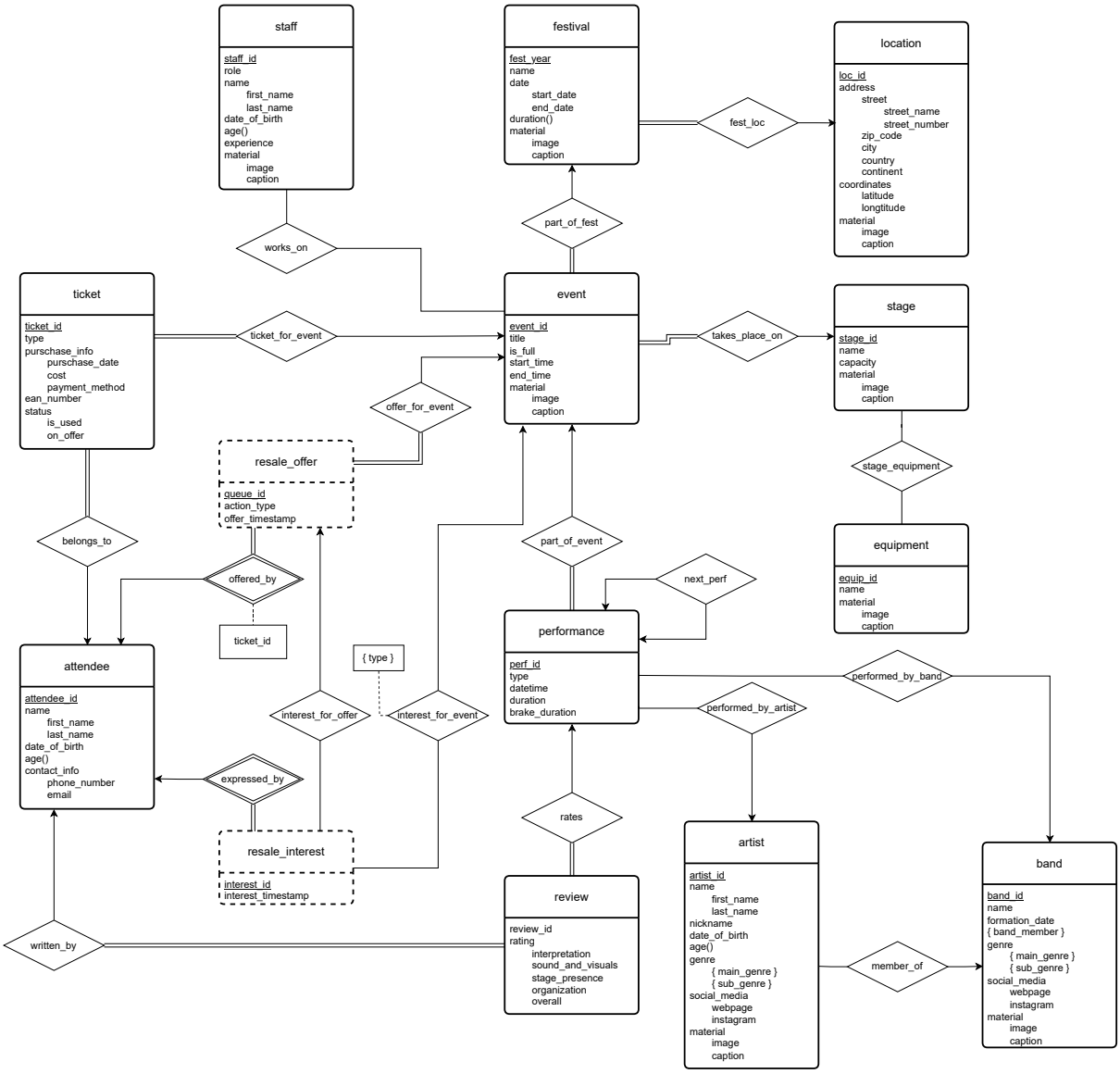
- `cli/`: Περιέχει το βασικό εργαλείο γραμμής εντολών `db137.py`, το οποίο υλοποιεί όλες τις λειτουργίες διαχείρισης της βάσης: δημιουργία/διαγραφή/φόρτωση, reset, εκτέλεση ερωτημάτων, καθώς και εντολές για χρήστες. Το υποσύστημα `users/manager.py` είναι υπεύθυνο για τη σύνδεση με τη MySQL και τον χειρισμό χρηστών και δικαιωμάτων.
- `code/`: Περιέχει scripts που υποστηρίζουν την παραγωγή, τον έλεγχο και την ανάλυση δεδομένων:
  - `data_generation/`: Περιλαμβάνει τους δύο τρόπους παραγωγής δεδομένων:
    - `faker.py`: Εκτελεί εισαγωγή δεδομένων απευθείας στη βάση, σε συμμόρφωση με triggers και constraints.
    - `faker_sql.py`: Δημιουργεί trigger-safe SQL dump για φόρτωση offline μέσω `load.sql`.
  - `code_utils/`: Scripts για αυτόματη δημιουργία ερωτημάτων και ελέγχους αρχείων, όπως:
    - `qgen.py`: Δημιουργία queries και αποτελεσμάτων.
    - `fixeof.py`, `dropgen.py`: Επεξεργασία και διόρθωση τερματισμού αρχείων ή constraints.
  - `organization/`: Περιλαμβάνει το `struct.py`, που αναλύει και ελέγχει τη συνοχή της δομής του έργου.
- `sql/`: Κεντρικός φάκελος με όλο το SQL schema και τα graded ερωτήματα:
  - `install.sql`: Ορίζει όλες τις οντότητες και τους περιορισμούς.
  - `indexing.sql`, `triggers.sql`, `procedures.sql`, `views.sql`: Εξειδικευμένα αρχεία με indexes, triggers (33), stored procedures (7) και views.
  - `queries/`: Αρχεία `Q01.sql` έως `Q15.sql` και τα αντίστοιχα outputs, με επιπλέον εκδοχές για trace (π.χ. `Q04_plan1_out.txt`).
- `frontend/`: SPA εφαρμογή σε React 18 + Vite, με Tailwind CSS και shadcn/ui για μοντέρνο UI. Περιλαμβάνει:
  - `src/pages/`, `src/tabs/`: Καρτέλες του UI για Schema Overview, Browse Schema, Run Query και Run CLI.
  - `src/api/`: API client για επικοινωνία με Flask backend.
  - `src/auth/`, `src/hooks/`: Authentication flow και custom hooks.
  - `index.html`, `main.jsx`, `App.jsx`: Root αρχεία της εφαρμογής.
- `docs/`: Φάκελος τεκμηρίωσης, με:
  - `assignment.pdf`: Επίσημη εκφώνηση.
  - `ddl.pdf`: Περιγραφή των SQL αρχείων.
  - `report.pdf`: Η παρούσα αναφορά της εργασίας μας.
  - `organization/`: Αναφορές σε συμμετοχές, δομή, δεδομένα (`project_structure.txt`, `db_data.txt`).
- `diagrams/`: ER και σχεσιακά διαγράμματα της βάσης, σε μορφή PDF για εύκολη επισκόπηση.
- `test/`: Περιλαμβάνει το `test_cli.sh`, ένα script που ελέγχει όλες τις CLI λειτουργίες και εξάγει συγκεντρωτικά αποτελέσματα (`test_cli_results.txt`).
- `.envrc`: Ορισμός περιβαλλοντικών μεταβλητών (π.χ. DB credentials, Python path).

Στόχοι σχεδιασμού:

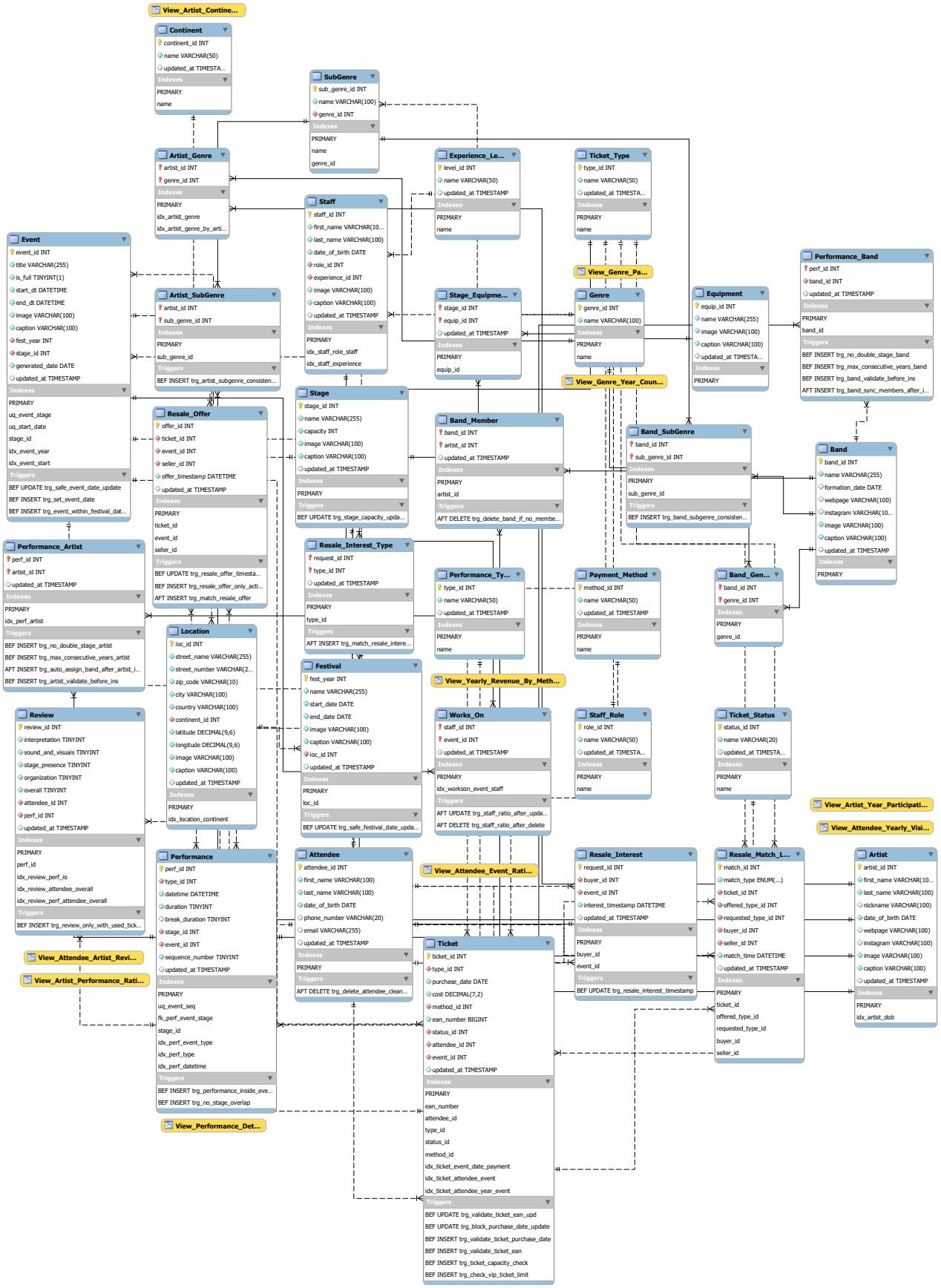
- Ο πλήρης λειτουργικός διαχωρισμός μεταξύ backend (SQL schema, CLI logic) και frontend (SPA user interface).
- Η ακεραιότητα των δεδομένων, με πολυάριθμα triggers και χρήση constraints, ώστε κάθε εισαγωγή και αλλαγή να συμμορφώνεται πλήρως με το schema.
- Η δυνατότητα εύκολης επέκτασης και debugging μέσω εργαλείων δομής και εργαλείων κώδικα (π.χ. `struct.py`, `dropgen.py`).
- Η δυνατότητα εύκολης αξιολόγησης queries και ελέγχου αποτελεσμάτων για benchmarking (με χρήση `_out.txt`, indexing.sql και views).

2 Σχεδιασμός Βάσης Δεδομένων - ER & Relational

2.1 Διάγραμμα Οντοτήτων και Συσχετίσεων - ER



2.2 Σχεσιακό Μοντέλο - Relational



2.3 Παραδοχές Υλοποίησης

Κατά τον σχεδιασμό και την υλοποίηση του συστήματος διαχείρισης δεδομένων για το φεστιβάλ *Pulse University*, ελήφθησαν υπόψη οι προδιαγραφές της εκφώνησης. Παρόλα αυτά, προκειμένου να διασφαλιστεί η πληρότητα και η ορθότητα της βάσης δεδομένων, έγιναν οι ακόλουθες **παραδοχές υλοποίησης**, οι οποίες δεν καθορίζονται ρητά στην εκφώνηση:

- **Ήπειροι**: Θεωρήθηκε ότι το φεστιβάλ λαμβάνει χώρα κάθε χρόνο σε μία από τις εξής 6 ηπείρους: Αφρική, Ασία, Ευρώπη, Βόρεια Αμερική, Νότια Αμερική, Ωκεανία.
- **Staff Roles**: Θεωρήθηκε ότι κάθε μέλος του προσωπικού λαμβάνει έναν από τους εξής ρόλους: security, support, sound engineer, light technician, stagehand, medic, cleaning, backstage assistant. Όλες οι κατηγορίες πέραν του προσωπικού ασφαλείας (security) και του βοηθητικού προσωπικού (support) θεωρηούνται τεχνικό προσωπικό. Οι περιορισμοί που διατυπώνονται στην εκφώνηση περί security (5%) και support (2%) ερμηνεύτηκαν στατικά, ως προς τη χωρητικότητα της κάθε σκηνής, και όχι δυναμικά, ως προς τον πραγματικό αριθμό των θεατών. Δεν θεωρήθηκε επιπλέον απαίτηση του τεχνικού προσωπικού για τη λειτουργία κάθε σκηνής.
- **Κατηγορίες εισιτηρίων**: Όλα τα εισιτήρια ανήκουν σε μία από τις παρακάτω κατηγορίες: general, VIP, backstage, early bird, student. Ο μόνος περιορισμςό που επιβλήθηκε επί αυτών αφορά στα VIP εισιτήρια, τα οποία, σύμφωνα με την εκφώνηση, δεν μπορούν να υπερβαίνουν το 10% της χωρητικότητας της σκηνής.
- **Ημερομηνία αγοράς εισιτηρίου**: Θεωρήθηκε ότι η αγορά του κάθε εισιτηρίου μπορεί να γίνει απαραίτητα την ίδια χρονιά (άρα σίγουρα μετά την ολοκλήρωση του προηγούμνενου φεστιβάλ), και έως τη στιγμή έναρξης της παράστασης στην οποίαν αφορά.
- **Αριθμοί EAN13**: Θεωρήθηκαν έγκυροι αριθμοί EAN13 μόνο όσοι πληρούν τον έλεγχο του τελευταίου ψηφίου (checksum).
- **Μουσικά είδη**: Έχουμε θεωρήσει 10 μουσικά είδη, και περί τα 3 υποείδη για το κάθε είδος. Τα είδα και τα υποείδη φαίνονται λεπτομερώς στους πίνακες Genre και SubGenre του σχήματος της Βάσης.
- **Καλλιτέχνες και συγκροτήματα**: Όλοι οι μουσικοί, είτε ερμηνεύον σόλο, είτε σε συγκροτήματα, καταχωρούνται στη Βάση ως ξεχωριστοί καλλιτέχνες (Artists). Κάθε συγκρότημα (Band) είναι ένα μοναδικό υποσύνολο του συνόλου των καλλιτεχνών, πληθικότητας τουλάχιστον δύο. Ένας καλλιτέχνης μπορεί να ανήκει σε κανένα ή περισσότερα συγκροτήματα.
- **Μουσικά είδη και συγκροτήματα**: Έχουμε θεωρήσει ότι κάθε καλλιτέχνης χαρακτηρίζεται, πιθανώς μεταξύ άλλων, και από τα μουσικά είδη όλων των συγκροτημάτων στα οποία είναι μέλος. Το αντίστροφο δεν έχει θεωρηθεί.
- **Αυτόματη συσχέτιση συγκροτήματος σε εμφάνιση**: Αν όλοι οι καλλιτέχνες μιας εμφάνισης είναι μέλη του ίδιου συγκροτήματος, και δεν υπάρχει άλλο μέλος στο συγκρότημα αυτό, τότε θεωρήθηκε ότι αυτομάτως εμφανίζεται και το ίδιο το συγκρότημα.
- **Παραστάσεις ανά ημέρα**: Έχει θεωρηθεί ότι λαμβάνει χώρα ακριβώς μία παράσταση (event) ανά ημέρα του κάθε φεστιβάλ. Η παραδοχή αυτή προέκυψε από την λογική συναλήθευση των δύο παρακάτω προτάσεων της εκφώνησης: "Κάθε επισκέπτης μπορεί να αγοράσει ένα εισιτήριο ανά ημέρα και παράσταση, αλλά μπορεί να αγοράσει εισιτήρια για κάθε παράσταση και ημέρα του φεστιβάλ."

3 Υλοποίηση Σχήματος

3.1 Table Constraints

Σε αυτήν την ενότητα παρουσιάζουμε τους περιορισμούς ακεραιότητας που επιβάλλονται μαζί με τους ορισμούς των πινάκων, στο αρχείο `install.sql`. Οι περιορισμοί αυτοί εξασφαλίζουν τη λογική ορθότητα, τη συνέπεια με την εκφώνηση, αλλά και εν γένει την κατά το δυνατόν πιστή αναπαράσταση των πραγματικών συμβάσεων που σχετίζονται με την οργάνωση ενός φεστιβάλ μουσικής.

3.1.1 Primary Keys

Για κάθε οντότητα έχει οριστεί πρωτεύον κλειδί, το οποίο εγγυάται τη μοναδικότητα των εγγραφών και τη δυνατότητα αξιόπιστης ταυτοποίησής τους. Οι βασικοί πίνακες, όπως οι `Artist`, `Ticket`, `Stage`, κ.λπ., διαθέτουν απλά πρωτεύοντα κλειδιά (auto-increment IDs). Οι πίνακες που μοντελοποιούν συσχετίσεις many-to-many (π.χ. `Artist_Genre`, `Band_Member`) χρησιμοποιούν σύνθετα πρωτεύοντα κλειδιά.

Η χρήση των πρωτεύοντων κλειδιών επιτρέπει την ασφαλή συσχέτιση δεδομένων μεταξύ των οντοτήτων και την αποφυγή διπλοεγγραφών.

3.1.2 Foreign Keys

Η αναφορική ακεραιότητα επιτυγχάνεται με τη χρήση ξένων κλειδιών που συνδέουν τις οντότητες μεταξύ τους. Κάθε τέτοια σύνδεση διασφαλίζει ότι δεν μπορούν να υπάρξουν εγγραφές που αναφέρονται σε στοιχεία που δεν υπάρχουν στον άμεσα συσχετισμένο πίνακα. Ενδεικτικά παραδείγματα είναι η συσχέτιση των εισιτηρίων με τους επισκέπτες (`Ticket.attendee_id`) και των παραστάσεων με τα αντίστοιχα φεστιβάλ (`Event.fest_year`).

Όπου η λογική λειτουργίας του φεστιβάλ το απαιτεί, εφαρμόζεται `ON DELETE CASCADE` για αυτόματο καθαρισμό εξαρτώμενων εγγραφών, ενώ σε κρίσιμα σημεία επιλέγεται `ON DELETE RESTRICT` για αποτροπή ανεπιθύμητων διαγραφών.

3.1.3 Unique Constraints

Για την αποφυγή διπλοτύπων και την εφαρμογή ειδικών κανόνων, έχουν οριστεί περιορισμοί μοναδικότητας (unique constraints) σε συγκεκριμένα πεδία. Χαρακτηριστικά παραδείγματα είναι η μοναδικότητα του ονόματος μιας ηπείρου (`Continent.name`), του κωδικού EAN13 κάθε εισιτηρίου (`Ticket.ean_number`), καθώς και η απαίτηση να μην μπορεί κάποιος επισκέπτης να αγοράσει δεύτερο εισιτήριο για το ίδιο event (`Ticket.attendee_id, event_id`).

Αυτοί οι περιορισμοί αποτρέπουν την ύπαρξη λογικών σφαλμάτων στη βάση δεδομένων και διασφαλίζουν την ορθότητα των δεδομένων σε κρίσιμες περιπτώσεις.

3.1.4 Check Constraints

Για την προστασία της ποιότητας των δεδομένων, έχουν οριστεί περιορισμοί ελέγχου (check constraints) σε συγκεκριμένα πεδία. Ενδεικτικά παραδείγματα περιλαμβάνουν:

- Εξασφάλιση θετικής τιμής στη χωρητικότητα σκηνής (`Stage.capacity > 0`).
- Έλεγχος ορίων στη διάρκεια εμφανίσεων, ορισμένων από την εκφώνηση (`Performance.duration BETWEEN 1 AND 180`).
- Έλεγχος ότι η αγορά εισιτηρίων γίνεται απαραίτητα πριν την έναρξη του event (`Ticket.purchase_date < Event.start_dt`).
- Εξασφάλιση χρήσης έγκυρων URL για εικόνες (`image LIKE 'https://%'`).
- Ορθό format για Instagram handles (`instagram LIKE '@%'`).

Οι περιορισμοί αυτοί αποτρέπουν την εισαγωγή μη αποδεκτών τιμών και προστατεύουν από συντακτικά ή λογικά σφάλματα.

3.1.5 Composite Foreign Keys

Σε περιπτώσεις όπου απαιτείται η ταυτόχρονη σύνδεση δύο ή περισσότερων πεδίων για τον ορισμό μιας σχέσης, χρησιμοποιούνται σύνθετα ξένα κλειδιά. Ένα χαρακτηριστικό παράδειγμα είναι ο πίνακας `Performance`, στον οποίο η αναφορά σε event και stage γίνεται μέσω composite foreign key `(event_id, stage_id)`.

Με αυτόν τον τρόπο διασφαλίζεται ότι κάθε εμφάνιση συνδέεται με το σωστό ζεύγος event-stage, αποτρέποντας την ύπαρξη λανθασμένων συνδυασμών.

3.2 Indexing

Σε αυτήν την ενότητα παρουσιάζουμε τα ευρετήρια (indexes) που έχουν οριστεί στο αρχείο `indexing.sql`. Τα ευρετήρια χρησιμοποιούνται για τη βελτίωση της απόδοσης στις αναζητήσεις, στις συνενώσεις (joins) και στους περιορισμούς μοναδικότητας. Ο ορισμός τους είναι κρίσιμος για την αποδοτική εκτέλεση των ερωτημάτων του φεστιβάλ.

3.2.1 Primary Indexes (Clustered)

Οι πρωτεύοντες δείκτες δημιουργούνται αυτόματα από τα `PRIMARY KEY` κάθε πίνακα και διασφαλίζουν τη γρήγορη προσπέλαση των εγγραφών μέσω των πρωτευόντων κλειδιών. Ενδεικτικά:

- Ο πίνακας `Ticket` έχει primary index στο πεδίο `ticket_id`.
- Ο πίνακας `Performance` έχει primary index στο `perf_id`.

Οι primary indexes είναι clustered και επιβάλλουν τη φυσική διάταξη των εγγραφών στον δίσκο, γεγονός που βελτιστοποιεί τις αναζητήσεις μέσω των primary keys.

3.2.2 Unique Indexes

Κάθε `UNIQUE` constraint δημιουργεί αυτόματα ένα μοναδικό ευρετήριο, το οποίο εξασφαλίζει τόσο τη γρήγορη αναζήτηση όσο και την αποτροπή διπλότυπων τιμών. Για παράδειγμα:

- Ο πίνακας `Ticket` έχει unique index στο πεδίο `ean_number` (EAN-13 barcode).
- Ο πίνακας `Event` διαθέτει unique index στον συνδυασμό `(event_id, stage_id)`.

3.2.3 Secondary Indexes (Non-Unique)

Εκτός των primary και unique indexes, έχουν οριστεί και δευτερεύοντα ευρετήρια για τη βελτιστοποίηση ερωτημάτων που βασίζονται σε μη-μοναδικά πεδία.

Χαρακτηριστικά παραδείγματα:

- `idx_artist_dob` στον πίνακα `Artist`, για ταχύτερη εύρεση καλλιτεχνών βάσει ημερομηνίας γέννησης (χρήσιμο για ερωτήματα ηλικιακής κατάταξης).
- `idx_event_start` στον πίνακα `Event`, που επιταχύνει τα ερωτήματα εύρεσης events βάσει ημερομηνίας έναρξης.
- `idx_ticket_attendee_event` στον πίνακα `Ticket`, για την ταχεία εύρεση εισιτηρίων ανά επισκέπτη και event.

Οι δείκτες αυτοί στοχεύουν στη βελτίωση της απόδοσης σε συχνά εκτελούμενα ερωτήματα και συσχετίσεις.

3.2.4 Composite Indexes

Σε περιπτώσεις όπου τα ερωτήματα φιλτράρουν ταυτόχρονα από περισσότερα πεδία, δημιουργούνται σύνθετα ευρετήρια (composite indexes). Ενδεικτικά:

- `idx_ticket_event_date_payment` στον πίνακα `Ticket`, που επιταχύνει αναζητήσεις εισιτηρίων βάσει `event_id`, `purchase_date` και `method_id`.
- `idx_review_perf_attendee_overall` στον πίνακα `Review`, που διευκολύνει τα ερωτήματα σύγκρισης επιδόσεων καλλιτεχνών ανά επισκέπτη και συνολική αξιολόγηση.

Η επιλογή σύνθετων ευρετηρίων γίνεται με βάση τις πραγματικές ανάγκες των ερωτημάτων, όπως αυτές ορίζονται στις προδιαγραφές της εργασίας.

3.2.5 Index Usage Rationale

Η επιλογή και τοποθέτηση των indexes βασίζεται σε δύο κριτήρια:

1. Την υποστήριξη των ερωτημάτων αξιολόγησης (Q01-Q15), όπως π.χ. η εύρεση ετήσιων εσόδων ανά μέθοδο πληρωμής, οι συμμετοχές καλλιτεχνών και οι αξιολογήσεις.
2. Την αποδοτικότητα στις συνενώσεις (joins) μεταξύ πινάκων μεγάλου όγκου, όπως `Ticket`, `Performance_Artist`, `Review`.

Τα indexes μειώνουν δραστικά τον χρόνο αναζήτησης και προσφέρουν βελτιστοποίηση στην εκτέλεση σύνθετων queries, χωρίς να επιβαρύνουν υπερβολικά την εισαγωγή δεδομένων.

3.3 Triggers

Σε αυτήν την ενότητα παρουσιάζουμε τους περιορισμούς ακεραιότητας που υλοποιούνται μέσω triggers, όπως ορίζονται στο αρχείο `Triggers.sql`. Οι περιορισμοί αυτοί επιβάλλουν πιο σύνθετους κανόνες που δεν μπορούν να εκφραστούν απευθείας μέσω constraints στους πίνακες και εξασφαλίζουν τη δυναμική συνέπεια των δεδομένων, βάσει των λογικών απαιτήσεων του φεστιβάλ.

3.3.1 Participation Rules

Triggers αυτής της κατηγορίας επιβάλλουν κανόνες συμμετοχής καλλιτεχνών και συγκροτημάτων σε εμφανίσεις (performances).

- **Performance Band Validation** (`trg_band_validate_before_ins`): Ελέγχει ότι δεν γίνεται ανάθεση δεύτερου συγκροτήματος στην ίδια εμφάνιση και ότι οι ήδη ανατεθειμένοι καλλιτέχνες είναι μέλη του συγκροτήματος.
- **Performance Artist Validation** (`trg_artist_validate_before_ins`): Ελέγχει ότι όταν υπάρχει ήδη συσχετισμένο συγκρότημα, ο νέος καλλιτέχνης είναι υποχρεωτικά μέλος του. Επίσης, αν δεν υπάρχει συγκρότημα, επιβάλλει κοινή συμμετοχή σε συγκρότημα μεταξύ των καλλιτεχνών.
- **Auto-Band Assignment** (`trg_auto_assign_band_after_artist_ins`): Όταν όλοι οι καλλιτέχνες μιας εμφάνισης ανήκουν στο ίδιο συγκρότημα, αυτό προστίθεται αυτόματα στην Performance.

Οι triggers αυτοί εξασφαλίζουν ότι η συσχέτιση καλλιτεχνών και συγκροτημάτων γίνεται με συνέπεια και ότι δεν δημιουργούνται λανθασμένες ή ασυνεπείς αναθέσεις.

3.3.2 Scheduling Constraints

Αυτή η κατηγορία triggers ελέγχει κανόνες προγραμματισμού (scheduling) και χρονικής τοποθέτησης.

- **No Double Stage Artist** (`trg_no_double_stage_artist`): Αποτρέπει την ανάθεση καλλιτέχνη σε δύο σκηνές την ίδια χρονική στιγμή.
- **No Double Stage Band** (`trg_no_double_stage_band`): Αντίστοιχος έλεγχος για συγκροτήματα.
- **No Stage Overlap** (`trg_no_stage_overlap`): Διασφαλίζει ότι δεν προγραμματίζεται δεύτερη εμφάνιση σε σκηνή όταν υπάρχει χρονική επικάλυψη.
- **Max Consecutive Years** (`trg_max_consecutive_years_artist`, `trg_max_consecutive_years_band`): Επιβάλλουν τον περιορισμό συμμετοχής έως 3 συνεχόμενα έτη.
- **Event Within Festival** (`trg_event_within_festival_dates`): Ελέγχει ότι οι ημερομηνίες των events εμπίπτουν στα όρια του φεστιβάλ.
- **Performance Inside Event** (`trg_performance_inside_event`): Διασφαλίζει ότι η εμφάνιση εκτελείται εντός του χρονικού πλαισίου του event.

Οι triggers αυτοί αποτρέπουν ασυμβατότητες σε επίπεδο προγράμματος και εξασφαλίζουν τη ρεαλιστική λειτουργία του φεστιβάλ.

3.3.3 Data Safety and Cleanup

Triggers που προστατεύουν την ακεραιότητα των δεδομένων κατά την ενημέρωση ή διαγραφή εγγραφών.

- **Safe Date Updates** (`trg_safe_festival_date_update`, `trg_safe_event_date_update`): Ελέγχουν ότι η τροποποίηση ημερομηνιών φεστιβάλ ή event δεν παραβιάζει ήδη προγραμματισμένες εμφανίσεις.
- **Band Auto-Deletion** (`trg_delete_band_if_no_members`): Διαγράφει αυτόματα συγκρότημα όταν δεν έχει πια μέλη.
- **Cascade Cleanup for Attendee** (`trg_delete_attendee_cleanup`): Με την διαγραφή ενός επισκέπτη, διαγράφονται τα εισιτήρια, οι προσφορές, τα ενδιαφέροντα και οι αξιολογήσεις του.

Με αυτά τα triggers, προστατεύεται η συνοχή των δεδομένων σε περιπτώσεις τροποποίησης ή διαγραφής, αποφεύγοντας ασυνεπή ή ”ορφανά” δεδομένα.

3.3.4 Ticketing Rules

Triggers που διασφαλίζουν την ορθή διαχείριση των εισιτηρίων.

- **Capacity Control** (`trg_ticket_capacity_check`): Αποτρέπει την υπέρβαση της χωρητικότητας σκηνής.
- **VIP Ticket Limit** (`trg_check_vip_ticket_limit`): Ελέγχει ότι τα VIP εισιτήρια δεν υπερβαίνουν το 10% της χωρητικότητας.
- **EAN Validation** (`trg_validate_ticket_ean`, `trg_validate_ticket_ean_upd`): Υλοποιούν έλεγχο εγκυρότητας του EAN-13 checksum κατά την εισαγωγή ή τροποποίηση εισιτηρίων.
- **Immutable Purchase Date** (`trg_block_purchase_date_update`): Αποτρέπει την αλλαγή της ημερομηνίας αγοράς μετά την εισαγωγή.
- **Purchase Deadline** (`trg_validate_ticket_purchase_date`): Ελέγχει ότι τα εισιτήρια αγοράζονται πριν την έναρξη του event.

Αυτοί οι triggers εγγυώνται την αξιοπιστία της διαδικασίας έκδοσης και διαχείρισης εισιτηρίων, ευθυγραμμισμένα με τους κανόνες του φεστιβάλ.

3.3.5 Resale and Reviews

Triggers που σχετίζονται με τη λειτουργία μεταπώλησης εισιτηρίων και την αξιολόγηση εμφανίσεων.

- **Resale Eligibility** (`trg_resale_offer_only_active`): Επιτρέπει μεταπώληση μόνο για ενεργά εισιτήρια.
- **Review Eligibility** (`trg_review_only_with_used_ticket`): Επιτρέπει αξιολόγηση μόνο εφόσον έχει χρησιμοποιηθεί εισιτήριο.
- **Immutable Timestamps** (`trg_resale_offer_timestamp`, `trg_resale_interest_timestamp`): Αποτρέπουν την αλλοίωση timestamps σε προσφορές και ενδιαφέροντα.
- **Automatic Matching** (`trg_match_resale_interest`, `trg_match_resale_offer`): Υλοποιούν αυτόματη αντιστοίχιση αγοραστών και πωλητών σε ουρά FIFO.

Αυτοί οι triggers εξασφαλίζουν την εύρυθμη λειτουργία της αγοράς μεταπώλησης και την ορθότητα των αξιολογήσεων.

3.3.6 Operational Rules

- **Staff Ratio Enforcement** (`trg_staff_ratio_after_delete`, `trg_staff_ratio_after_update`): Ελέγχουν ότι η αναλογία ασφαλείας (5%) και υποστήριξης (2%) παραμένει εντός ορίων μετά από αλλαγές στον προγραμματισμό προσωπικού.
- **Stage Capacity Adjustments** (`trg_stage_capacity_update`): Αποτρέπει τη μείωση χωρητικότητας σκηνής και επανενεργοποιεί events ως `is_full = FALSE` σε περίπτωση αύξησης.
- **Event Date Generation** (`trg_set_event_date`): Θέτει αυτόματα το `generated_date` από την ημερομηνία έναρξης του event.

Αυτά τα triggers υποστηρίζουν τη σωστή λειτουργία των επιχειρησιακών διαδικασιών του φεστιβάλ, πέραν της απλής ακεραιότητας.

3.4 Procedures

Η βάση δεδομένων *Pulse University* υλοποιεί σειρά αποθηκευμένων διαδικασιών (procedures) για την εφαρμογή κανόνων ακεραιότητας, συντήρησης δεδομένων και λειτουργικών ελέγχων. Κάθε procedure αναφέρεται αναλυτικά παρακάτω:

3.4.1 UpdateExpiredTickets()

Η διαδικασία αυτή ενημερώνει την κατάσταση των εισιτηρίων (tickets) σε `unused` για όλα τα events που έχουν ολοκληρωθεί (η σημερινή ημερομηνία είναι μεταγενέστερη του `end_dt` του event).

- Επηρεάζονται μόνο τα εισιτήρια με κατάσταση `active` ή `on offer`.
- Αποσκοπεί στην αυτόματη ενημέρωση εισιτηρίων που έμειναν αχρησιμοποίητα μετά το τέλος του event.

3.4.2 ExpireResaleOffers()

Αυτή η procedure διαγράφει όλες τις ενεργές προσφορές μεταπώλησης εισιτηρίων (`Resale_Offer`) που σχετίζονται με events τα οποία έχουν ήδη ολοκληρωθεί.

- Βασίζεται σε JOIN με τον πίνακα `Event` για τον έλεγχο ημερομηνιών.
- Αποτρέπει την ύπαρξη λανθασμένων ή ανενεργών προσφορών στο σύστημα.

3.4.3 ExpireResaleInterests()

Αντίστοιχη με την προηγούμενη, διαγράφει τα `Resale_Interests` (αιτήματα αγοράς) που σχετίζονται με ολοκληρωμένα events.

- Εξυπηρετεί τον καθαρισμό της βάσης από αιτήματα που πλέον δεν έχουν νόημα.

3.4.4 ScanTicket(IN p_ean BIGINT)

Η procedure αυτή προσομοιώνει τη σάρωση ενός εισιτηρίου στην είσοδο ενός event:

- Δέχεται ως παράμετρο τον `ean_number` του εισιτηρίου.
- Ελέγχει αν το εισιτήριο υπάρχει και αν είναι σε `active` κατάσταση.
- Αν είναι έγκυρο, ενημερώνει την κατάστασή του σε `used`.
- Αν δεν είναι έγκυρο, επιστρέφει αντίστοιχο σφάλμα.

3.4.5 RunMaintenance()

Πρόκειται για κεντρική διαδικασία συντήρησης που εκτελεί:

1. Την ανανέωση καταστάσεων εισιτηρίων (`UpdateExpiredTickets`).
2. Τη διαγραφή ληγμένων προσφορών/αιτημάτων (`ExpireResaleOffers`, `ExpireResaleInterests`).
3. Τον έλεγχο στελέχωσης (`check_staff_ratio`) για κάθε event με χρήση cursor.
4. Χειρίζεται σφάλματα ratio check με warnings ανά event.

Αυτοματοποιεί τις βασικές ενέργειες συντήρησης του φεστιβάλ.

3.4.6 sp_rename_self(IN new_name VARCHAR(64))

Αυτή η procedure επιτρέπει στον συνδεδεμένο χρήστη να αλλάξει το username του:

- Εντοπίζει το username του caller μέσω της συνάρτησης `USER()`.
- Εκτελεί δυναμικό SQL `ALTER USER ... RENAME TO` για τη μετονομασία.
- Υλοποιείται με `SQL SECURITY DEFINER` για ελεγχόμενη πρόσβαση.

Αφορά λειτουργία ειδικού ενδιαφέροντος για διαχείριση χρηστών.

3.4.7 check_staff_ratio(IN ev INT)

Η συγκεκριμένη procedure ελέγχει αν η στελέχωση ενός event πληροί τις ελάχιστες απαιτήσεις:

- Τουλάχιστον 5% της χωρητικότητας της σκηνής σε `security staff`.
- Τουλάχιστον 2% της χωρητικότητας σε `support staff`.

Σε περίπτωση παραβίασης, επιστρέφει σφάλμα SQLSTATE '45000'. Χρησιμοποιείται και αυτόνομα, αλλά και μέσα από το `RunMaintenance()`.

Η υλοποίηση αυτών των procedures καλύπτει βασικούς άξονες λειτουργικότητας: συντήρηση δεδομένων, επιχειρησιακοί έλεγχοι, προσομοίωση χρήσης (ticket scanning), και διαχείριση χρηστών, συνεισφέροντας στη συνοχή και αξιοπιστία της βάσης δεδομένων.

3.5 Views

Η υλοποίηση του συστήματος *Pulse University* περιλαμβάνει σειρά από views, τα οποία χρησιμοποιούνται τόσο για την απλοποίηση πολύπλοκων ερωτημάτων, όσο και για τη βελτίωση της αναγνωσιμότητας και της απόδοσης. Κάθε view εξυπηρετεί συγκεκριμένο επιχειρησιακό στόχο, όπως αναλύεται παρακάτω.

3.5.1 View_Yearly_Revenue_By_Method

Το view αυτό συγκεντρώνει τα συνολικά έσοδα από πωλήσεις εισιτηρίων, ομαδοποιημένα ανά έτος διεξαγωγής φεστιβάλ (`fest_year`) και μέθοδο πληρωμής (`method_id`).

- Εξάγει άμεσα τα δεδομένα για το ερώτημα Q01 (Yearly Revenues by Payment Method).
- Υπολογίζει `SUM(t.cost)` πάνω στον πίνακα `Ticket`, συνενώνοντας με `Event` για να εντοπίσει το αντίστοιχο `fest_year`.

3.5.2 View_Artist_Year_Participation

Το view αυτό παρέχει λίστα με καλλιτέχνες και τα έτη στα οποία συμμετείχαν σε φεστιβάλ.

- Χρησιμοποιείται στο ερώτημα Q02 (αν συμμετείχαν σε συγκεκριμένο έτος).
- Συνδέει τους πίνακες `Performance_Artist`, `Performance` και `Event` για την εξαγωγή του έτους συμμετοχής.

3.5.3 View_Performance_Detail

Παρέχει λεπτομέρειες για κάθε εμφάνιση (performance), συνδυάζοντας:

- ID εμφάνισης, event, έτος φεστιβάλ, τύπο εμφάνισης, ημερομηνία.
- Πληροφορίες για τον καλλιτέχνη ή συγκρότημα.

Είναι χρήσιμο για ερωτήματα όπως το Q03 (π.χ. ζεύγη artist-performance με όλα τα context).

3.5.4 View_Artist_Performance_Rating

Υπολογίζει στατιστικά ανά καλλιτέχνη:

- Σύνολο εμφανίσεων (`performance_count`).
- Μέσος όρος βαθμολογίας για `interpretation` και `overall`.

Απαιτείται στα ερωτήματα Q04 (average interpretation) και Q11 (συγκρίσεις συμμετοχών).

3.5.5 View_Attendee_Event_Rating

Συσχετίζει επισκέπτες με events που παρακολούθησαν και υπολογίζει τον μέσο όρο βαθμολογίας που έδωσαν.

- Βασίζεται στο `Ticket`, `Review` και `Performance`.
- Απαιτείται στο ερώτημα Q06 (average rating per attendee per event).

3.5.6 View_Attendee_Yearly_Visits

Παρουσιάζει πόσες παραστάσεις παρακολούθησε κάθε επισκέπτης ανά έτος.

- Υπολογίζει `events_attended` ανά έτος (`YEAR(purchase_date)`).
- Χρησιμοποιείται στο Q09 (επισκέπτες με περισσότερες από 3 συμμετοχές ανά έτος).

3.5.7 View_Genre_Pairs

Παρέχει τα ζεύγη ειδών μουσικής (genres) που καλύπτονται από τους ίδιους καλλιτέχνες.

- Υπολογίζει για καλλιτέχνες που έχουν ήδη εμφανιστεί.
- Αφορά το ερώτημα Q10 (top-3 genre pairs με κοινές εμφανίσεις).

3.5.8 View_Artist_Continents

Δείχνει σε πόσες διαφορετικές ηπείρους έχει εμφανιστεί κάθε καλλιτέχνης.

- Συσχετίζει καλλιτέχνες με locations μέσω `Festival` και `Event`.
- Αφορά το ερώτημα Q13 (καλλιτέχνες με παρουσία σε ≥3 ηπείρους).

3.5.9 View_Genre_Year_Counts

Επιστρέφει τον αριθμό εμφανίσεων ανά είδος μουσικής (genre) και έτος.

- Συνδυάζει `Performance_Detail` και `Artist_Genre`.
- Χρήσιμο για το Q14 (συνεχόμενες χρονιές με ίδιο πλήθος εμφανίσεων).

3.5.10 View_Attendee_Artist_Review

Υπολογίζει τη συνολική βαθμολογία που έχει δώσει κάθε επισκέπτης σε κάθε καλλιτέχνη.

- Βασίζεται στη σύνδεση `Review` και `Performance_Artist`.
- Απαιτείται για το ερώτημα Q15 (top-5 επισκέπτες με υψηλότερες συνολικές αξιολογήσεις ανά καλλιτέχνη).

Τα παραπάνω views σχεδιάστηκαν με σκοπό να απλοποιήσουν την εκτέλεση των σύνθετων ερωτημάτων αξιολόγησης (Q01–Q15), να βελτιώσουν την αναγνωσιμότητα του κώδικα και να μειώσουν τον υπολογιστικό φόρτο μέσω προϋπολογισμένων αποτελεσμάτων.

4 Παραγωγή Συνθετικών Δεδομένων

4.1 Η Διαδικασία της Παραγωγής

Η παραγωγή των συνθετικών δεδομένων υλοποιήθηκε με δύο scripts, γραμμένα σε Python:

- `faker.py`, το οποίο παράγει τα δεδομένα και κατόπιν εκτελεί εισαγωγές απευθείας στη βάση δεδομένων, μέσω της MySQL, χρησιμοποιώντας το `mysql-connector`.
- `faker_sql.py`, το οποίο παράγει τα δεδομένα και τις αντίστοιχες εντολές SQL και κατόπιν τις γράφει στο αρχείο `sql/load.sql`, χωρίς ωστόσο να εκτελεί άμεση εισαγωγή στη βάση.

Και τα δύο σενάρια σέβονται:

- τους **περιορισμούς ακεραιότητας**, όπως αυτοί ορίζονται στο `install.sql`, δηλαδή τα foreign keys και τυχόν `CHECK`, `NOT NULL`, `UNIQUE` constraints, κ.τ.ό.
- τα **triggers**, όπως αυτά ορίζονται στο `triggers.sql`

Επιπλέον, παράγουν και καταχωρούν τα δεδομένα σε ειδική σειρά, η οποία προλαμβάνει τυχόν παραβιάσεις κατά την αναφορά στα αντίστοιχα - έως τότε κενά - πεδία ή αλυσιδωτές εξαρτήσεις, ενώ αμφότερα εκτελούν δοκιμαστικά και δειγματοληπτικά queries απεθείας στη βάση, ώστε να επιτευχθεί η επιτυχής εγγραφή νέων δεδομένων, σύμφωνα με τους αντίστοιχους περιορισμούς.

4.1.1 Εισαγωγή Δεδομένων με `faker.py`

Το αρχείο `faker.py` παράγει συνθετικά δεδομένα και τα εισάγει απευθείας στη βάση δεδομένων, χρησιμοποιώντας το `mysql-connector` της Python. Η παραγωγή και εισαγωγή των δεδομένων γίνονται σειριακά, ως εξής:

1. **Σταθερές οντότητες**: εξοπλισμός, 36 σκηνές (3 ανά έτος), τοποθεσίες (όσες δεν περιλαμβάνονται στα lookup tables, όπως πόλεις και οδοί), 12 φεστιβάλ (σε χρονολογίες 2016-2027, 2 μελλοντικά) και 4-6 events ανά φεστιβάλ, τα οποία όλα τηρούν τους επιβαλλόμενους από τα triggers περιορισμούς, όπως:
    - ένα φεστιβάλ ανά έτος και καταχώρηση δύο μελλοντικών φεστιβάλ, με την παραδοχή ότι το φεστιβάλ του 2025 (φετινό) έχει ήδη ολοκληρωθεί,
    - τα events εντάσσονται εντός της διάρκειας του φεστιβάλ, θεωρώντας ως ημερομηνία του event την ημέρα έναρξής του.
2. **Προσωπικό**: προσωπικό ασφαλείας, προσωπικό υποστήριξης και τεχνικό προσωπικό, με κατανομή βάσει της χωρητικότητας της εκάστοτε σκηνής και χρήση του procedure `check_staff_ratio()` και σεβόμενοι περιορισμούς, όπως:
    - επάρκεια βοηθητικού προσωπικού και προσωπικού ασφαλείας, ως συνάρτηση της χωρητικότητας της εκάστοτε σκηνής,
    - την παραδοχή ότι το προσωπικό εργάζεται μόνο την ημέρα έναρξης του event (σε περίπτωση που αυτό καλύπτει δύο ημέρες, πριν και μετά τις 00:00).
3. **Καλλιτέχνες και μπάντες**: 45 καλλιτέχνες και 10 μπάντες με genres και αντίστοιχα subgenres, τηρώντας constraints που επιβάλλονται από τα triggers, όπως:
    - αμοιβαίος αποκλεισμός solo artists και bands,
    - εισαγωγή νέου καλλιτέχνη σε εμφάνιση, μόνο εάν μοιράζεται κοινή μπάντα με όλους όσους έχουν ήδη ανατεθεί στην εμφάνιση αυτή,
    - η μέγιστη συνεχόμενη συμμετοχή ενός καλλιτέχνη (μέσω μπάντας ή solo) είναι 3 έτη,
    - δεν επιτρέπεται ταυτόχρονη εμφάνιση σε πολλαπλές σκηνές,
    - συμβατότητα του subgenre με το genre στο οποίο ανήκει,
    - και όσα ακόμα αναφέρονται στις αντίστοιχες ενότητες του `triggers.sql`.

4. **Παραστάσεις**: περιλαμβάνονται 3–6 ανά event, με κατανομή 80% μπάντες και 20% solo, και καθορισμένα slots για "warm up", "headline", κ.τ.ό., σεβόμενες τους περιορισμούς που επιβάλλονται από τα triggers, όπως:
    - να βρίσκονται εντός της διάρκειας του event στο οποίο εντάσσονται,
    - να μην επικαλύπτονται,
    - να τηρούνται πιστά οι επιτρεπτοί χρόνοι διαλείμματος.
5. **Συμμετέχοντες, Εισιτήρια και Κριτικές**: εισαγωγή εισιτηρίων:
    - με έγκυρο EAN-13,
    - ημερομηνία αγοράς πριν την έναρξη του event,
    - status (`active`, `used`, `on offer` κ.λπ.) ανάλογα με την ημερομηνία του event στο οποίο αναφέρονται (παρελθοντικό ή μελλοντικό),

    ενώ οι κριτικές του performance πρέπει να συνοδεύονται από attendee με used ticket για το συγκεκριμένο event.
6. **Ουρά μεταπώλησης**: η ουρά μεταπώλησης λειτουργεί ορθά και από την πλευρά του αγοραστή και από αυτη του πωλητή, μέσω triggers, και καταγράφεται σε log (το οποίο φαίνεται με την εκτέλεση της εντολής `db137 viewq`), ενώ τα timestamps στα resale entries παραμένουν αμετάβλητα, για λόγους ακεραιότητας.

Η εισαγωγή τερματίζει με αυτόματη (εσωτερική) εκτέλεση του `db137 db-status` και αποθήκευση του πλήθους των εγγραφών ανά πίνακα στο `docs/organization/db_data.txt`.

4.1.2 Παραγωγή SQL με `faker_sql.py`

Το `faker_sql.py` εκτελεί παραγωγή συνθετικών δεδομένων με την ίδια λογική με το `faker.py`, αλλά καταγράφει τις εντολές SQL που προκύπτουν στο αρχείο `sql/load.sql`, το οποίο:

- μπορεί να εκτελεστεί ανεξάρτητα, σε οποιοδήποτε περιβάλλον, για λόγους αναπαραγωγής των δεδομένων (τα οποία παράγονται πιθανοτικά) και των αποτελεσμάτων των ερωτημάτων,
- ενσωματώνεται στο CLI μέσω της εντολής `load-db --g`, όπου και εκτελείται, γράφει στο αρχείο load.sql και κατόπιν φορτώνει τα δεδομένα του στη βάση.

Η έξοδος του query στο `load.sql` παράγεται μόνο αν κάθε `INSERT` ολοκληρωθεί επιτυχώς. Αν υπάρξει παραβίαση σε επίπεδο entity ή trigger, τότε η εγγραφή παραλείπεται.

4.2 Εξασφάλιση Κάλυψης Ερωτημάτων

Για να διασφαλιστεί η επιτυχής εκτέλεση και η επιστροφή επαρκών και αντιπροσωπευτικών δεδομένων σε όλα τα ερωτήματα (Q01 – Q15), ενσωματώθηκαν ρητές δομές δεδομένων, στην ενότητα 8 των αρχείων `faker.py` και `faker_sql.py`. Ειδικότερα, μεριμνήσαμε για τα εξής:

- **Q1**: Καθορίζεται ηλικιακή κατανομή στους καλλιτέχνες: 65% κάτω των 30 ετών.
- **Q3**: Για κάθε έτος υπάρχουν τουλάχιστον 3 εμφανίσεις τύπου "warm up".
- **Q5**: Έξι καλλιτέχνες κάτω των 30 εξισώνονται σε αριθμό εμφανίσεων με τον πρώτο σε εμφανίσεις, μέσω επιπλέον εμφανίσεων ή νέων bands.
- **Q9**: 50 attendees έχουν 4 ή περισσότερα εισιτήρια στο ίδιο έτος
- **Q11**: Οι attendees #1 και #2 αξιολογούν από κοινού τουλάχιστον 5 εμφανίσεις του ίδιου artist.
- **Q14**: Εμφανίζονται τουλάχιστον 3 εμφανίσεις Rock, Pop και Jazz για καθένα από τα έτη 2024 και 2025.

4.3 Τα Δεδομένα της Βάσης

Ο παρακάτω πίνακας παρουσιάζει το πλήθος εγγραφών ανά πίνακα της βάσης δεδομένων, μετά την εκτέλεση του `faker_sql.py` και την εισαγωγή των δεδομένων από το `load.sql`. Η αποτύπωση πραγματοποιείται αυτόματα μέσω της εντολής `db137 db-status` και αποθηκεύεται στο αρχείο `db_data.txt`.

Πίνακας	**Πλήθος Εγγραφών**
Artist	55
Artist_Genre	100
Artist_SubGenre	100
Attendee	2000
Band	10
Band_Genre	20
Band_Member	29
Band_SubGenre	20
Continent	6
Equipment	30
Event	58
Experience_Level	5
Festival	12
Genre	10
Location	12
Payment_Method	3
Performance	260
Performance_Artist	426
Performance_Band	158
Performance_Type	5
Resale_Interest	20
Resale_Interest_Type	20
Resale_Match_Log	80
Resale_Offer	70
Review	9733
Staff	60
Staff_Role	8
Stage	36
Stage_Equipment	263
SubGenre	30
Ticket	5536
Ticket_Status	4
Ticket_Type	5
Works_On	788

5 Τεκμηρίωση Ερωτημάτων Αξιολόγησης

5.1 Περιγραφή και Ανάλυση Ερωτημάτων Q01–Q15

Σε αυτήν την ενότητα περιγράφεται αναλυτικά η λογική εκτέλεσης κάθε ερωτήματος της εκφώνησης (Q1–Q15) σε επίπεδο SQL.

5.1.1 Q01

Το ερώτημα επιστρέφει τα συνολικά έσοδα από εισιτήρια, ομαδοποιημένα ανά έτος φεστιβάλ και μέθοδο πληρωμής.

1. Το view `View_Yearly_Revenue_By_Method` υπολογίζει το `SUM(cost)` από τον πίνακα `Ticket`, συσχετίζοντας κάθε εισιτήριο με το event και μέσω αυτού με το `fest_year`.
2. Το τελικό query συσχετίζει το view με τον πίνακα `Payment_Method` για να εξάγει τη μέθοδο αγοράς του κάθε εισιτηρίου.
3. Τα αποτελέσματα ταξινομούνται ανά `fest_year` και `payment_method`.

5.1.2 Q02

Σκοπός του ερωτήματος είναι να εμφανίσει όλους τους καλλιτέχνες ενός είδους (π.χ. Rock), επισημαίνοντας ποιοι συμμετείχαν στο φεστιβάλ του τρέχοντος έτους.

1. Ο πίνακας `Artist_Genre` συσχετίζει καλλιτέχνες με τα είδη τους.
2. Η συσχέτιση με τα `Genres` φέρνει την ονομασία του genre.
3. Μέσω `LEFT JOIN` με το view `View_Artist_Year_Participation`, γίνεται έλεγχος αν ο καλλιτέχνης εμφανίστηκε στο φεστιβάλ του τρέχοντος έτους.
4. Η συνθήκη `WHERE` περιορίζει το αποτέλεσμα στο επιθυμητό genre.

5.1.3 Q03

Το ερώτημα εντοπίζει καλλιτέχνες που έχουν εμφανιστεί ως `warm up` περισσότερες από δύο φορές στο ίδιο φεστιβάλ.

1. Το view `View_Performance_Detail` παρέχει πλήρη δεδομένα εμφανίσεων.
2. Φιλτράρονται μόνο οι εμφανίσεις τύπου `warm up`.
3. Ομαδοποίηση ανά καλλιτέχνη και έτος φεστιβάλ.
4. Με το `HAVING COUNT(*) > 2` διατηρούνται μόνο οι καλλιτέχνες που ικανοποιούν την προϋπόθεση.

5.1.4 Q05

Στο Q5 εντοπίζουμε τους νέους καλλιτέχνες (<30 ετών) με τις περισσότερες εμφανίσεις.

1. Υπολογίζεται η ηλικία κάθε καλλιτέχνη μέσω της `date_of_birth` (TIMESTAMPDIFF).
2. Φιλτράρονται οι καλλιτέχνες με ηλικία <30.
3. Από το view `View_Artist_Performance_Rating` ανακτώνται τα counts εμφανίσεων.
4. Εντοπίζεται ο μέγιστος αριθμός εμφανίσεων.
5. Επιστρέφονται όσοι νέοι καλλιτέχνες έχουν αυτόν τον μέγιστο αριθμό.

5.1.5 Q07

Σκοπός του ερωτήματος είναι να βρεθεί το φεστιβάλ με τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού.

1. Από τον πίνακα `Works_On` αντλούνται οι συμμετοχές προσωπικού σε events.
2. Συσχετίζεται το προσωπικό (`Staff`) και το επίπεδο εμπειρίας του (`Experience_Level`).
3. Εξαιρούνται τα roles `security` και `support`.
4. Ομαδοποίηση και υπολογισμός του μέσου όρου `level_id` ανά `fest_year`.
5. Επιλογή του φεστιβάλ με το χαμηλότερο μέσο όρο (LIMIT 1).

5.1.6 Q08

Εντοπίζει τα μέλη του υποστηρικτικού προσωπικού που δεν έχουν ανατεθεί σε event μιας συγκεκριμένης ημερομηνίας.

1. Ο πίνακας `Staff` φιλτράρεται στο role `support`.
2. Για κάθε εργαζόμενο γίνεται έλεγχος μέσω `NOT EXISTS` αν εργάζεται σε κάποιο event της επιλεγμένης ημερομηνίας.
3. Αν δεν υπάρχει τέτοιο event, ο εργαζόμενος συμπεριλαμβάνεται στο αποτέλεσμα.

5.1.7 Q09

Επιστρέφει επισκέπτες που παρακολούθησαν τον ίδιο αριθμό παραστάσεων (πάνω από 3) μέσα στο ίδιο έτος.

1. Το view `View_Attendee_Yearly_Visits` δίνει το πλήθος επισκέψεων ανά έτος.
2. Εντοπίζονται ζεύγη (year, events_attended) με πάνω από έναν επισκέπτη.
3. Επιστρέφονται όλοι οι επισκέπτες που συμμετείχαν σε αυτά τα ζεύγη.

5.1.8 Q10

Βρίσκει τα 3 συχνότερα ζεύγη ειδών μουσικής που καλύπτονται από τους ίδιους καλλιτέχνες.

1. Το view `View_Genre_Pairs` υπολογίζει το πλήθος εμφανίσεων που καλύπτει κάθε ζεύγος genres.
2. Εμφανίζονται τα τρία με το μεγαλύτερο πλήθος μέσω `ORDER BY artist_count DESC LIMIT 3`.

5.1.9 Q11

Το Q11 επιστρέφει τους καλλιτέχνες που έχουν τουλάχιστον 5 εμφανίσεις λιγότερες από τον πιο ενεργό καλλιτέχνη.

1. Από το `View_Artist_Performance_Rating` ανακτώνται τα counts εμφανίσεων.
2. Εντοπίζεται ο μέγιστος αριθμός εμφανίσεων.
3. Φιλτράρονται όσοι καλλιτέχνες έχουν τουλάχιστον 5 λιγότερες εμφανίσεις.

5.1.10 Q12

Υπολογίζει τον απαιτούμενο αριθμό προσωπικού ασφαλείας και υποστήριξης ανά ημέρα διεξαγωγής event.

1. Από τον πίνακα `Event` εντοπίζονται οι ημερομηνίες.
2. Για κάθε event υπολογίζεται:
    - 5% της χωρητικότητας σκηνής για `security`.
    - 2% για `support`.
3. Το αποτέλεσμα επιστρέφει δύο γραμμές ανά event (μία για κάθε κατηγορία).

5.1.11 Q13

Βρίσκει καλλιτέχνες που έχουν εμφανιστεί σε τουλάχιστον 3 ηπείρους.

1. Το view `View_Artist_Continents` επιστρέφει για κάθε καλλιτέχνη το πλήθος distinct ηπείρων.
2. Το query φιλτράρει όσους έχουν `continents_performed >= 3`.

5.1.12 Q14

Εντοπίζει είδη μουσικής με ίδιο αριθμό εμφανίσεων σε δύο συνεχόμενες χρονιές, όπου κάθε έτος έχει τουλάχιστον 3 εμφανίσεις.

1. Το view `View_Genre_Year_Counts` δίνει πλήθος εμφανίσεων ανά genre και έτος.
2. Εντοπίζονται ζεύγη συνεχόμενων ετών με ίσες εμφανίσεις.

5.1.13 Q15

Επιστρέφει τους top-5 επισκέπτες που έχουν δώσει τις υψηλότερες συνολικές αξιολογήσεις σε καλλιτέχνες.

1. Το view `View_Attendee_Artist_Review` υπολογίζει το `SUM(overall)` για κάθε ζεύγος επισκέπτη–καλλιτέχνη.
2. Εμφανίζονται τα 5 μεγαλύτερα συνολικά scores.

Η ανάλυση των παραπάνω queries σε επίπεδο SQL αναδεικνύει τη χρήση views, joins, aggregations και filtering για την κάλυψη των απαιτήσεων της εκφώνησης με αποδοτικό τρόπο.

5.2 Ανάλυση επίδοσης για συγκεκριμένα ερωτήματα

Σε αυτήν την ενότητα αναλύεται η απόδοση των ερωτημάτων Q04 και Q06, εξετάζοντας δύο διαφορετικές υλοποιήσεις:

1. Χρήση view (χωρίς index hints).
2. Εναλλακτικό query με explicit joins και force index.

Για κάθε περίπτωση παρουσιάζεται η λογική εκτέλεσης, η χρήση ευρετηρίων και οι μετρήσεις των εντολών `EXPLAIN` `EXPLAIN ANALYZE` και `OPTIMIZER_TRACE`.

Τέλος, αναλύεται η επίδοση των διαφοετικών Join στρατηγικών, όπως ζητείται, και συγκεκριμένα των αλγορίθμων Hash Join και Nested Loop Join. Αξίζει να σημειωθεί ότι η σύγκριση έγινε αγνοώντας όλους τους δείκτες (indexes) των σχετικών πινάκων. Αυτό επιλέχθηκε κυρίως ώστε η σύγκριση να αναδεικύει την πραγματική διαφορά στην επίδοση των δύο στρατηγικών. Επιπλέον, η επιλογή αυτή ήταν απαραίτητη, προκειμένου να αναγκάσουμε την MySQL να ακολουθήσει τις στρατηγικές που προτείνουμε, και να μην τις παρακάμψει χρησιμοποιώντας τους δείκτες που έχουμε ορίσει, μη εκτελώντας πλήρως τα joins.

5.2.1 Q04 – Artist Performance Ratings

Το ερώτημα επιστρέφει τις μέσες βαθμολογίες ενός καλλιτέχνη (`artist_id = 7`) για interpretation και overall.

Αρχική Υλοποίηση (View_Artist_Performance_Rating)

- Το view υλοποιεί aggregation ανά καλλιτέχνη μέσω subqueries (COUNT, AVG).
- Ο optimizer κάνει materialize το view, εκτελώντας nested subqueries για κάθε aggregation.
- Για τον `artist_id = 7`, γίνεται index lookup στον πίνακα `Performance_Artist` (idx_perf_artist) και στον πίνακα `Review` (idx_review_perf_io).
- Το `EXPLAIN ANALYZE` έδειξε minimal κόστος, καθώς τα subqueries περιορίζονται στον συγκεκριμένο καλλιτέχνη.

```
## PLAN 2 — RESULT ##
artist_id  artist_name       avg_interpretation  avg_overall
---------  ----------------  ------------------  -----------
7          Artist6 Lastname  3.0139              3.0028

## PLAN 2 — EXPLAIN ##
id  select_type  table  partitions  type   possible_keys       key                 key_len  ref                          rows  filtered  Extra
--  -----------  -----  ----------  -----  ------------------  ------------------  -------  ---------------------------  ----  --------  -----------
1   SIMPLE       a      NULL        const  PRIMARY             PRIMARY             4        const                        1     100.0     NULL
1   SIMPLE       pa     NULL        ref    idx_perf_artist     idx_perf_artist     4        const                        13    100.0     Using index
1   SIMPLE       r      NULL        ref    idx_review_perf_io  idx_review_perf_io  4        pulse_university.pa.perf_id  48    100.0     Using index

## PLAN 2 — EXPLAIN ANALYZE ##
EXPLAIN
-------
-> Group aggregate: avg(r.interpretation), avg(r.overall)  (cost=132 rows=333) (actual time=0.114..0.114 rows=1 loops=1)
    -> Nested loop inner join  (cost=68.5 rows=636) (actual time=0.0168..0.0859 rows=360 loops=1)
        -> Covering index lookup on pa using idx_perf_artist (artist_id=7)  (cost=1.56 rows=13) (actual time=0.0044..0.00614 rows=13 loops=1)
        -> Covering index lookup on r using idx_review_perf_io (perf_id=pa.perf_id)  (cost=0.641 rows=48.9) (actual time=0.0035..0.00497 rows=27.7 loops=13)
```

Σχήμα 1: Ανάλυση απλής υλοποίησης ερωτήματος Q04.

Εναλλακτική Υλοποίηση (Explicit Joins + FORCE INDEX)

- Πραγματοποιούνται απευθείας joins:
  - `Artist` με `Performance_Artist` (FORCE INDEX idx_perf_artist).
  - `Performance_Artist` με `Review` (FORCE INDEX idx_review_perf_io).
- Ομαδοποίηση με `GROUP BY` στο `artist_id`.
- Ο optimizer επέλεξε covering index scan χωρίς table scans.
- Το κόστος παρέμεινε χαμηλό, αλλά με μικρή υπεροχή ως προς το I/O σε σύγκριση με την view-based λύση (λόγω παράκαμψης των subqueries).

```
## PLAN 1 — RESULT ##
artist_id  artist_name       avg_interpretation  avg_overall
---------  ----------------  ------------------  -----------
7          Artist6 Lastname  3.0139              3.0028

## PLAN 1 — EXPLAIN ##
id  select_type         table       partitions  type    possible_keys                                                  key                 key_len  ref                          rows  filtered  Extra
--  ------------------  ----------  ----------  ------  -------------------------------------------------------------  ------------------  -------  ---------------------------  ----  --------  -----------
1   PRIMARY             <derived2>  NULL        system  NULL                                                           NULL                NULL     NULL                         1     100.0     NULL
2   DERIVED             a           NULL        const   PRIMARY                                                        PRIMARY             4        const                        1     100.0     NULL
3   DEPENDENT SUBQUERY  pa          NULL        ref     PRIMARY,idx_perf_artist                                        idx_perf_artist     4        const                        13    100.0     Using index
3   DEPENDENT SUBQUERY  r           NULL        ref     perf_id,idx_review_perf_io,idx_review_perf_attendee_overall     idx_review_perf_io  4        pulse_university.pa.perf_id  48    100.0     Using index
4   DEPENDENT SUBQUERY  pa          NULL        ref     PRIMARY,idx_perf_artist                                        idx_perf_artist     4        const                        13    100.0     Using index
4   DEPENDENT SUBQUERY  r           NULL        ref     perf_id,idx_review_perf_io,idx_review_perf_attendee_overall     idx_review_perf_io  4        pulse_university.pa.perf_id  48    100.0     Using index
5   DEPENDENT SUBQUERY  pa          NULL        ref     PRIMARY,idx_perf_artist                                        idx_perf_artist     4        const                        13    100.0     Using index

## PLAN 1 — EXPLAIN ANALYZE ##
EXPLAIN
-------
-> Rows fetched before execution  (cost=0..0 rows=1) (actual time=54e-6..74e-6 rows=1 loops=1)
```

Σχήμα 2: Ανάλυση εναλλακτικής υλοποίησης ερωτήματος Q04.

Σύγκριση: Η χρήση του view προσφέρει ήδη ικανοποιητική απόδοση λόγω καλού indexing. Ωστόσο, η explicit join προσέγγιση με index hints απέδωσε ελαφρώς καλύτερα (σε I/O επίπεδο) εξαιτίας του άμεσου ελέγχου στη στρατηγική εκτέλεσης.

Hash Join υλοποίηση

- Ο optimizer χρησιμοποίησε **Inner Hash Joins** για τις σχέσεις `pa.perf_id = r.perf_id` και `a.artist_id = 7`.
- Η πλειοψηφία των εγγραφών διαβάζεται με **Table Scan** και προετοιμάζεται σε hash table.
- Το τελικό αποτέλεσμα (360 εγγραφές) επιτεύχθηκε σε 2ms actual time.

Nested Loop υλοποίηση

- Χρησιμοποιήθηκαν **Nested Loop Inner Joins** σε κάθε επίπεδο.
- Για κάθε εγγραφή στον πίνακα `r` (9727 rows), έγινε επανάληψη σε `pa` και στη συνέχεια σε `a`.
- Το αποτέλεσμα επιτεύχθηκε σε 676ms, με σημαντικά μεγαλύτερο κόστος λόγω επαναλήψεων.

Σύγκριση: O Hash Join απέδωσε πολλαπλάσια καλύτερα, καθώς απέφυγε τις επαναλήψεις του nested loop (O(N^2) συμπεριφορά) και εκμεταλλεύτηκε hash-based matching.

```
## PLAN 3 – EXPLAIN ANALYZE HASH ##
EXPLAIN
-------
-> Table scan on <temporary>  (actual time=2.07..2.07 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=2.07..2.07 rows=1 loops=1)
        -> Inner hash join (no condition)  (cost=640410 rows=52923) (actual time=1.88..1.91 rows=360 loops=1)
            -> Filter: (a.artist_id = 7)  (cost=303e-6 rows=1) (actual time=0.00427..0.0156 rows=1 loops=1)
                -> Table scan on a  (cost=303e-6 rows=55) (actual time=0.00342..0.0139 rows=55 loops=1)
            -> Hash
                -> Inner hash join (pa.perf_id = r.perf_id)  (cost=348279 rows=2.91e+6) (actual time=1.83..1.86 rows=360 loops=1)
                    -> Filter: (pa.artist_id = 7)  (cost=0.00415 rows=357) (actual time=0.0342..0.0593 rows=13 loops=1)
                        -> Table scan on pa  (cost=0.00415 rows=426) (actual time=0.0284..0.0461 rows=426 loops=1)
                    -> Hash
                        -> Table scan on r  (cost=979 rows=9727) (actual time=0.0978..1.27 rows=9733 loops=1)


## PLAN 3 – EXPLAIN ANALYZE NESTED LOOP ##
EXPLAIN
-------
-> Table scan on <temporary>  (actual time=676..676 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=676..676 rows=1 loops=1)
        -> Nested loop inner join  (cost=39.2e+6 rows=3.47e+6) (actual time=86.7..676 rows=360 loops=1)
            -> Nested loop inner join  (cost=452798 rows=3.47e+6) (actual time=86.7..671 rows=360 loops=1)
                -> Table scan on r  (cost=979 rows=9727) (actual time=0.0927..1.82 rows=9733 loops=1)
                -> Filter: ((pa.perf_id = r.perf_id) and (pa.artist_id = 7))  (cost=3.85 rows=357) (actual time=0.068..0.0687 rows=0.037 loops=9733)
                    -> Table scan on pa  (cost=3.85 rows=426) (actual time=0.0258..0.0526 rows=426 loops=9733)
            -> Filter: (a.artist_id = 7)  (cost=5.65 rows=1) (actual time=0.00242..0.0135 rows=1 loops=360)
                -> Table scan on a  (cost=5.65 rows=55) (actual time=0.00175..0.0118 rows=55 loops=360)
```

Σχήμα 3: Σύγκριση διαφορετικών στρατηγικών join ερωτήματος Q04.

5.2.2 Q06 – Attendee Event Rating

Το ερώτημα επιστρέφει για έναν επισκέπτη (`attendee_id = 42`) τη μέση βαθμολογία που έδωσε στα events που παρακολούθησε.

Αρχική Υλοποίηση (View_Attendee_Event_Rating)

- Το view υλοποιεί joins μεταξύ `Ticket`, `Event`, `Performance`, `Review`, με aggregation.
- O optimizer κάνει materialize το view σε temporary table.
- Στο `WHERE` εφαρμόζεται φίλτρο `attendee_id = 42` μετά το materialization.
- Το κόστος είναι υψηλότερο, καθώς γίνεται σάρωση όλου του view.

```
## PLAN 1 – RESULT ##
event_id  event_title  avg_event_rating
--------  -----------  ----------------
26        2018 Day 5   4.0000
44        2021 Day 1   4.0000

## PLAN 1 – EXPLAIN ##
id  select_type  table       partitions  type    possible_keys                                                                                      key           key_len  ref                               rows  filtered  Extra
--  -----------  ----------  ----------  ------  ---------------------------------------------------------------------------------------------------  ------------  -------  --------------------------------  ----  --------  ---------------
1   PRIMARY      <derived2>  NULL        ALL     NULL                                                                                               NULL          NULL     NULL                              116   100.0     NULL
2   DERIVED      att         NULL        const   PRIMARY                                                                                            PRIMARY       4        const                             1     100.0     Using temporary
2   DERIVED      t           NULL        ref     attendee_id,idx_ticket_event_date_payment,idx_ticket_attendee_event,idx_ticket_attendee_year_event  attendee_id   4        const                             2     100.0     Using index
2   DERIVED      e           NULL        eq_ref  PRIMARY,uq_event_stage                                                                             PRIMARY       4        pulse_university.t.event_id       1     100.0     NULL
2   DERIVED      p           NULL        ref     uq_event_seq,fk_perf_event_stage,idx_perf_event_type                                               uq_event_seq  4        pulse_university.t.event_id       58    100.0     Using index
2   DERIVED      r           NULL        eq_ref  perf_id,idx_review_perf_io,idx_review_attendee_overall,idx_review_perf_attendee_overall            perf_id       8        pulse_university.p.perf_id,const  1     100.0     NULL

## PLAN 1 – EXPLAIN ANALYZE ##
EXPLAIN
-------
-> Table scan on View_Attendee_Event_Rating  (cost=2.5..2.5 rows=0) (actual time=0.0557..0.0558 rows=2 loops=1)
    -> Materialize  (cost=0..0 rows=0) (actual time=0.0553..0.0553 rows=2 loops=1)
        -> Table scan on <temporary>  (actual time=0.0467..0.047 rows=2 loops=1)
            -> Aggregate using temporary table  (actual time=0.0458..0.0458 rows=2 loops=1)
                -> Nested loop left join  (cost=53.9 rows=116) (actual time=0.0159..0.0355 rows=9 loops=1)
                    -> Nested loop inner join  (cost=13.3 rows=116) (actual time=0.0106..0.0158 rows=9 loops=1)
                        -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.00742..0.00901 rows=2 loops=1)
                            -> Covering index lookup on t using attendee_id (attendee_id=42)  (cost=0.45 rows=2) (actual time=0.00417..0.00459 rows=2 loops=1)
                            -> Single-row index lookup on e using PRIMARY (event_id=t.event_id)  (cost=0.3 rows=1) (actual time=0.00171..0.00173 rows=1 loops=2)
                        -> Covering index lookup on p using uq_event_seq (event_id=t.event_id)  (cost=3.17 rows=58) (actual time=0.00208..0.00301 rows=4.5 loops=2)
                    -> Single-row index lookup on r using perf_id (perf_id=p.perf_id, attendee_id=42)  (cost=0.251 rows=1) (actual time=0.00206..0.00207 rows=0.556 loops=9)
```

Σχήμα 4: Ανάλυση απλής υλοποίησης ερωτήματος Q06.

Εναλλακτική Υλοποίηση (Explicit Joins + FORCE INDEX)

- Πραγματοποιούνται explicit joins:
  - `Ticket` με FORCE INDEX (idx_ticket_attendee_event).
  - Συσχέτιση με `Event`, `Performance`.

- LEFT JOIN με `Review` (FORCE INDEX idx_review_attendee_overall).

- Ομαδοποίηση με `GROUP BY` στο `event_id`.
- Ο optimizer επέλεξε covering index lookups για Ticket και Review.
- Το `EXPLAIN ANALYZE` έδειξε αισθητά μικρότερο κόστος σε σχέση με την υλοποίηση μέσω του view.

```
## PLAN 2 — RESULT ##
event_id  event_title  avg_event_rating
--------  -----------  ----------------
26        2018 Day 5   4.0000
44        2021 Day 1   4.0000

## PLAN 2 — EXPLAIN ##
id  select_type  table  partitions  type    possible_keys                                    key                          key_len  ref                          rows  filtered  Extra
--  -----------  -----  ----------  ------  -----------------------------------------------  ---------------------------  -------  ---------------------------  ----  --------  --------------------------
1   SIMPLE       att    NULL        const   PRIMARY                                          PRIMARY                      4        const                        1     100.0     Using index; Using temporary
1   SIMPLE       t      NULL        ref     idx_ticket_attendee_event                        idx_ticket_attendee_event    4        const                        2     100.0     Using index
1   SIMPLE       e      NULL        eq_ref  PRIMARY,uq_event_stage                           PRIMARY                      4        pulse_university.t.event_id  1     100.0     NULL
1   SIMPLE       p      NULL        ref     uq_event_seq,fk_perf_event_stage,idx_perf_event_type  uq_event_seq            4        pulse_university.t.event_id  58    100.0     Using index
1   SIMPLE       r      NULL        ref     idx_review_attendee_overall                      idx_review_attendee_overall  4        const                        5     100.0     Using where

## PLAN 2 — EXPLAIN ANALYZE ##
EXPLAIN
-------
-> Table scan on <temporary>  (actual time=0.0676..0.0678 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=0.0669..0.0669 rows=2 loops=1)
        -> Nested loop left join  (cost=216 rows=580) (actual time=0.0175..0.0544 rows=9 loops=1)
            -> Nested loop inner join  (cost=13.3 rows=116) (actual time=0.0102..0.0146 rows=9 loops=1)
                -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.00768..0.00928 rows=2 loops=1)
                    -> Covering index lookup on t using idx_ticket_attendee_event (attendee_id=42)  (cost=0.45 rows=2) (actual time=0.00452..0.00496 rows=2 loops=1)
                    -> Single-row index lookup on e using PRIMARY (event_id=t.event_id)  (cost=0.3 rows=1) (actual time=0.00167..0.0017 rows=1 loops=2)
                -> Covering index lookup on p using uq_event_seq (event_id=t.event_id)  (cost=3.17 rows=58) (actual time=0.00161..0.00229 rows=4.5 loops=2)
            -> Filter: (r.perf_id = p.perf_id)  (cost=1.25 rows=5) (actual time=0.00388..0.0042 rows=0.556 loops=9)
                -> Index lookup on r using idx_review_attendee_overall (attendee_id=42)  (cost=1.25 rows=5) (actual time=0.00337..0.00392 rows=5 loops=9)
```

Σχήμα 5: Ανάλυση εναλλακτικής υλοποίησης ερωτήματος Q06.

Σύγκριση: Η explicit join προσέγγιση με force index επέτρεψε στον optimizer να περιορίσει το scanning (targeted index lookups), μειώνοντας το I/O και το temporary table overhead του view.

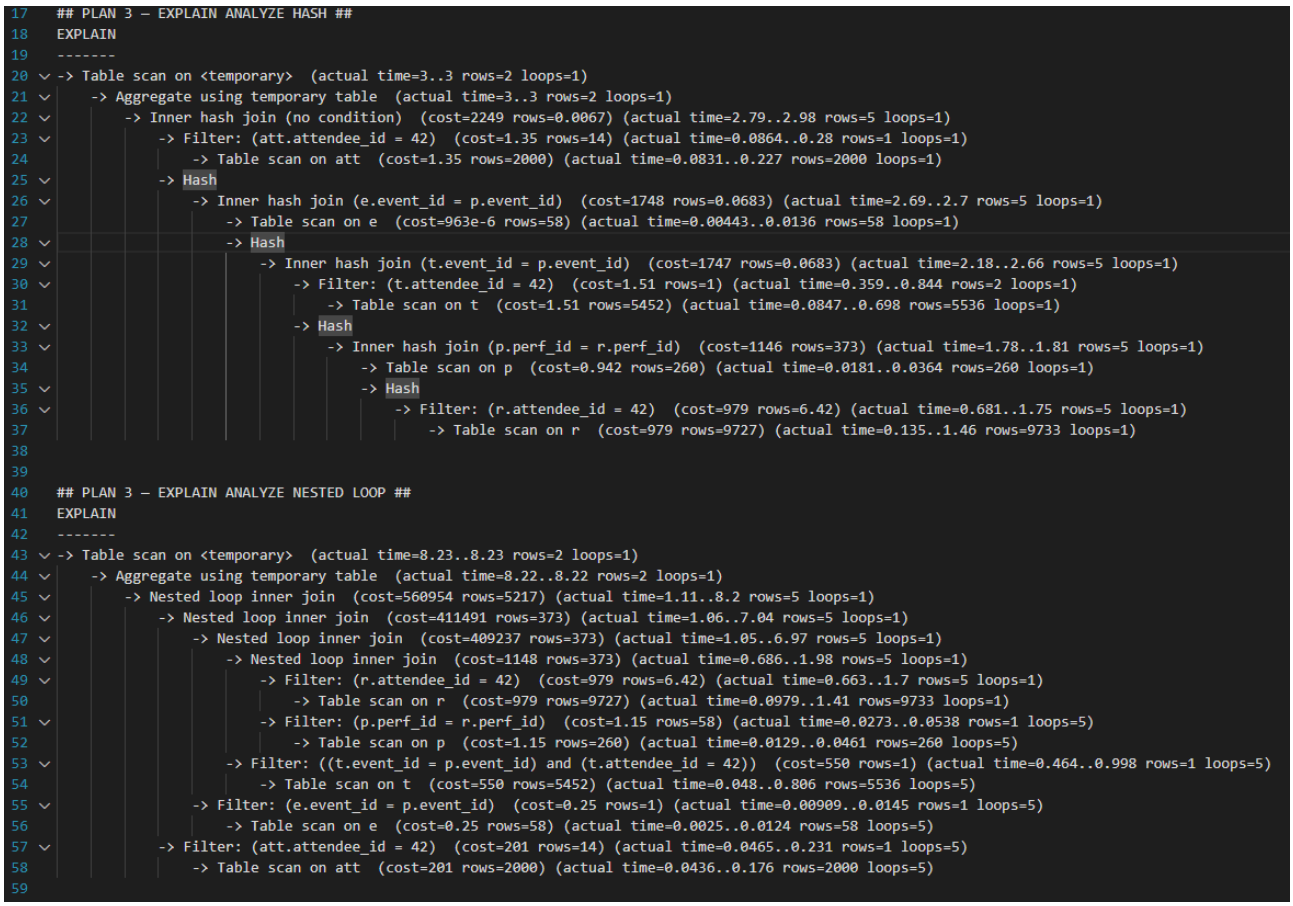
Hash Join υλοποίηση

- Χρησιμοποιήθηκε **Hash Join** για τις σχέσεις:
  - `Performance.perf_id = Review.perf_id`.
  - `Ticket.event_id = Performance.event_id`.
  - `Event.event_id = Performance.event_id`.
  - `Attendee.attendee_id = 42`.
- Παρότι έγιναν full table scans σε όλα τα involved tables (t, p, r, e, att), το αποτέλεσμα υπολογίστηκε σε 3ms.

Nested Loop υλοποίηση

- Κάθε πίνακας συμμετείχε σε **Nested Loop Inner Joins**, με σημαντικό βάθος επαναλήψεων.
- Ο πίνακας `Review` (9727 rows) έγινε outer loop, με εσωτερικές επαναλήψεις σε `Performance`, `Ticket`, `Event`, `Attendee`.
- Το αποτέλεσμα επιτεύχθηκε σε 8.2ms, με αυξημένο κόστος λόγω της πολυπλοκότητας του join plan.

Σύγκριση: Ο Hash Join και εδώ υπερίσχυσε, λόγω αποτελεσματικής χρήσης hash structures για μεγάλους πίνακες (Review, Ticket), ενώ τα Nested Loops παρουσίασαν σημαντική αύξηση κόστους σε I/O.



```
## PLAN 3 – EXPLAIN ANALYZE HASH ##
EXPLAIN
-------
-> Table scan on <temporary>  (actual time=3..3 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=3..3 rows=2 loops=1)
        -> Inner hash join (no condition)  (cost=2249 rows=0.0067) (actual time=2.79..2.98 rows=5 loops=1)
            -> Filter: (att.attendee_id = 42)  (cost=1.35 rows=14) (actual time=0.0864..0.28 rows=1 loops=1)
                -> Table scan on att  (cost=1.35 rows=2000) (actual time=0.0831..0.227 rows=2000 loops=1)
            -> Hash
                -> Inner hash join (e.event_id = p.event_id)  (cost=1748 rows=0.0683) (actual time=2.69..2.7 rows=5 loops=1)
                    -> Table scan on e  (cost=963e-6 rows=58) (actual time=0.00443..0.0136 rows=58 loops=1)
                    -> Hash
                        -> Inner hash join (t.event_id = p.event_id)  (cost=1747 rows=0.0683) (actual time=2.18..2.66 rows=5 loops=1)
                            -> Filter: (t.attendee_id = 42)  (cost=1.51 rows=1) (actual time=0.359..0.844 rows=2 loops=1)
                                -> Table scan on t  (cost=1.51 rows=5452) (actual time=0.0847..0.698 rows=5536 loops=1)
                            -> Hash
                                -> Inner hash join (p.perf_id = r.perf_id)  (cost=1146 rows=373) (actual time=1.78..1.81 rows=5 loops=1)
                                    -> Table scan on p  (cost=0.942 rows=260) (actual time=0.0181..0.0364 rows=260 loops=1)
                                    -> Hash
                                        -> Filter: (r.attendee_id = 42)  (cost=979 rows=6.42) (actual time=0.681..1.75 rows=5 loops=1)
                                            -> Table scan on r  (cost=979 rows=9727) (actual time=0.135..1.46 rows=9733 loops=1)


## PLAN 3 – EXPLAIN ANALYZE NESTED LOOP ##
EXPLAIN
-------
-> Table scan on <temporary>  (actual time=8.23..8.23 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=8.22..8.22 rows=2 loops=1)
        -> Nested loop inner join  (cost=560954 rows=5217) (actual time=1.11..8.2 rows=5 loops=1)
            -> Nested loop inner join  (cost=411491 rows=373) (actual time=1.06..7.04 rows=5 loops=1)
                -> Nested loop inner join  (cost=409237 rows=373) (actual time=1.05..6.97 rows=5 loops=1)
                    -> Nested loop inner join  (cost=1148 rows=373) (actual time=0.686..1.98 rows=5 loops=1)
                        -> Filter: (r.attendee_id = 42)  (cost=979 rows=6.42) (actual time=0.663..1.7 rows=5 loops=1)
                            -> Table scan on r  (cost=979 rows=9727) (actual time=0.0979..1.41 rows=9733 loops=1)
                        -> Filter: (p.perf_id = r.perf_id)  (cost=1.15 rows=58) (actual time=0.0273..0.0538 rows=1 loops=5)
                            -> Table scan on p  (cost=1.15 rows=260) (actual time=0.0129..0.0461 rows=260 loops=5)
                    -> Filter: ((t.event_id = p.event_id) and (t.attendee_id = 42))  (cost=550 rows=1) (actual time=0.464..0.998 rows=1 loops=5)
                        -> Table scan on t  (cost=550 rows=5452) (actual time=0.048..0.806 rows=5536 loops=5)
                -> Filter: (e.event_id = p.event_id)  (cost=0.25 rows=1) (actual time=0.00909..0.0145 rows=1 loops=5)
                    -> Table scan on e  (cost=0.25 rows=58) (actual time=0.0025..0.0124 rows=58 loops=5)
            -> Filter: (att.attendee_id = 42)  (cost=201 rows=14) (actual time=0.0465..0.231 rows=1 loops=5)
                -> Table scan on att  (cost=201 rows=2000) (actual time=0.0436..0.176 rows=2000 loops=5)

```

Σχήμα 6: Σύγκριση διαφορετικών στρατηγικών join ερωτήματος Q06.

Συνολικά, στις περιπτώσεις Q04 και Q06 παρατηρήθηκε ότι:

- Για queries με μικρή επιλεκτικότητα (π.χ. Q04 σε έναν artist), το view είναι αποδεκτά αποδοτικό λόγω pushdown filtering.
- Για queries με aggregation πάνω σε μεγάλα datasets (π.χ. Q06), η explicit join προσέγγιση με index hints παρέχει σημαντικά καλύτερη απόδοση.
- Τα Hash Joins υπερτερούν εμφανώς σε queries με μεγάλους πίνακες και ασαφείς σχέσεις N:M.
- Τα Nested Loops έχουν νόημα όταν οι involved πίνακες είναι μικροί ή οι σχέσεις είναι 1:N με καλό indexing.
- Για queries με heavy aggregation (όπως Q6), τα Hash Joins με χρήση indexes προσφέρουν σημαντική βελτίωση.

6 Χρήση Γραμμής Εντολών - CLI: db137

6.1 Γενική Περιγραφή και Αρχιτεκτονική

Η γραμμή εντολών `db137` αποτελεί την κεντρική διεπαφή διαχείρισης της βάσης δεδομένων του συστήματός μας, επιτρέποντας την εποπτεία του σχήματος, των εγγραφών και των χρηστών μέσω ενός ενιαίου, ασφαλούς και επεκτάσιμου περιβάλλοντος. Το CLI αναπτύχθηκε εξ ολοκλήρου σε **Python 3**, αξιοποιώντας:

- το πλαίσιο `Click`, για την οργάνωση σύνθετων εντολών σε θεματικές ενότητες,
- τη βιβλιοθήκη `mysql-connector-python`, για επικοινωνία με τον MySQL server, και
- περιβαλλοντικές μεταβλητές (`DB_HOST`, `DB_PORT`, `DB_ROOT_USER`), για παραμετροποίηση της σύνδεσης.

Η αρχιτεκτονική του CLI ακολουθεί πολυεπίπεδη λογική:

- το αρχείο `db137.py` λειτουργεί ως entrypoint και ορίζει όλες τις εντολές της γραμμής, ενώ
- η επιχειρησιακή λογική υλοποιείται στην κλάση `UserManager` (`cli/users/manager.py`), η οποία παρέχει:
  - δημιουργία, τροποποίηση και διαγραφή χρηστών,
  - διαχείριση προνομίων και ελέγχων πρόσβασης,
  - εκτέλεση αρχείων SQL και ερωτημάτων,
  - χειρισμό σφαλμάτων μέσω του `errorcode` της MySQL.

Κάθε εντολή του `db137` παρέχει αναλυτικά μηνύματα επιτυχίας ή αποτυχίας, και υποστηρίζει πολλαπλά modes λειτουργίας (π.χ. φόρτωση μέσω `faker.py` ή `load.sql`). Ο σχεδιασμός του CLI δίνει έμφαση στην επαναληψιμότητα, την ακεραιότητα και την ευκολία συντήρησης, τηρώντας αυστηρά όλους τους περιορισμούς του σχήματος και τα ενεργά triggers.

Για την παραμετροποίηση του CLI χρησιμοποιούνται οι ακόλουθες μεταβλητές περιβάλλοντος:

- `DB_ROOT_USER`, `DB_ROOT_PASS` — Στοιχεία σύνδεσης root
- `DB_HOST`, `DB_PORT` — Διεύθυνση και θύρα του MySQL server
- `DB_NAME` — Όνομα βάσης δεδομένων προς διαχείριση
- `PYTHONPATH` — Καθορίζει την τοπική ρίζα του project για σκοπούς ανάπτυξης

6.2 Εντολές db137

Οι εντολές του CLI είναι οργανωμένες σε τρεις βασικές κατηγορίες: *χρήστες*, *βάση δεδομένων*, και *ερωτήματα*, και επεξηγούνται αναλυτικά παρακάτω.

6.2.1 Διαχείριση Χρηστών και Δικαιωμάτων

Οι εντολές αυτές επιτρέπουν πλήρη έλεγχο των χρηστών και των προνομίων τους:

- `users register` — Δημιουργία νέου χρήστη
- `users grant / revoke` — Απόδοση ή αφαίρεση προνομίων σε βάση
- `users rename` — Μετονομασία λογαριασμού χρήστη
- `users passwd` — Αλλαγή κωδικού πρόσβασης
- `users list / whoami` — Λίστα ενεργών χρηστών και τρέχουσα σύνδεση
- `users drop / drop-all` — Διαγραφή μεμονωμένων ή όλων των μη συστημικών χρηστών
- `users set-defaults` — Απόδοση βασικών προνομίων: SELECT, INSERT, UPDATE, DELETE

Η κλάση `UserManager` υλοποιεί όλες τις σχετικές λειτουργίες SQL, διασφαλίζοντας ότι μόνο root χρήστες μπορούν να εκτελούν κρίσιμες εντολές.

6.2.2 Εντολές Δημιουργίας / Φόρτωσης Βάσης

Αφορούν την εγκατάσταση, επαναφορά και φόρτωση της βάσης:

- `create-db` — Εκτελεί τα: `install.sql`, `indexing.sql`, `procedures.sql`, `triggers.sql`, `views.sql`
- `drop-db` — Διαγράφει το πλήρες σχήμα
- `erase-db` — Εκκαθαρίζει όλους τους πίνακες (εκτός των lookup)
- `load-db` — Τρεις λειτουργίες:
  - `--i`: Εκτελεί το `faker.py` για έξυπνη παραγωγή και εισαγωγή δεδομένων
  - `--g`: Εκτελεί το `faker_sql.py` και φορτώνει το παραγόμενο `load.sql`
  - χωρίς flags: Φορτώνει απευθείας το υπάρχον `load.sql`
- `reset-db` — Ολική επαναφορά: drop, create, load
- `db-status` — Προβολή αριθμού εγγραφών ανά πίνακα
- `viewq` — Προβολή του log αντιστοίχισης προσφορών/αιτήσεων μεταπώλησης εισιτηρίων

6.2.3 Εκτέλεση Ερωτημάτων

Κάθε ερώτημα, Q X, εκτελείται μέσω του CLI και αποθηκεύεται ως αρχείο εξόδου:

- `q X` — Εκτελεί το `QX.sql` και γράφει σε `QX_out.txt`
- `q X Y` — Εκτελεί ακολουθία ερωτημάτων από `QX.sql` έως `QY.sql`

Η εκτέλεση υλοποιείται μέσω της `QueryDispatcher` και αξιοποιεί την μέθοδο `run_query_to_file()` του `UserManager`, με έλεγχο διαδρομών και εξαγωγή των αποτελεσμάτων σε μορφή ευανάγνωστου `.txt` αρχείου.

6.3 CLI Testing

Η λειτουργικότητα του CLI επαληθεύεται μέσω του script `test_cli.sh`, το οποίο:

- Εκτελεί σενάρια χρήστη (π.χ. `users register`, `rename`, `grant`)
- Ελέγχει την απόρριψη λανθασμένων εντολών και την επιτυχία έγκυρων περιπτώσεων
- Καταγράφει τα αποτελέσματα στο `test_cli_results.txt`

Το script βρίσκεται στον φάκελο `test/` και εκτελείται ως εξής:

```
chmod +x test_cli.sh
./test_cli.sh
```

Δεν υπάρχει ξεχωριστό script για δοκιμή των triggers, καθώς αυτά επαληθεύονται εμμέσως κατά την παραγωγή δεδομένων με το `faker.py`, το οποίο επιχειρεί μαζικές εισαγωγές, με ενεργούς όλους τους περιορισμούς. Επιπλέον, η αποτυχία τέτοιων εισαγωγών και η εκτέλεση της διαδικασίας `RunMaintenance()` λειτουργούν ως *implicit testing* της επιχειρησιακής λογικής. Ομοίως, οι εντολές διαχείρισης της βάσης (π.χ. `load-db`, `reset-db`) είναι απλοϊκές και έχουν ελεγθεί πολλάκις χειροκίνητα.

7 Web Περιβάλλον Χρήστη - Frontend

7.1 Γενική Περιγραφή και Αρχιτεκτονική

Το frontend της εφαρμογής μας αναπτύχθηκε με χρήση **React 18** και **Vite**, σύμφωνα με τη φιλοσοφία SPA (Single Page Application). Η αισθητική και διαδραστική σχεδίαση επιτεύχθηκε με τη χρήση **Tailwind CSS** και του UI framework `shadcn/ui`, προσφέροντας ένα σύγχρονο, ελαφρύ και εργονομικό περιβάλλον εργασίας.

Η εφαρμογή επιτρέπει την αλληλεπίδραση με το backend μέσω **RESTful API**, το οποίο υλοποιείται στο `serve.py` μέσω `Flask`. Το frontend εξυπηρετεί λειτουργίες όπως προβολή του σχήματος, εκτέλεση ερωτημάτων SQL, εντολών CLI, και σύνδεση/αποσύνδεση χρηστών με πλήρη έλεγχο πρόσβασης.

Δομή κύριων σελίδων και αρχείων:

- `Login.jsx`: Σελίδα σύνδεσης με inline επικύρωση και οπτική ανατροφοδότηση.
- `Logout.jsx`: Σελίδα αποσύνδεσης με καθαρισμό session και μήνυμα αποχαιρετισμού.
- `Main.jsx`: Κεντρική προστατευμένη σελίδα που διαχειρίζεται τα tabs.
- `ProtectedRoute.jsx`: Ελέγχει την εγκυρότητα του token και τον χρόνο συνεδρίας (timeout 15 λεπτών).
- `SchemaTab.jsx`, `BrowseTab.jsx`, `QueryTab.jsx`, `CliTab.jsx`: Υλοποιήσεις των τεσσάρων καρτελών που υλοποιούν την αλληλεπίδραση με το εγκατεστημένο σχήμα της βάσης.

7.2 Εμπειρία Χρήστη και Λογική UI

Το περιβάλλον χρήστη βασίζεται σε πλήρως αντιδραστικά στοιχεία και παρέχει:

- **Πλοήγηση** χωρίς ανανέωση σελίδας (SPA).
- **Αυτόματη εστίαση και syntax highlighting** στον SQL editor.
- **Ζωντανή ενημέρωση του σχήματος** μετά από μεταβαλλόμενα queries και εντολές CLI.
- **Πλήρη διαχείριση σφαλμάτων**, με φιλικά μηνύματα (π.χ. ”permission denied”, ”invalid syntax”, ”query empty”).

Η δομή των tabs είναι πλήρως component-based, με απομόνωση της κατάστασης κάθε καρτέλας. Το ιστορικό CLI αποθηκεύεται στο session και διατηρείται ακόμα και μετά από εναλλαγές μεταξύ καρτελών.

7.3 Δομή Κύριας Σελίδας και Λειτουργικότητα Χρήστη

Η `Main.jsx` οργανώνει την εφαρμογή σε τέσσερις βασικές καρτέλες:

Περιγραφή Καρτελών:

- **Schema Overview**: Παρουσιάζει όλα τα Tables, Views, Triggers και Procedures. Παρέχει SQL preview, κατηγοριοποίηση, και δυνατότητα αντιγραφής.
- **Browse Schema**: Δυναμική προβολή των εγγραφών και των ορισμών των πινάκων, των triggers, των views και των procedures του σχήματος. Περιλαμβάνει κουμπιά εξαγωγής σε μορφή `CSV` και `TXT`, των ορισμών και των δεδομένων.
- **Run Query**: SQL editor με `CodeMirror` που επιτρέπει εκτέλεση ερωτημάτων και την άμεση παρουσίαση αποτελεσμάτων, μέσω του προηγούμενου tab.
- **Run CLI**: Εκτέλεση εντολών CLI (π.χ. `db137 q 1`, `db137 reset-db`), εμφάνιση των διαθέσιμων εντολών και διατήρηση ιστορικού με collapsible blocks.

Ο μηχανισμός πλοήγησης βασίζεται σε `useState` και `React Router`, με κουμπί αποσύνδεσης στην κορυφή της σελίδας.

7.4 Ασφάλεια και Δικαιώματα

7.4.1 Login / Logout

Η σύνδεση υλοποιείται μέσω `Login.jsx`, με:

- Inline επαλήθευση username/password.
- Δυνατότητα εμφάνισης/απόκρυψης κωδικού.
- Ενημέρωση του `sessionStorage` με token και timestamp.
- Αυτόματη πλοήγηση προς το `Main.jsx` σε επιτυχία.

Η αποσύνδεση στη `Logout.jsx` διαγράφει τα session δεδομένα και εμφανίζει προσωποποιημένο μήνυμα (”Goodbye from the DB137, <χρήστη>”). Ο χρήστης μεταφέρεται πίσω στο login με οπτικό κουμπί.
Επιπλέον, το `ProtectedRoute.jsx` επιβάλλει αυτόματη ανακατεύθυνση στο login όταν:

- Το token λείπει ή είναι ληγμένο (διάρκεια: 15 λεπτά).
- Το session είναι κατεστραμμένο ή παραβιασμένο.

7.4.2 Αυθεντικοποίηση και Δικαιώματα

Η ασφάλεια βασίζεται στον έλεγχο ταυτότητας μέσω token και την αποστολή του σε κάθε αίτημα (`Authorization` header), χωρίς διατήρηση συνεδρίας στον server (stateless authentication). Η αρχιτεκτονική της API ακολουθεί RESTful πρότυπα, με ρητά endpoints και διαχωρισμό μεταξύ client και server. Ειδικότερα:

- Η **προστασία των routes** βασίζεται στο `ProtectedRoute.jsx`, με timeout 15 λεπτών.
- Tabs όπως το `Run CLI` και `Run Query` εμφανίζουν μήνυμα λάθους, όταν ο χρήστης δεν έχει δικαιώματα και προβεί σε μη επιτρεπτή ενέργεια.

Δικαιώματα Χρηστών (Privileges):

- **Ο χρήστης `root`** διαθέτει πλήρη δικαιώματα (`ALL PRIVILEGES`) και μπορεί να εκτελέσει κάθε εντολή του CLI, να δημιουργεί, να τροποποιεί ή να διαγράφει άλλους χρήστες, καθώς και να έχει πρόσβαση στα συστήματα της MySQL (`mysql.user`, `SHOW GRANTS`, `DROP DATABASE`, κ.λπ.).
- **Οι μη-root χρήστες** δημιουργούνται με ελάχιστα ή περιορισμένα δικαιώματα (όπως `SELECT`, `INSERT`), τα οποία προσαρμόζονται κατάλληλα μέσω του CLI. Δεν έχουν πρόσβαση σε ευαίσθητα συστήματα ή επικίνδυνες εντολές όπως `drop-db`, `create-db`, `users list`, κ.λπ.
- Τα CLI commands που απαιτούν αυξημένα δικαιώματα δεν εμφανίζονται καθόλου στο tab `Run CLI` για τους απλούς χρήστες.
- Οι προσπάθειες εκτέλεσης μη εξουσιοδοτημένων εντολών επιστρέφουν άμεσα μήνυμα σφάλματος στην διεπαφή, χωρίς να εκτελείται καμία πράξη στον server.

7.5 Αλληλεπίδραση με το API (API Layer & Client)

Η εφαρμογή επικοινωνεί αποκλειστικά με RESTful API που υλοποιείται στο `serve.py`. Το αρχείο `client.js` ορίζει κοινές συναρτήσεις για όλα τα αιτήματα:

- `authHeaders()`: Επιστρέφει τα κατάλληλα HTTP headers με token.
- `safeFetch()`: Κάνει `fetch()` και εκτελεί αυτόματη ανακατεύθυνση σε περίπτωση `401/403`.

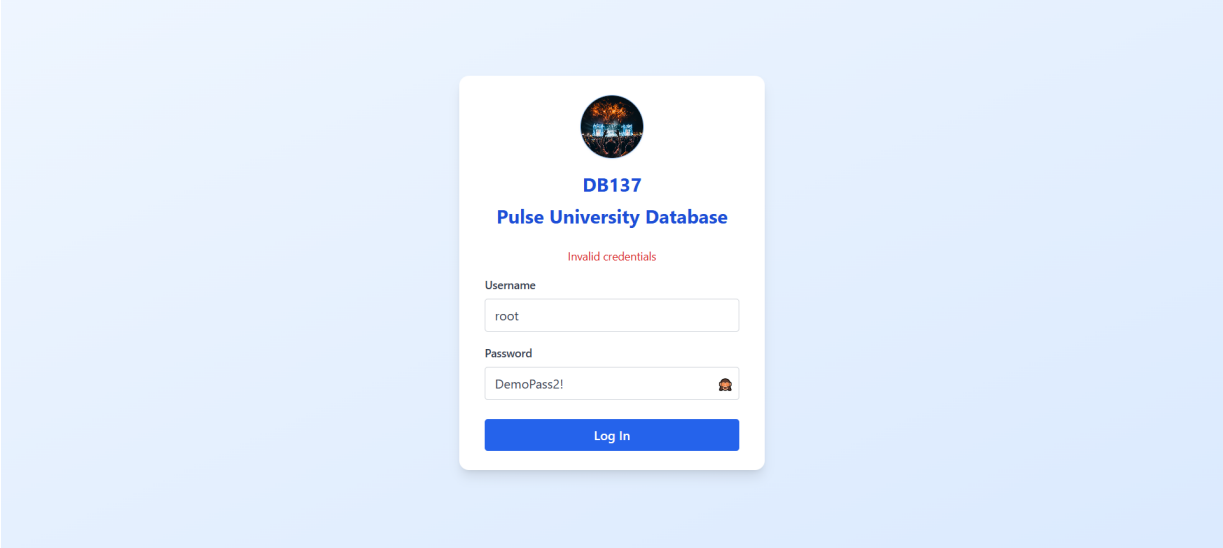
Ενδεικτικά endpoints:

- **Αυθεντικοποίηση**: `POST /api/login`.
- **Σχήμα και δεδομένα**: `POST /api/schema`, `POST /api/browse/{table}`, `GET /api/definition/{table}`.
- **Ερωτήματα**: `POST /api/query`.
- **CLI**: `GET /api/cli/list`, `POST /api/cli/run`.

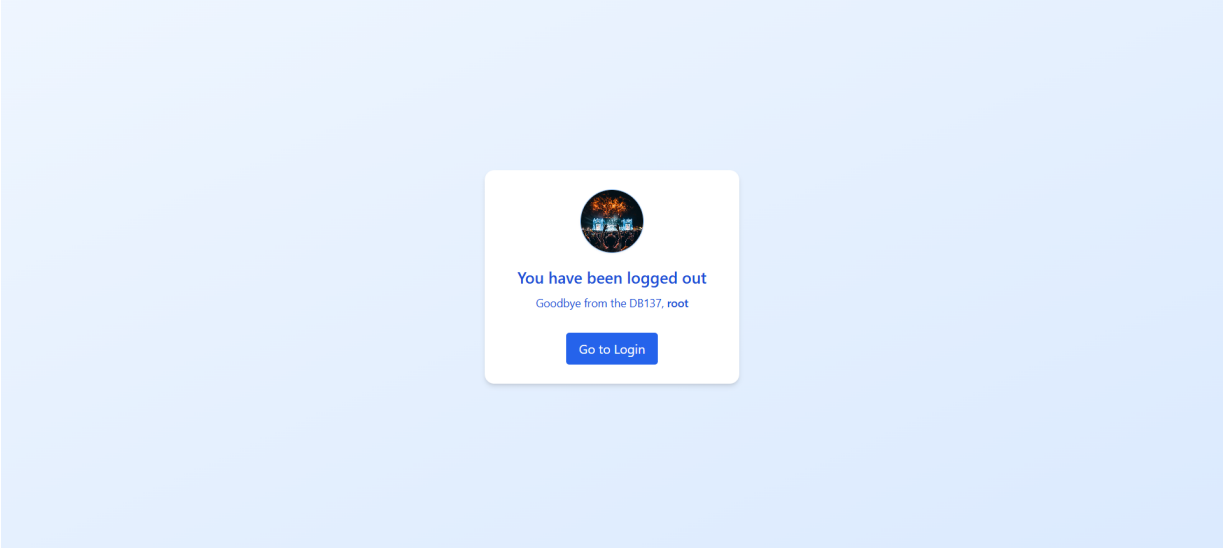
Όλα τα responses είναι σε JSON ή plain text (π.χ. SQL definition), και η λογική του frontend καλύπτει πλήρως τις περιπτώσεις σφάλματος, καθυστερήσεων ή μη εξουσιοδότησης.

7.6 Ενδεικτική Παρουσίαση του Frontend

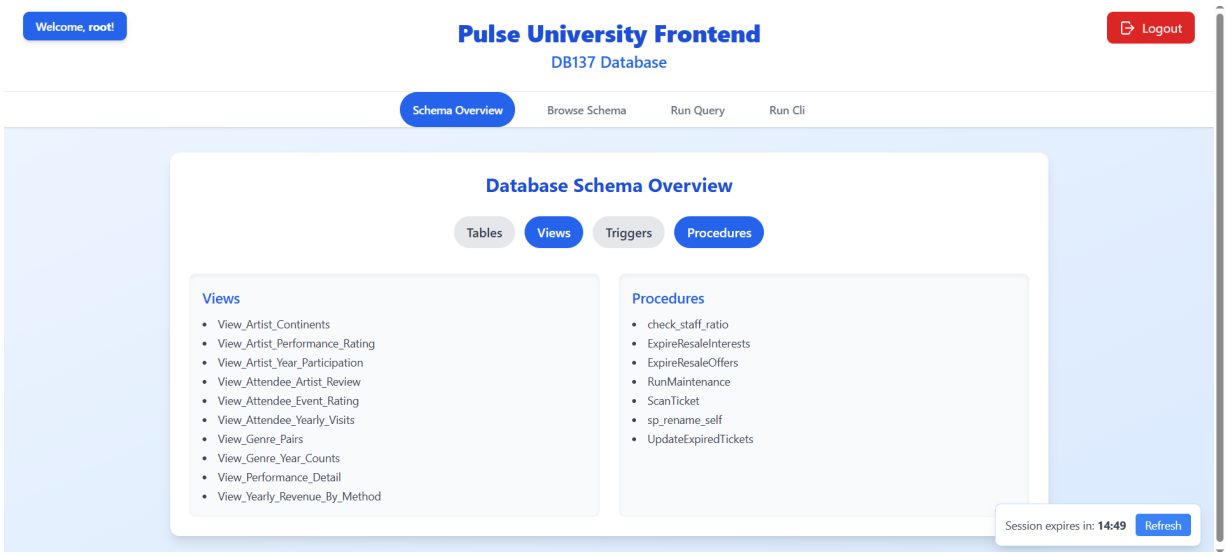
Στην παρούσα ενότητα παρατίθενται ενδεικτικά στιγμιότυπα του περιβάλλοντος χρήστη, τα οποία απεικονίζουν τις βασικές δυνατότητες και τη δομή της εφαρμογής.



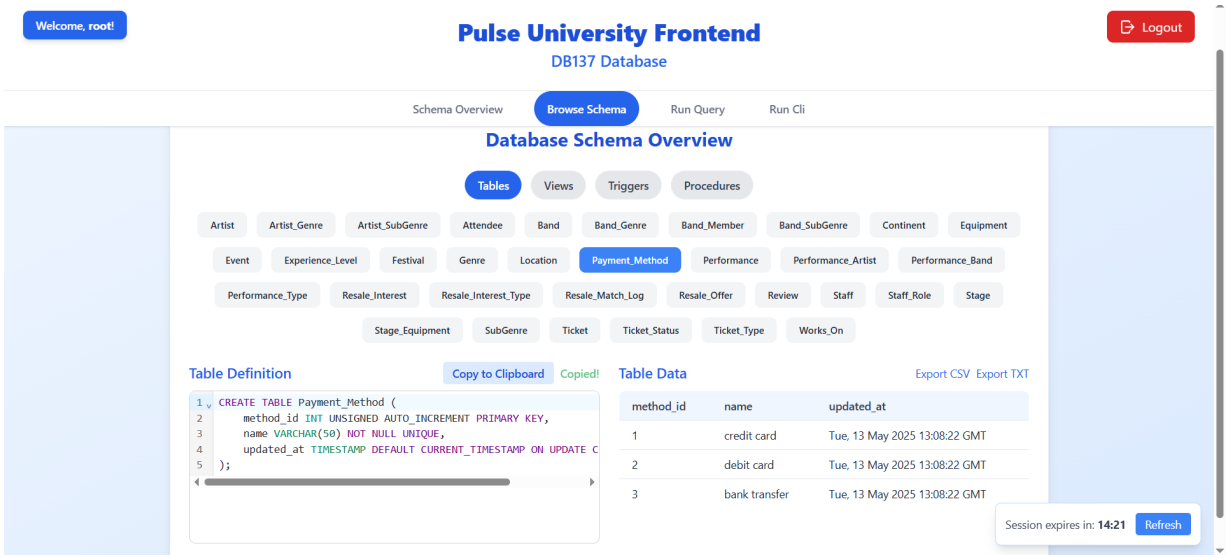
Σχήμα 7: Σελίδα Σύνδεσης — υποστηρίζονται έλεγχος credentials, οπτική ανατροφοδότηση και αποθήκευση token.



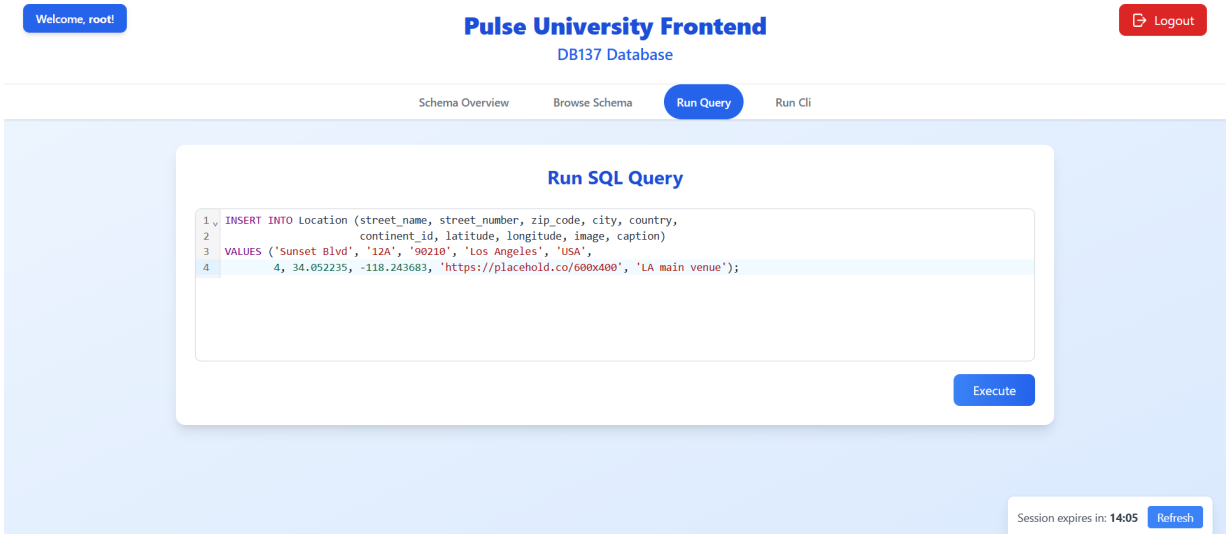
Σχήμα 8: Σελίδα Αποσύνδεσης — καθαρισμός session και προσωποποιημένο μήνυμα αποχαιρετισμού.



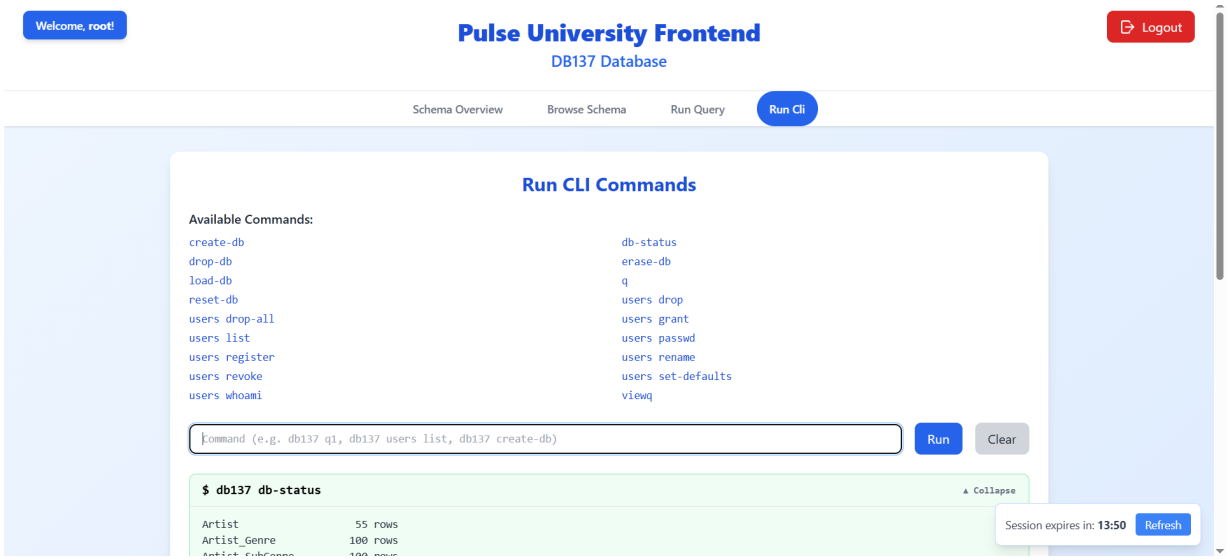
Σχήμα 9: Schema Overview Tab — εμφάνιση των πινάκων, views, triggers και procedures της βάσης.



Σχήμα 10: Browse Tab — εξερεύνηση δεδομένων πίνακα, προβολή ορισμών και δυνατότητα εξαγωγής.



Σχήμα 11: Run Query Tab — εκτέλεση ερωτημάτων SQL με editor και προβολή αποτελεσμάτων.



Σχήμα 12: Run CLI Tab — υποστήριξη εντολών CLI, λίστα διαθέσιμων επιλογών και ιστορικό αποτελεσμάτων.

8 Οδηγίες Εγκατάστασης & Χρήση Εργαλείων AI

8.1 Οδηγίες Εγκατάστασης

8.1.1 Schema

Η εγκατάσταση του σχήματος της βάσης δεδομένων πραγματοποιείται με την εντολή:

```
db137 create-db
```

η οποία εκτελεί διαδοχικά τα ακόλουθα SQL αρχεία:

- `install.sql`: Ορισμός του πλήρους σχήματος και όλων των πινάκων.
- `indexing.sql`: Προσθήκη δεικτών βελτιστοποίησης.
- `procedures.sql`: Αποθηκευμένες διαδικασίες (π.χ. ticket scanning, maintenance).
- `triggers.sql`: Triggers για ελέγχους συνέπειας και λογικής.
- `views.sql`: Views που αντιστοιχούν σε graded queries.

Απαιτείται η ορθή ρύθμιση των μεταβλητών περιβάλλοντος στο αρχείο `.envrc`:

- `DB_ROOT_USER`, `DB_ROOT_PASS`: root credentials.
- `DB_HOST`, `DB_PORT`, `DB_NAME`: πληροφορίες σύνδεσης.

Προβλήματα που ενδέχεται να παρουσιαστούν:

- **DB Connection Error**: λανθασμένα credentials ή host.
- **Privileges Error**: ο χρήστης δεν διαθέτει απαραίτητα δικαιώματα (π.χ. DROP/CREATE).
- **Syntax Error σε SQL αρχείο**: εντοπίζεται μέσω `ClickException`, με αναφορά γραμμής.

8.1.2 CLI

Η εγκατάσταση της γραμμής εντολών `db137` πραγματοποιείται αποκλειστικά σε **Linux ή WSL περιβάλλον**. Τα βήματα είναι τα εξής:

Α. Εξαρτήσεις

- **Python** ≥ 3.10
- **Βιβλιοθήκες Python**:

  ```
  pip install --user click mysql-connector-python
  ```

Β. Ορισμός αρχείου εγκατάστασης Δημιουργείται αρχείο `pyproject.toml` στο project root με το εξής περιεχόμενο:

```
[project]
name = "db137"
version = "0.1"
description = "Pulse University Database Helper CLI"
requires-python = ">=3.8"
dependencies = [
    "click",
    "mysql-connector-python",
]
[project.scripts]
db137 = "cli.db137:cli"
```

Γ. Τοπική Εγκατάσταση (Editable) Από τον root φάκελο του project:

```
pip uninstall db137      # προαιρετικά
pip install --user -e .
```

Αυτό δημιουργεί το εκτελέσιμο `db137` στο `$HOME/.local/bin/db137`.

Δ. Ρύθμιση Shell Για να λειτουργεί η εντολή `db137` από οπουδήποτε:

```
export PATH="$HOME/.local/bin:$PATH"
source ~/.bashrc  # ή ~/.zshrc
which db137        # έλεγχος επιτυχίας
```

Ε. Ρύθμιση Περιβάλλοντος μέσω `.envrc` Ορίζονται οι απαιτούμενες μεταβλητές:

```
export DB_ROOT_USER=$(echo -n 'root' | tr -d '')
export DB_ROOT_PASS=$(echo -n 'yourpassword' | tr -d '')
export PYTHONPATH=$PWD
export DB_HOST='localhost'
export DB_NAME='pulse_university'
export DB_PORT=3306
```

Ενεργοποίηση:

```
direnv allow
```

ΣΤ. Αντιμετώπιση Προβλημάτων

- **MySQL connection fails**:

  ```
  sudo service mysql status
  sudo service mysql start
  ```

- **Blank DB_ROOT_USER or PASS**: Συνήθως σημαίνει ότι το `.envrc` δεν έχει φορτωθεί. Εκτελέστε:

  ```
  direnv allow
  echo $DB_ROOT_USER
  ```

- **`db137: command not found`**: Προσθέστε το `$HOME/.local/bin` στο `PATH`, κάντε `source /.bashrc` και ελέγξτε με `which db137`.
- **FileNotFoundError για SQL αρχεία**: Βεβαιωθείτε ότι έχετε δηλώσει σωστά το `--sql-dir`, π.χ.:

  ```
  db137 create-db --sql-dir sql --database pulse_university
  ```

Ζ. Καθαρισμός Cache και Εγκαταστάσεων (προαιρετικά)

```
find . -type d -name '__pycache__' -exec rm -r {} +
rm -rf db137.egg-info/
```

Η. Εκτέλεση Ελέγχων CLI (Manual Testing)

```
bash test/test_cli.sh
```

Τα αποτελέσματα αποθηκεύονται στο `test/test_cli_results.txt`.

8.1.3 Frontend

Η εγκατάσταση του frontend γίνεται εντός περιβάλλοντος **WSL** και περιλαμβάνει την υλοποίηση SPA διεπαφής με χρήση `React`, `Vite`, `Tailwind CSS` και backend σε `Flask`.

Α. Εγκατάσταση Node.js μέσω nvm

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.bashrc
nvm install 20.11.0
nvm use 20.11.0
```

Β. Εγκατάσταση Εξαρτήσεων Frontend Από τον φάκελο `frontend/`:

```
npm install
npm install --save-dev @vitejs/plugin-react
npm install @uiw/react-codemirror @codemirror/lang-sql
```

Γ. Εγκατάσταση Flask Backend για Local χρήση

```
pip install flask flask-cors mysql-connector-python
```

Δ. Εκκίνηση Frontend και Backend (σε ξεχωριστά terminals)

```
npm run dev                          # Frontend (http://localhost:3000)
python3 frontend/src/api/serve.py    # Backend (port 8000)
```

Ε. Αντιμετώπιση Κοινών Προβλημάτων

- **ERR_INVALID_ARG_TYPE: Received undefined**
  Προκαλείται από απουσία ή βλάβη του `node`. Επανεγκαταστήστε με `nvm`.
- **npm audit warnings**
  Αγνοούνται κατά την ανάπτυξη. Μην εκτελέσετε `npm audit fix --force` — υπάρχει κίνδυνος ασυμβατότητας με `Vite`.
- **Διαδρομές τύπου OneDrive ή προβλήματα δικαιωμάτων**
  Μετακινήστε το έργο εκτός `OneDrive`, π.χ. σε `C:\Projects DB25_137`.
- **Tailwind styles δεν εφαρμόζονται**
  Επιβεβαιώστε ότι το `index.css` περιλαμβάνει:

  ```
  @tailwind base;
  @tailwind components;
  @tailwind utilities;
  ```

- **Failed to resolve import "@/"**
  Ελέγξτε τα alias στο `vite.config.js` — πρέπει να αντιστοιχούν στον φάκελο `src/`.

Ζ. Σημειώσεις Περιβάλλοντος

- Το frontend χρησιμοποιεί `sessionStorage` για token authentication διάρκειας 15 λεπτών
- Όλες οι κλήσεις γίνονται προς το Flask backend στη θύρα `8000`
- Η δομή αρχείων και οι imports βασίζονται στο `vite.config.js` και το `tailwind.config.js`

8.2 Βοηθητικά Εργαλεία Κώδικα

Ο φάκελος `code/` περιέχει scripts για debugging, παραγωγή δεδομένων και συντήρηση του project:

- `data_generation/faker.py`: Εισάγει πλήρη και trigger-compliant δεδομένα απευθείας στη βάση.
- `data_generation/faker_sql.py`: Παράγει SQL queries με χρήση write helpers και αποθηκεύει στο `load.sql`.
- `code_utils/qgen.py`: Δημιουργεί αρχεία `Q01.sql` έως `Q15.sql` και τα αντίστοιχα `Q01_out.txt`, αν δεν υπάρχουν ή είναι κενά.
- `code_utils/fixeof.py`: Εισάγει μια κενή γραμμή στο τέλος σημαντικών αρχείων για αποφυγή σφαλμάτων κατά το parsing.
- `code_utils/dropgen.py`: Προσθέτει στα SQL αρχεία που σχετίζονται με το σχήμα (π.χ. `install`, `triggers`) ένα `DROP IF EXISTS` block πριν από δημιουργία των αντίστοιχων entities, triggers, κ.λπ.
- `organization/struct.py`: Σαρώνει τη δομή του project (πλην των .gitignore) και την τυπώνει στο `docs/oragnization/project_structure.txt` με χρήση της εντολής `tree`.
- `runall.py`: Ολιστικό script που εκτελεί όλα τα παραπάνω, σε σωστή σειρά.

8.3 Χρήση Εργαλείων AI

Κατά την υλοποίηση της εργασίας αξιοποιήθηκε το **ChatGPT Plus (GPT-4)** ως εργαλείο Τεχνητής Νοημοσύνης για καθοδήγηση, ενίσχυση παραγωγικότητας και παραγωγή κώδικα σε συγκεκριμένα υποστηρικτικά σημεία.

Α. Παραγωγή Python & JavaScript Κώδικα

Το AI παρήγαγε έτοιμο και προσαρμόσιμο κώδικα για:

- Shell scripts για CLI testing: `test_cli.sh`.
- Βοηθητικά Python scripts: `dropgen.py`, `fixeof.py`, `qgen.py`, `runall.py`.
- Σύλληψη και ενσωμάτωση generating Scripts (π.χ. `faker.py`).
- Πολύπλοκα python modules, πέραν της εμβέλειας του μαθήματος, όπως `manager.py` και `serve.py`.
- JavaScript και React modules για την οργάνωση του frontend (React Tabs, CLI Terminal, CodeMirror Editor).

Όλα τα παραπάνω επανελέγχθηκαν, διορθώθηκαν ή επεκτάθηκαν από την ομάδα ώστε να διασφαλιστεί η τεχνική πληρότητα, η ασφάλεια και η συμβατότητα με το σύνολο της εφαρμογής.

Β. Υποστήριξη Debugging και Εγκατάστασης

- Ανάλυση σφαλμάτων CLI και React κατά την εγκατάσταση (π.χ. `ModuleNotFoundError`, `import failures`, `nvm issues`).
- Αντιμετώπιση προβλημάτων σε `vite.config.js`, `tailwind.config.js`, `.envrc` περιβάλλον.
- Εντοπισμός λαθών σε privileges, triggers, procedures και άλλους περιορισμούς.

Γ. Μορφοποίηση και Τεκμηρίωση

- Βοήθεια στη διαμόρφωση των αρχείων `README.md`.
- Εξαγωγή σε markdown / LaTeX.

Σημείωση: Όλες οι υλοποιήσεις που σχετίζονται με schema, queries, triggers, constraints και views σε SQL έγιναν χειροκίνητα από την ομάδα, με μόνο συντακτική επιβεβαίωση από την AI. Δεν παραδόθηκε ή χρησιμοποιήθηκε παραγόμενος SQL κώδικας.