2024-2025

ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Project

Ζεκυριάς Απόστολος (1100554)

Καραμαλίκης Ανδρέας (1100562)

**Γενικές Πληροφορίες**

Στις επόμενες σελίδες παρουσιάζονται οι απαντήσεις της ομάδας μας στο Project του εργαστηρίου. Σε αυτήν τη σελίδα παρέχονται πληροφορίες σχετικά με τα μέλη της ομάδας.

Η ομάδα αποτελέιται από τους εξής φοιτητές:

Ζεκυριάς Απόστολος

Καραμαλίκης Ανδρέας

**Αναλυτικότερες Πληροφορίες:**

Απόστολος Ζεκυριάς 1100554

[up1100554@ac.upatras.gr](mailto:up1100554@ac.upatras.gr)

Φοιτητής 3ου έτους

Ανδρέας Καραμαλίκης 1100562

[up1100562@ac.upatras.gr](mailto:up1100613@ac.upatras.gr)

Φοιτητής 3ου έτους

**Περιεχόμενα**

[Κεφάλαιο 1: Δημιουργία της ΒΔ 2](#_Toc188206249)

[Σχεσιακό Διάγραμμα Της Συνολικής Αναθεωρημένης ΒΔ 2](#_Toc188206250)

[Περιγραφή Παραδοχών Σχεδίασης 3](#_Toc188206251)

[Νέοι Πίνακες 3](#_Toc188206252)

[Πίνακας Συναυλιών (Concert) 4](#_Toc188206253)

[Πίνακας Χώρων Διεξαγωγής (Venues) 4](#_Toc188206254)

[Πίνακας Ιστορικού Συναυλιών (ConcertHistory) 4](#_Toc188206255)

[Πίνακας Διαχειριστών Βάσης Δεδομένων (DBA) 5](#_Toc188206256)

[Πίνακας Καταγραφής Ενεργειών (Log) 5](#_Toc188206257)

[Εντολές SQL Για Τη Δημιουργία Της ΒΔ 5](#_Toc188206258)

[Κεφάλαιο 2: Δημιουργία Stored Procedures 5](#_Toc188206259)

[Stored Procedure CalculateVenueScore (3.1.3.1) 5](#_Toc188206260)

[Κώδικας SQL 6](#_Toc188206261)

[Screenshots Από Την Εκέλεση 6](#_Toc188206262)

[Stored Procedure ManageConcerts (3.1.3.2) 7](#_Toc188206263)

[Κώδικας SQL 7](#_Toc188206264)

[Screenshots Από Την Εκέλεση 9](#_Toc188206265)

[Αποτελέσματα αν ο χρήστης εισάγει i 9](#_Toc188206266)

[Αποτελέσματα αν ο χρήστης εισάγει c 10](#_Toc188206267)

[Αποτελέσματα αν ο χρήστης εισάγει a 10](#_Toc188206268)

[Stored Procedure VenueFinder (3.1.3.3) 11](#_Toc188206269)

[Κώδικας SQL 11](#_Toc188206270)

[Screenshots Από Την Εκέλεση 13](#_Toc188206271)

[Stored Procedure TicketSearch (3.1.3.4.Α) 15](#_Toc188206272)

[Screenshots Από Την Εκέλεση 15](#_Toc188206273)

[Stored Procedure VenueNameDates(3.1.3.4.Β) 17](#_Toc188206274)

[Screenshots Από Την Εκτέλεση 17](#_Toc188206275)

[Κεφάλαιο 3: Δημιουργία Trigger 19](#_Toc188206276)

[Trigger (3.1.4.1) 19](#_Toc188206277)

[Trigger Before Insert για τον πίνακα venues: 19](#_Toc188206278)

[Screenshots Από Την Εκτέλεση 24](#_Toc188206279)

[Trigger (3.1.4.2) 24](#_Toc188206280)

[Screenshots Από Την Εκτέλεση 25](#_Toc188206281)

[Trigger (3.1.4.3) 25](#_Toc188206282)

[Screenshots Από Την Εκτέλεση 26](#_Toc188206283)

[Κεφάλαιο 4: Υλοποίηση GUI 26](#_Toc188206284)

# **Κεφάλαιο 1: Δημιουργία της ΒΔ**

## Σχεσιακό Διάγραμμα Της Συνολικής Αναθεωρημένης ΒΔ

## Περιγραφή Παραδοχών Σχεδίασης

Κατά τη διαδικασία σχεδίασης της βάσης δεδομένων, βασιστήκαμε στις ακόλουθες παραδοχές:

1. Κάθε συναυλία αντιστοιχεί σε έναν καλλιτέχνη και έναν χώρο διεξαγωγής.
2. Ο πίνακας του ιστορικού συναυλιών (ConcertHistory ) δεν απαιτεί περιορισμούς αναφορικής ακεραιότητας με άλλους πίνακες.
3. Για λόγους διευκόλυνσης, αποφασίσαμε ότι το πεδίο ReqCapacity του πίνακα Concert και το πεδίο NumTickets του πίνακα ConcertHistory θα έχουν τα ίδια δεδομένα.

## Νέοι Πίνακες

### Πίνακας Συναυλιών (Concert)

Ο πίνακας Concert αποθηκεύει πληροφορίες για τις συναυλίες. Περιέχει τα ακόλουθα πεδία:

* ConID: Μοναδικό αναγνωριστικό της συναυλίας.
* VenID: Μοναδικό αναγνωριστικό του χώρου διεξαγωγής.
* ArtistID: Μοναδικό αναγνωριστικό του καλλιτέχνη.
* ConDate: Ημερομηνία διεξαγωγής της συναυλίας.
* Status: Κατάσταση της συναυλίας (π.χ. scheduled, completed, cancelled).
* ReqCapacity: Η απαιτούμενη χωρητικότητα για να πραγματοποιηθεί η συναυλία.

### Πίνακας Χώρων Διεξαγωγής (Venues)

Ο πίνακας Venues περιέχει στοιχεία για τους διαθέσιμους χώρους διεξαγωγής. Περιέχει τα ακόλουθα πεδία:

* VenID: Μοναδικό αναγνωριστικό του χώρου.
* Name: Το όνομα του χώρου.
* Location: Η τοποθεσία του χώρου.
* Capacity: Η χωρητικότητα του χώρου.
* ConcertsHeld: Ο συνολικός αριθμός των συναυλιών που έχουν πραγματοποιηθεί στον συγκεκριμένο χώρο.
* DateOpened: Η ημερομηνία κατά την οποία άνοιξε ο χώρος διεξαγωγής.

### Πίνακας Ιστορικού Συναυλιών (ConcertHistory)

Ο πίνακας ConcertHistory αποθηκεύει τις ολοκληρωμένες ή ακυρωμένες συναυλίες. Περιέχει τα ακόλουθα πεδία:

* ConcertID: Μοναδικό αναγνωριστικό της συναυλίας.
* ArtistID: Μοναδικό αναγνωριστικό του καλλιτέχνη.
* VenueId: Μοναδικό αναγνωριστικό του χώρου διεξαγωγής.
* NumTickets: Ο αριθμός των εισιτηρίων που πωλήθηκαν.
* ConcertDate: Ημερομηνία διεξαγωγής (ή NULL για ακυρωμένες συναυλίες).
* Status: Κατάσταση της συναυλίας (completed, cancelled).

### Πίνακας Διαχειριστών Βάσης Δεδομένων (DBA)

Ο πίνακας DBA αποθηκεύει πληροφορίες για τους διαχειριστές βάσεων δεδομένων. Περιέχει τα ακόλουθα πεδία:

* Username: Το όνομα χρήστη κάθε διαχειριστή (το οποίο αποτελεί και μοναδικό αναγνωριστικό).
* start\_date: Ημερομηνία ανάληψης καθηκόντων (δεν μπορεί να είναι NULL).
* end\_date: Ημερομηνία λήξης καθηκόντων (NULL εάν είναι ενεργός).

### Πίνακας Καταγραφής Ενεργειών (Log)

Ο πίνακας Log καταγράφει τις ενέργειες των διαχειριστών.

* Username: Αναφορά στον διαχειριστή που πραγματοποίησε την ενέργεια.
* Time: Η ημερομηνία και η ώρα της ενέργειας.
* Αction: Περιγραφή της ενέργειας (π.χ. Insert, Update, Delete).
* Table: Το όνομα του πίνακα που επηρεάστηκε (π.χ. person, band, album, concert, venue).

## Εντολές SQL Για Τη Δημιουργία Της ΒΔ

# **Κεφάλαιο 2: Δημιουργία Stored Procedures**

## Stored Procedure CalculateVenueScore (3.1.3.1)

Η διαδικασία CalculateVenueScore υπολογίζει μια βαθμολογία για έναν χώρο διεξαγωγής με βάση τη χωρητικότητα, τον αριθμό συναυλιών που έχουν πραγματοποιηθεί και τα χρόνια λειτουργίας. Ανακτά τα δεδομένα του χώρου από τον πίνακα venues, υπολογίζει επιμέρους βαθμολογίες (χωρητικότητας, συναυλιών και χρόνων λειτουργίας) και επιστρέφει το συνολικό σκορ. Οι βαθμολογίες βασίζονται σε στρογγυλοποιήσεις και συγκεκριμένους συντελεστές για κάθε παράμετρο. Η διαδικασία αυτή αποτελεί έναν αρκετά τυποποιημένο τρόπο αξιολόγησεις.

### Κώδικας SQL

DELIMITER $

CREATE PROCEDURE CalculateVenueScore(IN venueId INT, OUT venueScore INT)

BEGIN

    DECLARE capacityScore INT;

    DECLARE concertsScore INT;

    DECLARE yearsScore INT;

    DECLARE yearsOfOperation INT;

    SELECT

        Capacity,

        ConcertsHeld,

        TIMESTAMPDIFF(YEAR, DateOpened, CURDATE()) AS YearsOfOperation

    INTO

        @venueCapacity,

        @venueConcerts,

        yearsOfOperation

    FROM

        venues

    WHERE

        VenId = venueId;

--Το FLOOR στρογγυλοποιεί το αποτέλεσμα προς τα κάτω. Το χρησιμοποιούμε ώστε να μην έχουμε δεκαδικόυς αριθμόυς

    SET capacityScore = FLOOR(@venueCapacity / 1000) \* 1;

    SET concertsScore = FLOOR(@venueConcerts / 100) \* 3;

    SET yearsScore = yearsOfOperation \* 2;

    SET venueScore = capacityScore + concertsScore + yearsScore;

END $

DELIMITER ;

Κώδικας Εκτέλεσης Του Procedure

CALL CalculateVenueScore(1, @score);

SELECT @score AS VenueScore;

### Screenshots Από Την Εκέλεση

A screen shot of a computer

Description automatically generated

Αυτό το αποτέλεσμα προκύπτει για το Venue με τα ακόλουθα δεδομένα:

(1, 'Athens Concert Hall', 'Athens, Greece', 2000, 350, '2005-06-15')

## Stored Procedure ManageConcerts (3.1.3.2)

Η διαδικασία ManageConcerts διαχειρίζεται συναυλίες για έναν συγκεκριμένο καλλιτέχνη και ημερομηνία με βάση την ενέργεια που της ζητείται. Ελέγχει αν υπάρχει ήδη συναυλία για τον καλλιτέχνη και την ημερομηνία, και ανάλογα με το action εκτελεί εισαγωγή νέας συναυλίας, ακύρωση υπάρχουσας, ή επανενεργοποίηση ακυρωμένης. Για κάθε περίπτωση, επιστρέφει ένα κατάλληλο μήνυμα που περιγράφει το αποτέλεσμα της ενέργειας.

### Κώδικας SQL

DELIMITER $

CREATE PROCEDURE ManageConcert(

    IN artistId INT,

    IN concertDate DATE,

    IN action CHAR(1)

)

BEGIN

    DECLARE existingStatus ENUM('Scheduled', 'Cancelled', 'Completed');

    DECLARE message VARCHAR(255);

    -- Elegxei an yparxei synaulia gia ton kallitexnh kai thn hmeromhnia

    SELECT Status

    INTO existingStatus

    FROM concert

    WHERE ArtistId = artistId AND ConDate = concertDate;

    -- Cases gia to action

    CASE action

        WHEN 'i' THEN

            -- An yparxei

            IF existingStatus IS NOT NULL THEN

                IF existingStatus = 'Scheduled' THEN

                   SET message = 'A concert is already scheduled on this date.';

                ELSEIF existingStatus = 'Cancelled' THEN

                   SET message = 'A cocnert is cancelled on this date.';

                ELSE

                    SET message = 'The concert has already been completed.';

                END IF;

            ELSE

                -- Kainourgia synaulia me status scheduled

                INSERT INTO concert (VenId, ArtistId, ConDate, Status, ReqCapacity)

                VALUES (NULL, artistId, concertDate, 'Scheduled', NULL);

                SET message = 'A new concert was scheduled successfully.';

            END IF;

        WHEN 'c' THEN

            -- Akyrwsh

            IF existingStatus IS NULL THEN

                SET message = 'No concert exists on this date.';

            ELSEIF existingStatus = 'Cancelled' THEN

               SET message = 'The concert is already cancelled.';

            ELSE

                UPDATE concert

                SET Status = 'Cancelled'

                WHERE ArtistId = artistId AND ConDate = concertDate;

                SET message = 'The concert was cancelled successfully.';

            END IF;

        WHEN 'a' THEN

            -- Ksana energopoiei mia akyrwmenh synaulia

            IF existingStatus IS NULL THEN

                SET message = 'No concert exists on this date.';

            ELSEIF existingStatus = 'Scheduled' THEN

                SET message = 'A concert is already scheduled on this date.';

            ELSEIF existingStatus = 'Cancelled' THEN

                UPDATE concert

                SET Status = 'Scheduled'

                WHERE ArtistId = artistId AND ConDate = concertDate;

                SET message = 'The concert was rescheduled successfully.';

            END IF;

        ELSE

        -- Gia opoiondhpote allo xarakthra

            SET message = 'I dont know what to do with that bro.';

    END CASE;

      -- Epistrefei to mhnuma sto telos

    SELECT message AS ResultMessage;

END$

DELIMITER ;

### Screenshots Από Την Εκέλεση

### Αποτελέσματα αν ο χρήστης εισάγει i

A screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a program

Description automatically generated

### Αποτελέσματα αν ο χρήστης εισάγει c

A computer screen with white text

Description automatically generated

A screen shot of a computer program

Description automatically generated

### Αποτελέσματα αν ο χρήστης εισάγει a

A screen shot of a computer program

Description automatically generated

A screen shot of a concert program

Description automatically generated

## Stored Procedure VenueFinder (3.1.3.3)

Η διαδικασία VenueFinder βρίσκει τον καταλληλότερο χώρο διεξαγωγής (venue) για μια συγκεκριμένη συναυλία, λαμβάνοντας υπόψη την απαιτούμενη χωρητικότητα (ReqCapacity). Επίσης ελέγχει αν η συναυλία είναι προγραμματισμένη και αν υπάρχει ήδη ανατεθειμένος χώρος, ενώ στη συνέχεια αναζητά διαθέσιμους χώρους με επαρκή χωρητικότητα και υπολογίζει τη βαθμολογία τους χρησιμοποιώντας τη διαδικασία CalculateVenueScore από το προηγούμενο ερώτημα. Τελικά, επιστρέφει τον καλύτερο διαθέσιμο χώρο με βάση τη βαθμολογία ή ενημερώνει ότι δεν βρέθηκε κατάλληλος χώρος.

### Κώδικας SQL

DELIMITER $$

CREATE PROCEDURE VenueFinder (

    IN GivenConId INT,

    IN GivenReqCap INT,

    OUT VenIdOutput INT,

    OUT VenCapOutput INT

)

proc\_label:BEGIN

    DECLARE FoundStatus ENUM('Scheduled','Cancelled','Completed') DEFAULT NULL;

    DECLARE FoundVenId INT DEFAULT NULL;

    DECLARE FoundConDate DATE DEFAULT NULL;

    DECLARE TempVenId INT;

    DECLARE TempVenCap INT;

    DECLARE MaxScore INT DEFAULT -1;

    DECLARE CurrentScore INT;

    DECLARE debcount INT DEFAULT 0;

    -- Cursor for venues

    DECLARE VenueCursor CURSOR FOR

        SELECT VenId, Capacity FROM venues

        WHERE Capacity >= GivenReqCap \* 1.1

        AND NOT EXISTS (

            SELECT 1 FROM concert

            WHERE ConDate = FoundConDate AND Status = 'Scheduled' AND concert.VenId = venues.VenId

        );

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET TempVenId = NULL;

    -- Select required values for procedure

    SELECT Status, VenId, ConDate INTO FoundStatus, FoundVenId, FoundConDate FROM concert WHERE ConId = GivenConId;

    IF FoundStatus IS NULL OR FoundStatus = 'Cancelled' THEN

        SET VenIdOutput = NULL;

        SET VenCapOutput = 0;

        LEAVE proc\_label;

    END IF;

    IF FoundVenId IS NOT NULL THEN

        SET VenIdOutput = FoundVenId;

        SELECT Capacity INTO VenCapOutput FROM venues WHERE VenId = FoundVenId;

        SELECT VenIdOutput, VenCapOutput, FoundConDate;

        LEAVE proc\_label;

    END IF;

    OPEN VenueCursor;

    FETCH VenueCursor INTO TempVenId, TempVenCap;

    WHILE TempVenId IS NOT NULL DO

        -- Call CalculateVenueScore for each venue

        CALL CalculateVenueScore(TempVenId, @CurrentScore);

        IF @CurrentScore IS NOT NULL THEN

        SET debcount = debcount + @CurrentScore;

        -- Check if this venue has the highest score

            IF @CurrentScore > MaxScore THEN

                SET MaxScore = @CurrentScore;

                SET VenIdOutput = TempVenId;

                SET VenCapOutput = TempVenCap;

            END IF;

        END IF;

        FETCH VenueCursor INTO TempVenId, TempVenCap;

    END WHILE;

    CLOSE VenueCursor;

    -- If no suitable venue is found, return VenId = NULL and VenCap = 0

    IF MaxScore = -1 THEN

        SET VenIdOutput = NULL;

        SET VenCapOutput = 0;

        Select 'No Eligible Venue Found!';

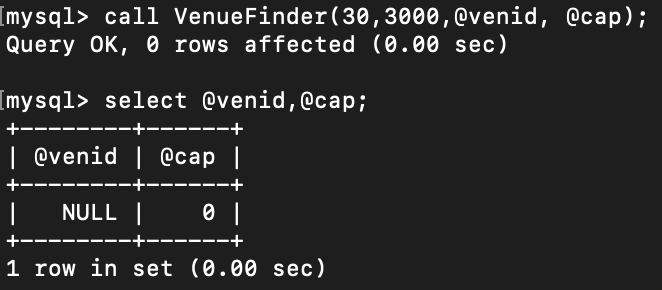
    END IF;

END $$

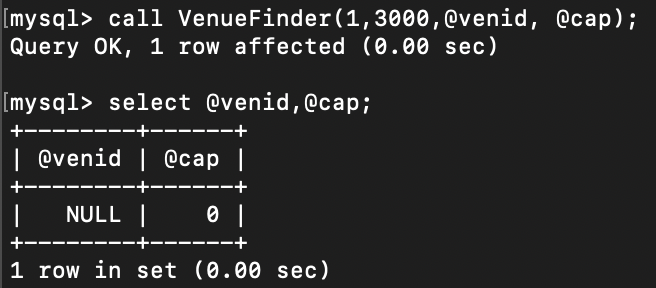
DELIMITER ;

### Screenshots Από Την Εκέλεση

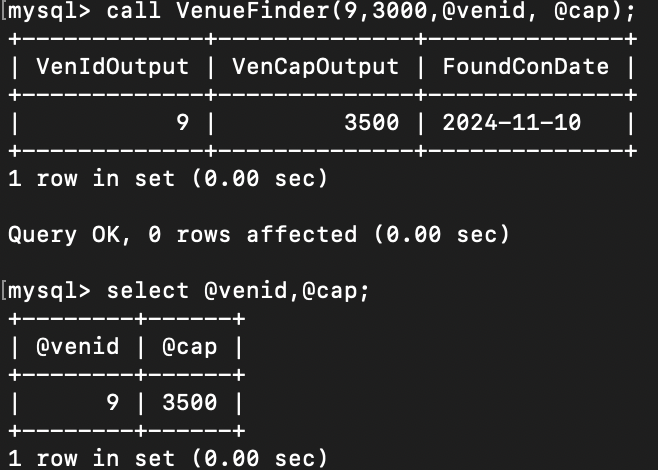
Για την περίπτωση που δεν βρέθηκε το ConId:



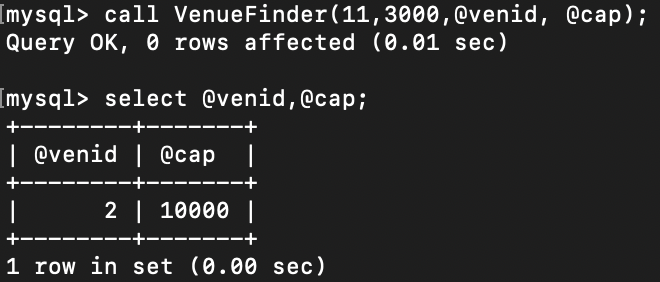
Για την περίπτωση που βρέθηκε το ConId αλλά είναι ακυρωμένη:

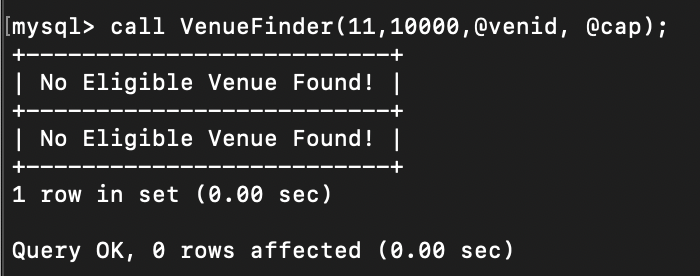


Για την περίπτωση που έχει κλειστεί ήδη χώρος διεξαγωγής για το συγκεκριμένο ConId:



Για την περίπτωση που βρέθηκε το ConId αλλά δεν έχει κλειστεί ήδη χώρος διεξαγωγής:



Για την περίπτωση που δεν είχε κλειστεί χώρος διεξαγωγής αλλά δεν βρέθηκε κατάλληλος χώρος εκείνη την ημερομηνία με την κατάλληλη χωρητικότητα:  


## Stored Procedure TicketSearch (3.1.3.4.Α)

Η διαδικασία αυτή δημιουργεί έναν index στον πίνακα concerthistory για τη στήλη NumTickets, βελτιώνοντας την απόδοση των αναζητήσεων που βασίζονται σε αυτή τη στήλη. Στη συνέχεια, υλοποιεί τη διαδικασία TicketSearch, η οποία αναζητά άτομα (π.χ. καλλιτέχνες) που έχουν πουλήσει έναν αριθμό εισιτηρίων εντός ενός δεδομένου εύρους (minTickets έως maxTickets). Η διαδικασία συνδυάζει τα δεδομένα από τους πίνακες concerthistory και person και επιστρέφει το όνομα και το επώνυμο των καλλιτεχνών που πληρούν τα κριτήρια.

CREATE  INDEX numtickets\_ind ON concerthistory(NumTickets);

DELIMITER $$

CREATE PROCEDURE TicketSearch(

    IN minTickets INT,

    IN maxTickets INT

)

BEGIN

    SELECT

        person.FirstName,

        person.LastName

    FROM

        concerthistory

    INNER JOIN

        person

    ON

        concerthistory.ArtistId = person.ArtistId

    WHERE

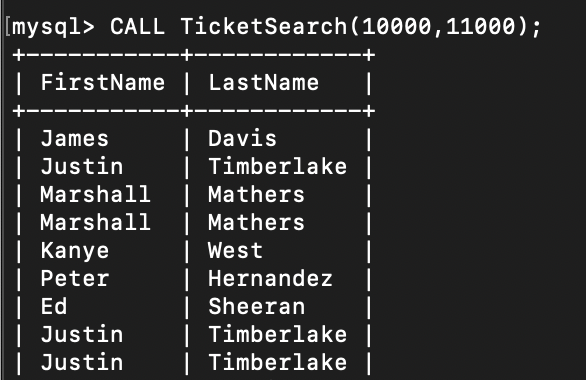
        concerthistory.NumTickets BETWEEN MinTickets AND MaxTickets;

END $$

DELIMITER ;

### Screenshots Από Την Εκέλεση

Με την εκτέλεση του TicketSearch() χρησιμοποιώντας index έχουμε το εξής αποτέλεσμα:



και στο τέλος του εκτυπωμένου πίνακα:



Βλέπουμε πως ο χρόνος εκτέλεσης είναι 0.02 δευτερόλεπτα.

Αν κάνουμε drop το index βλέπουμε πως ο χρόνος αυξάνεται στα 0.04 δευτερόλεπτα, άρα το index πράγματι επιταχύνει την εκτέλεση:



## Stored Procedure VenueNameDates(3.1.3.4.Β)

Η διαδικασία δημιουργεί έναν index στη στήλη VenueId του πίνακα concerthistory για τη βελτίωση της απόδοσης των ερωτημάτων που σχετίζονται με τον συγκεκριμένο χώρο. Στη συνέχεια, υλοποιεί τη διαδικασία VenueNameDate, η οποία δέχεται το όνομα ενός χώρου (givenVenName) ως είσοδο, βρίσκει το αντίστοιχο VenueId από τον πίνακα venues και επιστρέφει τις ημερομηνίες (ConDate) των συναυλιών που έχουν πραγματοποιηθεί σε αυτόν τον χώρο, αναζητώντας στον πίνακα concerthistory.

CREATE  INDEX venid\_ind ON concerthistory(VenueId);

DELIMITER $$

CREATE PROCEDURE VenueNameDates(

    IN givenVenName VARCHAR(250)

)

BEGIN

    DECLARE foundVenId INT DEFAULT NULL;

    SELECT VenId INTO foundVenId FROM venues WHERE VenName = givenVenName;

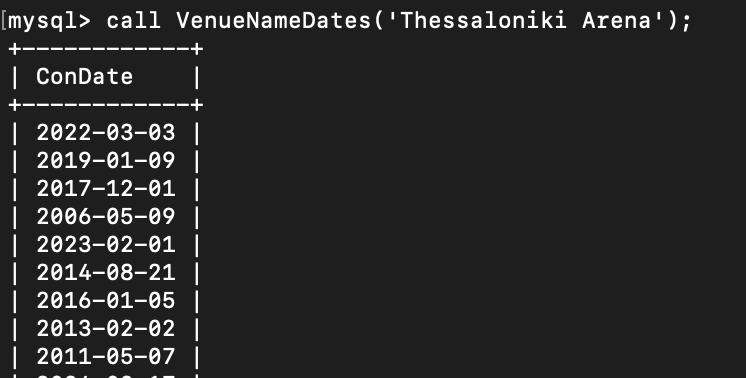
    SELECT ConDate FROM concerthistory WHERE VenueId = foundVenId;

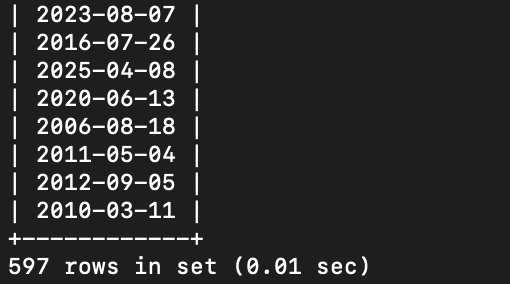
END $$

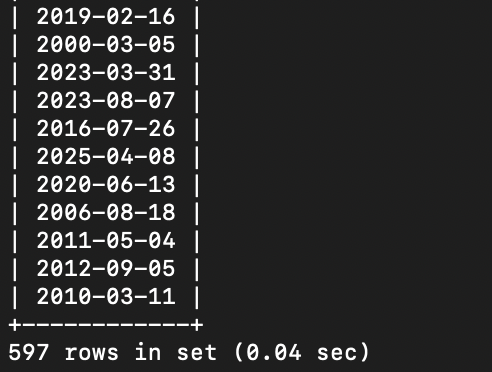
DELIMITER ;

### Screenshots Από Την Εκτέλεση

Με την εκτέλεση του VenueNameDates() με το index και χωρίς αυτό προκύπτουν και πάλι τα ίδια αποτελέσματα:







# **Κεφάλαιο 3: Δημιουργία Trigger**

## Trigger (3.1.4.1)

### Trigger Before Insert για τον πίνακα venues:

DELIMITER $$

CREATE TRIGGER insVen

AFTER INSERT

ON venues

FOR EACH ROW

BEGIN

    INSERT INTO log VALUES(USER(), NOW(), 'Insert', 'venues');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER updVen

AFTER UPDATE

ON venues

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Update', 'venues');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER delVen

AFTER DELETE

ON venues

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Delete', 'venues');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER insPers

AFTER INSERT

ON person

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Insert', 'person');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER updPers

AFTER UPDATE

ON person

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Update', 'person');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER delPers

AFTER DELETE

ON person

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Delete', 'person');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER insBand

AFTER INSERT

ON band

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Insert', 'band');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER updBand

AFTER UPDATE

ON band

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Update', 'band');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER delBand

AFTER DELETE

ON band

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Delete', 'band');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER insAlbum

AFTER INSERT

ON album

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Insert', 'album');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER updAlbum

AFTER UPDATE

ON album

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Update', 'album');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER delAlbum

AFTER DELETE

ON album

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Delete', 'album');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER insConc

AFTER INSERT

ON concert

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Insert', 'concert');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER updConc

AFTER UPDATE

ON concert

FOR EACH ROW

BEGIN

INSERT INTO log VALUES(USER(), NOW(), 'Update', 'concert');

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER delConc

AFTER DELETE

ON concert

FOR EACH ROW

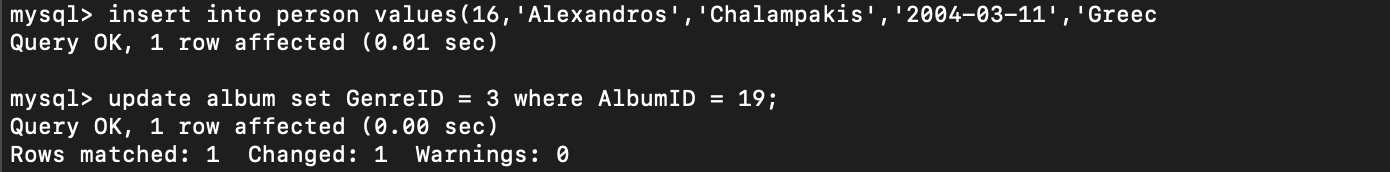
BEGIN

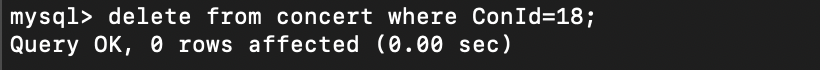
INSERT INTO log VALUES(USER(), NOW(), 'Delete', 'concert');

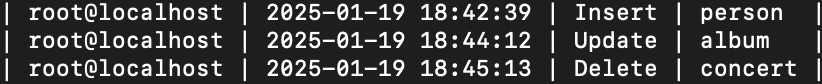
END $$

DELIMITER ;

### Screenshots Από Την Εκτέλεση







## Trigger (3.1.4.2)

DELIMITER $$

CREATE TRIGGER conScheduleCheck

BEFORE INSERT

ON concert

FOR EACH ROW

BEGIN

DECLARE scheduled\_count INT;

IF ABS(DATEDIFF(CURDATE(), NEW.ConDate)) < 5 THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'Concert must be scheduled at least 5 days before Concert Date!';

END IF;

SELECT COUNT(\*)

INTO scheduled\_count

FROM concert

WHERE ArtistId = NEW.ArtistId

AND Status = 'Scheduled';

IF scheduled\_count >= 3 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Artist already has 3 concerts scheduled!';

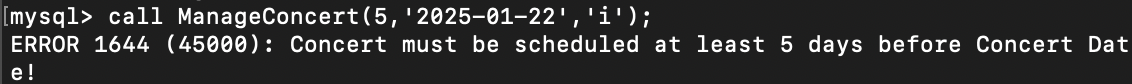
END IF;

END $$

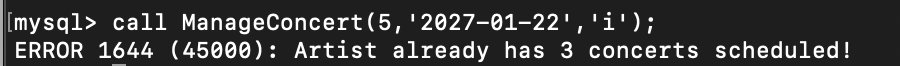
DELIMITER ;

### Screenshots Από Την Εκτέλεση

Αν ο προγραμματισμός της συναυλίας είναι σε λιγότερο από 5 ημέρες:



Αν ο καλλιτέχνης έχει ήδη 3 προγραμματισμένες συναυλίες:



## Trigger (3.1.4.3)

DELIMITER $$

CREATE TRIGGER conCancelCheck

BEFORE UPDATE

ON concert

FOR EACH ROW

BEGIN

IF (NEW.Status = 'Scheduled' AND OLD.Status = 'Cancelled') OR (NEW.Status = 'Cancelled' AND OLD.Status = 'Scheduled') THEN

IF DATEDIFF(NEW.ConDate, CURDATE()) <= 3 THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'You cannot cancel or reschedule a concert 3 days before concert date!';

END IF;

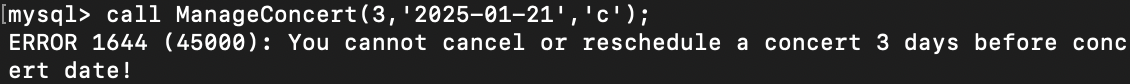
END IF;

END $$

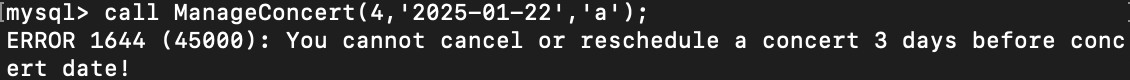
DELIMITER ;

### Screenshots Από Την Εκτέλεση

Για ακύρωση σε λιγότερο από 3 μέρες πριν την συναυλία:



Για επαναπρογραμματισμό μιας συναυλίας 3 μέρες πριν:



# **Κεφάλαιο 4: Υλοποίηση GUI**

#Αρχείο DBConnector

package com.mycompany.gui;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

private Connection conn;

// Method to establish the connection to the MySQL database

public Connection connect() {

try {

// Load the MySQL JDBC driver (make sure the connector JAR is included in your project)

Class.forName("com.mysql.cj.jdbc.Driver");

// Database URL, replace with your details

String url = "jdbc:mysql://localhost:3306/projectdb"; // Change projectdb to your database name

String user = "root"; // Your MySQL username

String password = "dbdb1234"; // Your MySQL password

// Establishing the connection

conn = DriverManager.getConnection(url, user, password);

System.out.println("Database connected!");

} catch (ClassNotFoundException | SQLException e) {

System.out.println("Error while connecting to database: " + e.getMessage());

}

return conn;

}

// Close the connection

public void close() {

try {

if (conn != null && !conn.isClosed()) {

conn.close();

System.out.println("Connection closed!");

}

} catch (SQLException e) {

System.out.println("Error while closing the connection: " + e.getMessage());

}

}

}

Παράδειγμα button1 για εμφάνιση album table:

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

System.out.println("Fetching data...");

String query = "SELECT \* FROM album"; // Replace 'your\_table\_name' with your table name

try {

// Create a Statement

Statement stmt = dbConnection.connect().createStatement();

// Execute the query and get the result set

ResultSet rs = stmt.executeQuery(query);

// Get column names to populate the table header

ResultSetMetaData metaData = rs.getMetaData();

int columnCount = metaData.getColumnCount();

// Create a table model to hold the data

DefaultTableModel model = new DefaultTableModel();

// Add columns to the model based on ResultSet metadata

for (int i = 1; i <= columnCount; i++) {

model.addColumn(metaData.getColumnName(i));

}

// Add rows to the model from the ResultSet

while (rs.next()) {

Object[] row = new Object[columnCount];

for (int i = 1; i <= columnCount; i++) {

row[i - 1] = rs.getObject(i);

}

model.addRow(row);

}

// Set the model to the table to display the data

table.setModel(model);

// Close the resources

rs.close();

stmt.close();

} catch (SQLException e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Error fetching data: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

}

}