Εργαστήριο Βάσεων Δεδομένων

Εργασία

2024 – 2025

Μέλη ομάδας:

Σπυρίδων Μανταδάκης *1100613* [*up1100613@ac.upatras.gr*](mailto:up1100613@ac.upatras.gr) *Φοιτητής 3ου έτους*

Παναγιώτης Παπανικολάου *1104804* [*up1104804@ac.upatras.gr*](mailto:up1104804@ac.upatras.gr) *Φοιτητής 3ου έτους*

Αλέξανδρος Γεώργιος Χαλαμπάκης *1100754* [*up1100754@ac.upatras.gr*](mailto:up1100754@ac.upatras.gr) *Φοιτητής 3ου έτους*

# **ΠΕΡΙΕΧΟΜΕΝΑ**

## **Μέρος Α:**

1. Περιγραφή Βάσης δεδομένων................................................. 3
2. Stored Procedures…………………………………………………………..18
3. Triggers…………………………………………………………………………. 34

**Μέρος Β:**

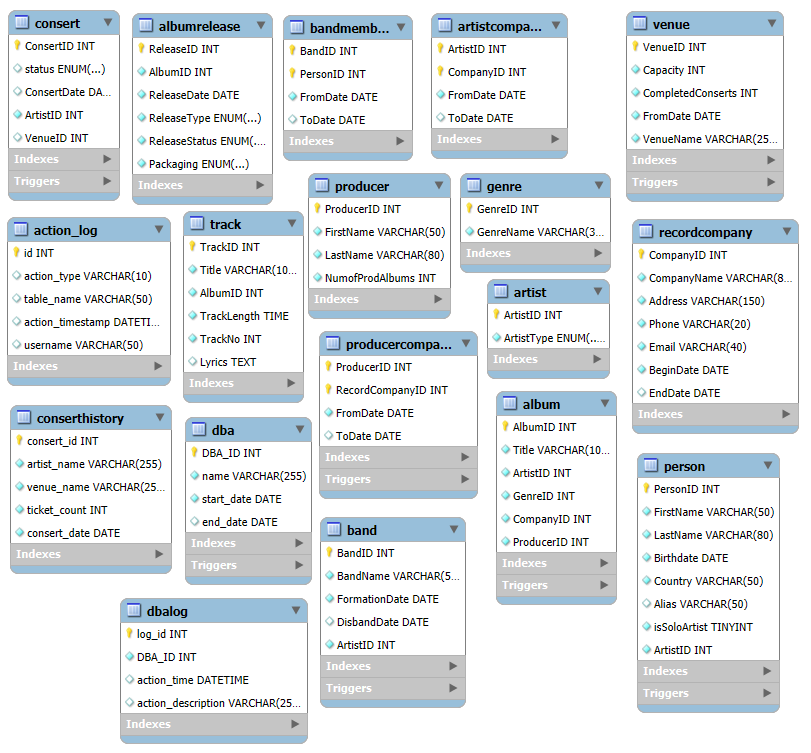
4. Περιγραφή λειτουργίας και κώδικα GUI……………………………….49

5. Πρόσθετη λειτουργικότητα..................................................... 57

# Περιγραφή Βάσης δεδομένων

Παρακάτω φαίνεται το σχεσιακό διάγραμμα της συνολοκής αναθεωρημένης Βάσης δεδομένων

με ολες τις επεκτάσεις που χρειαστηκαν να γίνουν κατα την υλοποίηση των ζητούμενων



Οι κύριες αλλαγές στο διάγραμμα είναι η προσθήκες νέων πινάκων consert και venue, αρχικά για την υλοποιήση των αρχικών απαιτήσεων, και action\_log, dba, conserthistory, dbalog είναι οι υπόλοιποι πίνακες που χρειάστηκαν για την ανάπτυξη του Α μέρους της εργασίας. Δεν έγινε κάποια αλλαγή στους πίνακες της προκατασκευαστικής φάσης

Ακολουθεί μια σύντομη περιγραφή των πινάκων της βάσης δεδομένων της προκατασκευαστικής φάσης:

1. **person**: Πληροφορίες ατόμων (καλλιτεχνών), με δυνατότητα να είναι solo καλλιτέχνες ή μέλη συγκροτημάτων.
2. **bandmember**: Συσχέτιση ατόμων με συγκροτήματα, με χρονικό διάστημα συμμετοχής.
3. **band**: Πληροφορίες για συγκροτήματα, όπως όνομα, ημερομηνία ίδρυσης/διάλυσης.
4. **genre**: Είδη μουσικής (π.χ., Rock, Pop).
5. **artist**: Καλλιτέχνες γενικά, με κατηγοριοποίηση (π.χ., άτομο, συγκρότημα).
6. **albumrelease**: Κυκλοφορίες άλμπουμ (μορφή, κατάσταση, συσκευασία).
7. **artistcompany**: Συνεργασίες καλλιτεχνών με δισκογραφικές εταιρείες.
8. **recordcompany**: Πληροφορίες δισκογραφικών εταιρειών (όνομα, στοιχεία επικοινωνίας).
9. **album**: Πληροφορίες άλμπουμ (τίτλος, καλλιτέχνης, είδος, παραγωγός).
10. **producercompany**: Συνεργασίες παραγωγών με δισκογραφικές εταιρείες.
11. **producer**: Πληροφορίες παραγωγών (όνομα, αριθμός άλμπουμ παραγωγής).
12. **track**: Πληροφορίες τραγουδιών (τίτλος, διάρκεια, αριθμός κομματιού σε άλμπουμ).

**Αναλυτική περιγραφή των νέων πινάκων και των πεδίων τους:**

**1. consert (Πίνακας Συναυλιών)**

Περιγράφει τις συναυλίες που έχουν προγραμματιστεί ή έχουν πραγματοποιηθεί.

* **ConsertID**: Μοναδικό αναγνωριστικό για κάθε συναυλία (Πρωτεύον Κλειδί).
* **status**: Κατάσταση της συναυλίας (π.χ., Scheduled, Completed, Cancelled).
* **ConsertDate**: Ημερομηνία διεξαγωγής της συναυλίας.
* **ArtistID**: Αναφορά στον καλλιτέχνη που συμμετέχει (Ξένο Κλειδί που συνδέεται με τον πίνακα artist).
* **VenueID**: Αναφορά στον χώρο όπου λαμβάνει χώρα η συναυλία (Ξένο Κλειδί που συνδέεται με τον πίνακα venue).

**2. venue (Πίνακας Χώρων Συναυλιών)**

Καταγράφει πληροφορίες για τους χώρους όπου διεξάγονται συναυλίες.

* **VenueID**: Μοναδικό αναγνωριστικό για κάθε χώρο (Πρωτεύον Κλειδί).
* **VenueName**: Όνομα του χώρου (υποχρεωτικό).
* **Capacity**: Χωρητικότητα του χώρου σε αριθμό θεατών.
* **CompletedConserts**: Αριθμός συναυλιών που έχουν ολοκληρωθεί στον συγκεκριμένο χώρο.
* **FromDate**: Ημερομηνία έναρξης λειτουργίας του χώρου.

**3. consertHistory (Ιστορικό Συναυλιών)**

Καταγράφει πληροφορίες για παρελθοντικές συναυλίες.

* **consert\_id**: Μοναδικό αναγνωριστικό της συναυλίας (Πρωτεύον Κλειδί).
* **artist\_name**: Όνομα του καλλιτέχνη που συμμετείχε στη συναυλία.
* **venue\_name**: Όνομα του χώρου όπου πραγματοποιήθηκε η συναυλία.
* **ticket\_count**: Αριθμός εισιτηρίων που πουλήθηκαν για τη συναυλία.
* **consert\_date**: Ημερομηνία διεξαγωγής της συναυλίας.

**4. DBA (Πίνακας Διαχειριστών της Βάσης Δεδομένων)**

Καταγράφει πληροφορίες για τους διαχειριστές της βάσης δεδομένων.

* **DBA\_ID**: Μοναδικό αναγνωριστικό για κάθε διαχειριστή (Πρωτεύον Κλειδί).
* **name**: Όνομα του διαχειριστή.
* **start\_date**: Ημερομηνία έναρξης των καθηκόντων του διαχειριστή.
* **end\_date**: Ημερομηνία λήξης των καθηκόντων του διαχειριστή (αν υπάρχει).

**5. DBALog (Καταγραφή Ενεργειών Διαχειριστών)**

Καταγράφει τις ενέργειες που πραγματοποιούν οι διαχειριστές στη βάση δεδομένων.

* **log\_id**: Μοναδικό αναγνωριστικό για κάθε καταγραφή (Πρωτεύον Κλειδί).
* **DBA\_ID**: Αναφορά στον διαχειριστή που πραγματοποίησε την ενέργεια (Ξένο Κλειδί που συνδέεται με τον πίνακα DBA).
* **action\_time**: Χρονική στιγμή που πραγματοποιήθηκε η ενέργεια (προκαθορισμένη τιμή το τρέχον timestamp).
* **action\_description**: Περιγραφή της ενέργειας (π.χ., "Διαγραφή εγγραφής", "Ενημέρωση δεδομένων").

**6. action\_log (Γενική Καταγραφή Ενεργειών)**

Καταγράφει ενέργειες χρηστών ή διαχειριστών στη βάση δεδομένων.

* **id**: Μοναδικό αναγνωριστικό για κάθε καταγραφή (Πρωτεύον Κλειδί).
* **action\_type**: Τύπος ενέργειας (π.χ., INSERT, UPDATE, DELETE).
* **table\_name**: Όνομα του πίνακα όπου πραγματοποιήθηκε η ενέργεια.
* **action\_timestamp**: Χρονική στιγμή της ενέργειας.
* **username**: Όνομα χρήστη που εκτέλεσε την ενέργεια.

Εντολές sql για τη δημιουργία της ΒΔ:

Πίνακες :

create table person(

PersonID int(11) not null auto\_increment,

FirstName varchar(50) not null,

LastName varchar(80) not null,

Birthdate date not null,

Country varchar(50) not null,

Alias varchar(50),

isSoloArtist tinyint(4) not null,

ArtistID int(11) not null,

primary key(PersonID),

constraint isartist

foreign key(ArtistID) references artist(ArtistID)

on delete cascade on update cascade);

create table bandmember(

BandID int(11) not null,

PersonID int(11) not null,

FromDate date not null,

ToDate date,

primary key(BandID,PersonID),

unique(FromDate),

constraint ismember

foreign key(PersonID) references person(PersonID)

on delete cascade on update cascade,

constraint fromband

foreign key(BandID) references band(BandID)

on delete cascade on update cascade);

create table band(

BandID int(11) not null auto\_increment,

BandName varchar(50) not null,

FormationDate date not null,

DisbandDate date,

ArtistID int(11) not null,

primary key(BandID),

constraint isartist

foreign key(ArtistID) references artist(ArtistID)

on delete cascade on update cascade);

create table genre(

GenreID int(11) not null auto\_increment,

GenreName varchar(30) not null,

primary key(GenreID));

create table artist(

ArtistID int(11) not null auto\_increment,

ArtistType enum('PERSON','BAND','ORCHESTRA','CHOIR','VIRTUAL','OTHER') not null,

primary key(ArtistID));

create table albumrelease(

ReleaseID int(11) not null auto\_increment,

AlbumID int(11) not null,

ReleaseDate date not null,

ReleaseType enum('LP','CD','MP3') not null,

ReleaseStatus enum('OFFICIAL','PROMOTION','BOOTLEG','WITHDRAWN','CANCELED') not null,

Packaging enum ('BOOK','CARDBOARD SLEEVE','DIGIPAK','JEWEL CASE','NA') not null,

primary key(ReleaseID),

constraint isalbum

foreign key(AlbumID) references album(AlbumID)

on delete cascade on update cascade);

create table artistcompany(

ArtistID int(11) not null,

CompanyID int(11) not null,

FromDate date not null,

ToDate date,

primary key(ArtistID,CompanyID),

unique(FromDate),

constraint isartist

foreign key(ArtistID) references artist(ArtistID)

on delete cascade on update cascade,

constraint iscompany

foreign key(CompanyID) references recordcompany(CompanyID)

on delete cascade on update cascade);

create table recordcompany(

CompanyID int(11) not null auto\_increment,

CompanyName varchar(80) not null,

Address varchar(150) not null,

Phone varchar(20) not null,

Email varchar(40) not null,

BeginDate date not null,

EndDate date,

primary key(CompanyID));

create table album(

AlbumID int(11) not null auto\_increment,

Title varchar(100) not null,

ArtistID int(11) not null,

GenreID int(11) not null,

CompanyID int(11) not null,

ProducerID int(11) not null,

primary key(AlbumID),

unique(Title),

constraint fromartist

foreign key(ArtistID) references artist(ArtistID)

on delete cascade on update cascade,

constraint isgenre

foreign key(GenreID) references genre(GenreID)

on delete cascade on update cascade,

constraint isartist

foreign key(CompanyID) references recordcompany(CompanyID)

on delete cascade on update cascade,

constraint produced

foreign key(ProducerID) references producer(ProducerID)

on delete cascade on update cascade);

create table producercompany(

ProducerID int(11) not null,

RecordCompanyID int(11) not null,

FromDate date not null,

ToDate date,

primary key(ProducerID,RecordCompanyID),

unique(FromDate),

constraint isproducer

foreign key(ProducerID) references producer(ProducerID)

on delete cascade on update cascade,

constraint reccompany

foreign key(RecordCompanyID) references recordcompany(RecordCompanyID)

on delete cascade on update cascade);

create table producer(

ProducerID int(11) not null auto\_increment,

FirstName varchar(50) not null,

LastName varchar(80) not null,

NumofProdAlbums int(11) not null,

primary key(ProducerID));

create table track(

TrackID int(11) not null auto\_increment,

Title varchar(100) not null,

AlbumID int(11) not null,

TrackLength time not null,

TrackNo int(11) not null,

Lyrics text,

primary key(TrackID),

constraint fromalbum

foreign key(AlbumID) references album(AlbumID)

on delete cascade on update cascade);

create table consert(

ConsertID int(11) not null auto\_increment,

status Enum('Scheduled','Completed','Cancelled') default 'scheduled',

ConsertDate date,

ArtistID int(11) not null,

VenueID int(11) not null,

primary key(ConsertID),

constraint Artisthasconsert

foreign key(ArtistID) references artist(ArtistID)

on delete cascade on update cascade,

constraint Takesplace

foreign key(VenueID) references venue(VenueID)

on delete cascade on update cascade

);

create table venue(

VenueID int(11) not null auto\_increment,

VenueName varchar not null,

Capacity int not null,

CompletedConserts int not null,

FromDate date,

primary key(VenueID)

);

CREATE TABLE consertHistory (

consert\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

artist\_name VARCHAR(255) NOT NULL,

venue\_name VARCHAR(255) NOT NULL,

ticket\_count INT NOT NULL,

consert\_date DATE NOT NULL

);

CREATE TABLE DBA (

DBA\_ID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE NULL

);

CREATE TABLE DBALog (

log\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

DBA\_ID INT NOT NULL,

action\_time DATETIME DEFAULT CURRENT\_TIMESTAMP,

action\_description VARCHAR(255),

FOREIGN KEY (DBA\_ID) REFERENCES DBA(DBA\_ID)

);

CREATE TABLE action\_log (

id INT PRIMARY KEY AUTO\_INCREMENT,

action\_type VARCHAR(10),

table\_name VARCHAR(50),

action\_timestamp DATETIME,

username VARCHAR(50)

);

Εντολές insert:

Σημείωση: οι εγγραφές των πινάκων ενδέχεται να μην είναι όλες ίδιες καθώς χρειάστηκε να γίνουν προσθήκες και τροποποιήσεις στους πίνακες για τον έλεγχο λειτουργίας των υπόλοιπων ζητούμενων.

insert into artist(ArtistType) values

('PERSON'),

('PERSON'),

('BAND'),

('CHOIR'),

('ORCHESTRA'),

('OTHER'),

('PERSON'),

('BAND'),

('BAND'),

('VIRTUAL');

insert into person(FirstName,LastName,BirthDate,Country,Alias,isSoloArtist,ArtistID) values

('James','Woods','1995-07-24','USA',null,1,1),

('Michael','Stevens','1980-11-05','Canada','Macklemore',1,2),

('Elizabeth','Owens','1983-04-14','UK',null,0,3),

('Howard','Nelson','1979-03-14','Mexico',null,0,3),

('David','Ackerland','1995-12-30','Belgium',null,1,7),

('Maria','Fockel','2000-09-28','France','Rosewalk',1,5),

('Anna','Larston','1959-03-31','Dennmark',null,0,8);

insert into recordcompany values

(null,'StarRecords','St. Birkins','424536546', 'starrrecords@gmail.com','1970-12-02',null),

(null,'PanikRecords','Vouliagmenis 77','2108943935','info@panikrecords.gr','2011-09-28',null),

(null,'Sony Music Entertainment','Mesogeion 311','2106722150','reception.enquiries@sonymusic.gr','1929-09-09',null),

(null,'SoundWave Records','123 Melody Lane','3105551234','info@soundwave.com','1995-06-15',null),

(null,'Harmony Studios','456 Rhythm Ave','6155555678','contact@harmonystudios.com','1987-03-10',null);

insert into producer values

(null,'Sophia', 'Taylor', 10),

(null,'Liam', 'Walker', 35),

(null,'Isabella', 'Clark', 16),

(null,'Ethan', 'Allen', 21),

(null,'Mia', 'Hernandez', 14),

(null,'Noah', 'White', 40);

insert into band values

(null,'Neon Horizon', '2012-05-14',null,3),

(null,'Twilight Rhythm', '1985-09-20',null,8),

(null,'Echoes of Eternity', '2007-02-28',null,9);

insert into bandmember values

(3,3,'2015-05-14',null),

(1,4,'2018-08-17',null),

(2,7,'2007-02-28',null),

(2,4,'1985-09-20',null);

insert into genre values

(null,'rock'),

(null,'jazz'),

(null,'pop'),

(null,'metal'),

(null,'rock'),

(null,'pop');

insert into artistcompany values

(1,1,"1990-04-23",null),

(2,1,"2000-03-14",null),

(3,3,"1999-11-30",null),

(7,4,"2010-08-10",null),

(8,5,"2015-12-01",null),

(9,2,"2016-03-09",null),

(1,4,"1998-12-08",null),

(2,3,"2022-01-23",null),

(7,2,"2004-07-17",null),

(8,4,"2017-04-18",null),

(9,5,"2019-02-20",null);

insert into producercompany values

(1,5,'1987-03-10',null),

(2,4,'1995-06-15',null),

(3,3,'1929-09-09',null),

(4,2,'2011-09-28',null),

(5,1,'1970-12-02',null),

(3,2,'1938-04-13',null),

(1,4,'2000-01-19',null),

(2,1,'2010-11-30',null);

insert into album values

(null,'Days Together',1,3,1,1),

(null,'Generations',2,5,4,2),

(null,'In the Sun',3,2,5,3),

(null,'Better Together',7,6,3,5),

(null,'Wanderlust Chronicles',8,1,4,3),

(null,'Over the Years',1,2,3,4),

(null,'This is us',8,6,2,6),

(null,'Smiling Ocean',9,6,1,2),

(null,'Teen Pain',1,1,2,4),

(null,'think about Promise',2,6,3,4);

insert into track values

(null,'All in Moon',1,'00:05:23',1,null),

(null,'The Unbeat',1,'00:05:20',2,null),

(null,'First Wave',1,'00:04:53',3,null),

(null,'Imagine Scars',1,'00:04:00',4,null),

(null,'Soft Horizon',1,'00:05:20',5,null),

(null,'The Passion',1,'00:04:28',6,null),

(null,'Inspired Summer',1,'00:05:21',7,null),

(null,'Running Out of Time',1,'00:05:22',8,null),

(null,'Soul of Miles',2,'00:05:23',1,null),

(null,'Catle of Secrets',2,'00:04:03',2,null),

(null,'Cheery Lover',2,'00:05:40',3,null),

(null,'A Time of Today',2,'00:03:43',4,null),

(null,'Hot Touch',2,'00:04:50',5,null),

(null,'Sky of Blue',2,'00:05:32',6,null),

(null,'Foul Soul',2,'00:04:18',7,null),

(null,'Earning Break',2,'00:03:57',8,null),

(null,'Story of my Life',3,'00:05:15',1,null),

(null,'Dreamy Future',3,'00:03:50',2,null),

(null,'Break of Dawn',3,'00:04:13',3,null),

(null,'Hold the Line',3,'00:05:33',4,null),

(null,'Late Club',3,'00:02:59',5,null),

(null,'Stairway to Heaven',3,'00:04:10',6,null),

(null,'Thank you',3,'00:03:57',7,null),

(null,'Tortured Soul',3,'00:05:27',8,null),

(null,'Light Breeze',3,'00:03:45',9,null);

insert into albumrelease values

(null,1,'2000-03-13','CD','OFFICIAL','BOOK'),

(null,2,'2005-01-23','MP3','PROMOTION','CARDBOARD SLEEVE'),

(null,3,'2013-07-30','LP','OFFICIAL','BOOK'),

(null,1,'2016-03-31','CD','PROMOTION','BOOK'),

(null,2,'1998-03-13','MP3','BOOTLEG','NA'),

(null,3,'1996-11-25','CD','WITHDRAWN','DIGIPACK'),

(null,1,'2004-07-17','LP','CANCELED','BOOK'),

(null,2,'1980-03-23','CD','OFFICIAL','BOOK'),

(null,3,'2014-05-13','MP3','OFFICIAL','NA'),

(null,1,'1997-02-25','LP','PROMOTION','CARDBOARD SLEEVE'),

(null,2,'2019-01-29','MP3','OFFICIAL','BOOK'),

(null,1,'2001-01-31','CD','BOOTLEG','NA');

INSERT INTO DBA (name, start\_date) VALUES

('John Doe', '2023-01-01'),

('Jane Smith', '2023-03-15'),

('Alice Johnson', '2023-06-01');

Trigger για απελευθέρωση Venue Οταν consert ολοκληρώνεται:

DROP TRIGGER IF EXISTS ReleaseVenueOnCompletion;

DELIMITER $

CREATE TRIGGER ReleaseVenueOnCompletion

BEFORE UPDATE ON consert

FOR EACH ROW

BEGIN

IF NEW.status = 'Completed' AND OLD.status != 'Completed' THEN

SET NEW.VenueID = NULL;

END IF;

END$

DELIMITER ;

# 2. Stored Procedures

**3.1.3.1**

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση της stored procedure υπολογισμού βαθμού του venue.

DROP PROCEDURE IF EXISTS getscore;

DELIMITER $

CREATE PROCEDURE getscore(IN venID INT, OUT res INT)

BEGIN

DECLARE capacity INT;

DECLARE compl INT;

DECLARE years\_active INT;

SELECT venue.Capacity, venue.CompletedConserts, TIMESTAMPDIFF(YEAR, FromDate, CURDATE())

INTO capacity, compl, years\_active

FROM venue

WHERE VenueID = venID;

SET res = ((capacity / 1000) \* 1) + ((compl / 100) \* 3) + (years\_active \* 2);

END $

DELIMITER ;

**Παράδειγμα λειτουργίας του stored procedure:**

**A screen shot of a computer

Description automatically generated**

αν εκτελέσω τον κώδικα SQL :



Τότε συμφωνα με τον Πίνακα 2 της Απαίτησης 3.1.2.2. ο βαθμός θα ισούται με: 5\*1 + 3\*5 + 12\*2 = 44

A screen shot of a computer program

Description automatically generated

**3.1.3.2**

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση της stored procedure ρύθμισης συναυλίας

DROP PROCEDURE IF EXISTS PlanConsert;

DELIMITER $

CREATE PROCEDURE PlanConsert(IN artistID INT, IN Date DATE, IN choice CHAR)

BEGIN

IF choice = 'i' THEN

IF (SELECT COUNT(\*) FROM consert WHERE status = 'Scheduled' AND consert.ArtistID = artistID) >= 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Artist already has scheduled consert';

ELSEIF (SELECT COUNT(\*) FROM consert WHERE status = 'Cancelled' AND consert.ArtistID = artistID) >= 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Artist already has cancelled consert';

ELSE

INSERT INTO consert(status, ConsertDate, ArtistID) VALUES ('Scheduled', Date, artistID);

SELECT 'Consert scheduled' AS Message;

END IF;

ELSEIF choice = 'c' THEN

IF (SELECT COUNT(\*) FROM consert WHERE status = 'Scheduled' AND consert.ArtistID = artistID AND consertDate = Date) >= 1 THEN

UPDATE consert SET status = 'Cancelled', VenueID = null WHERE status = 'Scheduled' AND consert.ArtistID = artistID AND consertDate = Date;

SELECT 'Consert cancelled' AS Message;

ELSEIF (SELECT COUNT(\*) FROM consert WHERE status = 'Cancelled' AND consert.ArtistID = artistID AND consertDate = Date) >= 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Consert already cancelled';

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Scheduled consert does not exist';

END IF;

ELSEIF choice = 'a' THEN

IF (SELECT COUNT(\*) FROM consert WHERE status = 'Cancelled' AND consert.ArtistID = artistID AND consertDate = Date) >= 1 THEN

UPDATE consert SET status = 'Scheduled' WHERE status = 'Cancelled' AND consert.ArtistID = artistID AND consertDate = Date;

SELECT 'Cancelled Consert set to scheduled' AS Message;

ELSEIF (SELECT COUNT(\*) FROM consert WHERE status = 'Scheduled' AND consert.ArtistID = artistID AND consertDate = Date) >= 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Consert already scheduled';

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cancelled consert does not exist';

END IF;

END IF;

END $

DELIMITER ;

**Παραδείγματα λειτουργίας του stored procedure:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**3.1.3.3**

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση της stored procedure εύρεσης venue για consert

DROP PROCEDURE IF EXISTS GetVenueData;

DELIMITER $

CREATE PROCEDURE GetVenueData(IN conID INT, IN req\_cap INT, OUT ven\_id INT, OUT ven\_cap INT)

BEGIN

DECLARE venueExists INT;

DECLARE highestScore INT DEFAULT 0;

DECLARE bestVenID INT DEFAULT NULL;

DECLARE bestVenCap INT DEFAULT 0;

DECLARE venueScore INT;

DECLARE ven\_id\_temp INT;

DECLARE ven\_cap\_temp INT;

DECLARE done INT DEFAULT 0;

DECLARE foundVenues CURSOR FOR

SELECT venue.VenueID, venue.Capacity

FROM venue

WHERE venue.Capacity >= req\_cap \* 1.10;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

IF (SELECT COUNT(\*) FROM consert WHERE consert.ConsertID = conID AND consert.status = 'Cancelled') > 0 OR

(SELECT COUNT(\*) FROM consert WHERE consert.ConsertID = conID) < 1 THEN

SET ven\_id = NULL;

SET ven\_cap = 0;

ELSE

SELECT COUNT(\*) INTO venueExists

FROM consert

WHERE consert.ConsertID = conID AND consert.VenueID IS NOT NULL AND consert.status != 'Cancelled';

IF venueExists = 1 THEN

SELECT venue.VenueID, venue.Capacity

INTO ven\_id, ven\_cap

FROM consert

INNER JOIN venue ON consert.VenueID = venue.VenueID

WHERE consert.ConsertID = conID;

ELSE

OPEN foundVenues;

FETCH foundVenues INTO ven\_id\_temp, ven\_cap\_temp;

WHILE done = 0 DO

CALL getscore(ven\_id\_temp, venueScore);

IF venueScore > highestScore THEN

SET highestScore = venueScore;

SET bestVenID = ven\_id\_temp;

SET bestVenCap = ven\_cap\_temp;

END IF;

FETCH foundVenues INTO ven\_id\_temp, ven\_cap\_temp;

END WHILE;

CLOSE foundVenues;

SET ven\_id = bestVenID;

SET ven\_cap = bestVenCap;

END IF;

END IF;

SELECT ven\_id AS Venue\_ID, ven\_cap AS Venue\_Capacity;

END $

DELIMITER ;

**Παραδείγματα λειτουργίας του stored procedure:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**3.1.3.4**

Κώδικας SQL για την δημιουργία indexes και stored procedure για την δημιουργία εγγραφών για ConsertHistory

CREATE INDEX idx\_tickets\_count ON consertHistory(ticket\_count);

CREATE INDEX idx\_venue\_name ON consertHistory(venue\_name);

DELIMITER $

DROP PROCEDURE IF EXISTS PopulateconsertHistory;

CREATE PROCEDURE PopulateconsertHistory()

BEGIN

DECLARE i INT DEFAULT 1;

WHILE i <= 60000 DO

INSERT INTO consertHistory (artist\_name, venue\_name, ticket\_count, consert\_date)

VALUES (

CONCAT('Artist ', i % 100),

CONCAT('Venue ', i % 50),

FLOOR(1000 + RAND() \* 9000),

DATE\_ADD('2020-01-01', INTERVAL i DAY)

);

SET i = i + 1;

END WHILE;

END$

DELIMITER ;

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση της stored procedure εύρεσης καλλιτεχνών με πλήθος εισητηρίων μεταξύ των 2 ορισμάτων

DELIMITER $

DROP PROCEDURE IF EXISTS GetArtistsByTicketRange;

CREATE PROCEDURE GetArtistsByTicketRange(IN min\_tickets INT, IN max\_tickets INT)

BEGIN

SELECT DISTINCT artist\_name

FROM consertHistory

WHERE ticket\_count BETWEEN min\_tickets AND max\_tickets;

END$

DELIMITER ;

**Παραδείγματα λειτουργίας του stored procedure:**

Για να τρέξω το stored procedure και το index στο διάστημα (2000, 3000), εκτελώ τον κώδικα:

SET profiling = 1;

CALL GetArtistsByTicketRange(2000, 3000);

SHOW PROFILE FOR QUERY 1;

SET profiling = 0;

A screenshot of a computer

Description automatically generated A screen shot of a black background with white text

Description automatically generated A screenshot of a computer

Description automatically generated

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση της stored procedure εμφάνισης ημερομηνιών διεξαγωγής venue.

DELIMITER $

DROP PROCEDURE IF EXISTS GetconsertDatesByVenue;

CREATE PROCEDURE GetconsertDatesByVenue(IN venue VARCHAR(255))

BEGIN

SELECT consert\_date

FROM consertHistory

WHERE venue\_name = venue;

END$

DELIMITER ;

**Παραδείγματα λειτουργίας του stored procedure:**

Για να τρέξω το stored procedure και το index για την εμφάνιση ημερομηνίων διεξαγωγης του venue με όνομα “Venue 4”, εκτελώ τις εντολές:

SET profiling = 1;

CALL GetconsertDatesByVenue('Venue 4');

SHOW PROFILE FOR QUERY 1;

SET profiling = 0;

………..

A black background with white numbers and numbers

Description automatically generatedA screenshot of a computer

Description automatically generated

…………

# 3. Triggers

**3.1.4.1**

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση των triggers ενημέρωσης action\_log, πάνω στους πίνακες: **person, band, album, concert, venue.**

**PERSON:**

DROP TRIGGER IF EXISTS person\_insert;

DELIMITER $

CREATE TRIGGER person\_insert AFTER INSERT ON person

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('INSERT', 'person', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS person\_update;

DELIMITER $

CREATE TRIGGER person\_update AFTER UPDATE ON person

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('UPDATE', 'person', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS person\_delete;

DELIMITER $

CREATE TRIGGER person\_delete AFTER DELETE ON person

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('DELETE', 'person', NOW(), USER());

END$

DELIMITER ;

**BAND:**

DROP TRIGGER IF EXISTS band\_insert;

DELIMITER $

CREATE TRIGGER band\_insert AFTER INSERT ON band

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('INSERT', 'band', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS band\_update;

DELIMITER $

CREATE TRIGGER band\_update AFTER UPDATE ON band

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('UPDATE', 'band', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS band\_delete;

DELIMITER $

CREATE TRIGGER band\_delete AFTER DELETE ON band

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('DELETE', 'band', NOW(), USER());

END$

DELIMITER ;

**ALBUM:**

DROP TRIGGER IF EXISTS album\_insert;

DELIMITER $

CREATE TRIGGER album\_insert AFTER INSERT ON album

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('INSERT', 'album', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS album\_update;

DELIMITER $

CREATE TRIGGER album\_update AFTER UPDATE ON album

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('UPDATE', 'album', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS album\_delete;

DELIMITER $

CREATE TRIGGER album\_delete AFTER DELETE ON album

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('DELETE', 'album', NOW(), USER());

END$

DELIMITER ;

**CONSERT:**

DROP TRIGGER IF EXISTS consert\_insert;

DELIMITER $

CREATE TRIGGER consert\_insert AFTER INSERT ON consert

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('INSERT', 'consert', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS consert\_update;

DELIMITER $

CREATE TRIGGER consert\_update AFTER UPDATE ON consert

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('UPDATE', 'consert', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS consert\_delete;

DELIMITER $

CREATE TRIGGER consert\_delete AFTER DELETE ON consert

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('DELETE', 'consert', NOW(), USER());

END$

DELIMITER ;

**VENUE:**

DROP TRIGGER IF EXISTS venue\_insert;

DELIMITER $

CREATE TRIGGER venue\_insert AFTER INSERT ON venue

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('INSERT', 'venue', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS venue\_update;

DELIMITER $

CREATE TRIGGER venue\_update AFTER UPDATE ON venue

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('UPDATE', 'venue', NOW(), USER());

END$

DELIMITER ;

DROP TRIGGER IF EXISTS venue\_delete;

DELIMITER $

CREATE TRIGGER venue\_delete AFTER DELETE ON venue

FOR EACH ROW

BEGIN

INSERT INTO action\_log (action\_type, table\_name, action\_timestamp, username)

VALUES ('DELETE', 'venue', NOW(), USER());

END$

DELIMITER ;

**Παραδείγματα λειτουργίας των Triggers:**

**Venue**

Πριν την προσθήκη νέας εγγραφής στον πίνακα venue:

A screenshot of a computer

Description automatically generated

Eκτελώ την εντολή insert στον πίνακα venue:



A screen shot of a computer

Description automatically generated

Μετά την προσθήκη νέας εγγραφής στον πίνακα venue:

A screenshot of a computer screen

Description automatically generated

**Consert:**

Πριν την τροποποίηση εγγραφής στον πίνακα consert:

A screenshot of a computer screen

Description automatically generated

Eκτελώ την εντολή update στον πίνακα consert:



A computer screen with white text

Description automatically generated

Μετά την τροποποίηση εγγραφής στον πίνακα consert:

A screenshot of a computer

Description automatically generated

**3.1.4.2**

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση του trigger ελέγχου προγραμματισμού συναυλίας πάνω στον πίνακα consert.

DROP TRIGGER IF EXISTS CheckConsertdata;

DELIMITER $

CREATE TRIGGER CheckConsertdata BEFORE INSERT ON consert

FOR EACH ROW

BEGIN

IF new.ConsertDate < DATE\_ADD(CURDATE(), INTERVAL 5 DAY) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Consert must be scheduled at least 5 days before the consert';

END IF;

IF (SELECT COUNT(\*) FROM consert WHERE status = 'Scheduled' AND ArtistID = new.ArtistID) >= 3 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Artist can have at most 3 scheduled conserts';

END IF;

END $

DELIMITER ;

**Παραδείγματα λειτουργίας του Trigger:**

Έλεγχος ημερομηνίας διεξαγωγής:

Προσθέτω νέα εγγραφή με ημερομηνία διεξαγωγής : '2025-01-11' που απέχει 3 μέρες απο την ημερομηνία προγραμματισμου της: '2025-01-08'

Ο ελεγχός λειτουργεί ορθά καθώς εμφανίζεται το μήνυμα σφάλματος.

**A screen shot of a computer

Description automatically generated**

Έλεγχος αριθμού προγραμματισμένων συναυλιών καλλιτέχνη:

Ο τρέχον πίνακας περιέχει 3 εγγραφές με ίδιο ArtistID.

Η προσθήκη νέας εγγραφής με ίδιο ArtistID πρέπει να εμφανίσει σφάλμα.

A screen shot of a computer

Description automatically generated

A black background with white text

Description automatically generated

Ο ελεγχός λειτουργεί ορθά καθώς εμφανίζεται το μήνυμα σφάλματος.

**3.1.4.3**

Παρακάτω είναι ο κώδικας SQL για την υλοποίηση του trigger ελέγχου προγραμματισμού συναυλίας πάνω στον πίνακα consert.

DROP TRIGGER IF EXISTS CheckCancellation;

DELIMITER $

CREATE TRIGGER CheckCancellation BEFORE UPDATE ON consert

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM consert WHERE new.status = 'Scheduled' AND old.status = 'Cancelled' AND ArtistID = new.ArtistID) >= 3 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Artist can have at most 3 scheduled conserts';

END IF;

IF new.status = 'Cancelled' AND old.status = 'Scheduled' AND new.ConsertDate < DATE\_ADD(CURDATE(), INTERVAL 3 DAY) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Consert cannot be cancelled';

END IF;

END $

DELIMITER ;

**Παραδείγματα λειτουργίας του Trigger:**

Έλεγχος ημερομηνίας διεξαγωγής για ακύρωση:

Ο τρέχον πίνακας περιέχει την εγγραφή με ημερομηνία διεξαγωγής: '2025-01-10' .

**A screenshot of a computer program

Description automatically generated**

Η προσπάθεια ακύρωσης της συναυλιας εμφανίζει σφάλμα καθως η ημερομηνία διεξαγωγής απέχει λιγότερο από 3 μέρες απο την ημερομηνία της ακύρωσης.

Έλεγχος αριθμού προγραμματισμένων συναυλιών καλλιτέχνη:

Ο τρέχον πίνακας περιέχει 4 εγγραφές με ίδιο ArtistID, 3 με status Scheduled και 1 με status Cancelled.

A screen shot of a computer program

Description automatically generated

Η προσπάθεια επαναπρογραμματισμού της συναυλίας με ConsertID = 2, εμφανίζει σφάλμα εφόσον υπάρχουν ήδη 3 προγραμματισμένες συναυλίες με ArtistID = 5.

# 4. Περιγραφή λειτουργίας και κώδικα GUI

**1. Τεκμηρίωση του Κώδικα**

**Σημείωση:**

Έγινε αρκετή προσπάθεια να διορθωθεί αλλά δεν καταφέραμε να τροπόποιησουμε την εμφάνιση των περιεγχομένων. Ορισμένοι πίνακες ενδέχεται να εμφανίζονται στο text area του MainGUI ανομοιόμορφα και πιθανόν να υπάρχει δυσκολία στην αναγνωσιμότητα.

**Γενική Επισκόπηση**

Το πακέτο com.mycompany.gui περιλαμβάνει το πρόγαμμα PROJECT, το main GUI και το popGUI. Το PROJECT παρέχει μεθόδους για την αλληλεπίδραση με τη βάση δεδομένων recordcompany. Οι μέθοδοι χρησιμοποιούν JDBC για τη διαχείριση δεδομένων (ανάκτηση, εισαγωγή, ενημέρωση, διαγραφή).

**Περιγραφή των Μεθόδων**

* **DisplayTable(String tableName)**
  + Σκοπός: Ανάκτηση δεδομένων από τον πίνακα tableName της βάσης δεδομένων.
  + Υλοποίηση: Συνδέεται στη βάση, εκτελεί SQL SELECT και εμφανίζει τα δεδομένα στο jTextArea2 του GUI.
* **InsertData(String input)**
  + Σκοπός: Εκτέλεση εντολών SQL INSERT.
  + Υλοποίηση: Εκτελεί SQL μέσω της Statement.executeUpdate() και επιστρέφει τον αριθμό των εγγραφών που εισήχθησαν.
* **UpdateData(String input)**
  + Σκοπός: Εκτέλεση εντολών SQL UPDATE.
  + Υλοποίηση: Εκτελεί SQL και επιστρέφει τον αριθμό των εγγραφών που τροποποιήθηκαν.
* **DeleteData(String input)**
  + Σκοπός: Εκτέλεση εντολών SQL DELETE.
  + Υλοποίηση: Εκτελεί SQL και επιστρέφει τον αριθμό των εγγραφών που διαγράφηκαν.

Ακολουθεί τεκμηρίωση του κώδικα του GUI:

**Λειτουργικότητα και Περιγραφή του Κώδικα**

**Κύριες Μεταβλητές**

* **tableName**: Δηλώνει τον επιλεγμένο πίνακα για εκτέλεση λειτουργιών (π.χ. προβολή δεδομένων, εισαγωγή, διαγραφή).
* **action**: Δηλώνει τη λειτουργία που θέλει να εκτελέσει ο χρήστης (π.χ. insert, update, delete).
* **jTextArea2**: Περιοχή κειμένου που εμφανίζει αποτελέσματα από ερωτήματα SQL ή μηνύματα σφάλματος.

**Μέθοδοι**

* **GUI()**: Ο default constructor καλεί την initComponents() για την αρχικοποίηση των στοιχείων του GUI.

**Διαχείριση Πίνακα**

1. **Κουμπιά για Επιλογή Πίνακα**
   * Κάθε κουμπί αντιστοιχεί σε έναν πίνακα της βάσης δεδομένων. Όταν πατηθεί, θέτει την τιμή της μεταβλητής tableName και καλεί τη μέθοδο DisplayTable της κλάσης PROJECT.
2. **Κουμπιά και Μενού για Ενέργειες (Insert, Update, Delete)**
   * Τα μενού (π.χ., jMenuItem56, jMenuItem57, jMenuItem10) ανοίγουν ένα νέο παράθυρο τύπου PopupGui, όπου ο χρήστης μπορεί να εισαγάγει την εντολή SQL.

**Γενική Δομή του GUI**

1. **Κουμπιά (Buttons)**
   * **jButton1 έως jButton18**: Αντιστοιχούν στους πίνακες της βάσης δεδομένων. Πατώντας κάθε κουμπί, γίνεται προβολή δεδομένων του αντίστοιχου πίνακα.
   * Παράδειγμα: jButton1 προβάλει τα δεδομένα του πίνακα consert.
2. **Περιοχή Κειμένου (Text Area)**
   * **jTextArea2**: Εμφανίζει δεδομένα που ανακτώνται από τη βάση δεδομένων ή μηνύματα σφάλματος.
3. **Μενού (Menu Items)**
   * **Εισαγωγή (Insert):** Μέσω του jMenuItem10, ο χρήστης μπορεί να εισαγάγει δεδομένα.
   * **Ενημέρωση (Update):** Μέσω του jMenuItem56, μπορεί να ενημερώσει δεδομένα.
   * **Διαγραφή (Delete):** Μέσω του jMenuItem57, μπορεί να διαγράψει δεδομένα.
4. **Αλληλεπίδραση με Popup Windows**
   * Ο χρήστης μπορεί να ανοίξει ένα παράθυρο (PopupGui) για να εισάγει χειροκίνητα εντολές SQL ανάλογα με τη λειτουργία (insert, update, delete).

**Περιγραφή και Τεκμηρίωση του Κώδικα PopupGui**

**Σημαντικά Τμήματα Κώδικα**

**1. Κύρια Κλάση**

Η κλάση PopupGui είναι μία υποκλάση της javax.swing.JFrame και χρησιμεύει για την εμφάνιση του popup παραθύρου.

* **Constructor**: Ο constructor PopupGui() αρχικοποιεί τα γραφικά στοιχεία της φόρμας καλώντας τη μέθοδο initComponents().

**2. Γραφικά Στοιχεία (Components)**

Στην κλάση περιλαμβάνονται τα εξής κύρια στοιχεία:

* jTextArea1: Πεδίο κειμένου για την είσοδο εντολών SQL από τον χρήστη.
* jButton1: Κουμπί με την ένδειξη "Run" που ενεργοποιεί την εκτέλεση της εντολής.
* jLabel1: Ετικέτα που εμφανίζει το τρέχον είδος ενέργειας (π.χ., INSERT, UPDATE, DELETE).
* jLabel2: Ετικέτα που εμφανίζει μηνύματα σφάλματος ή επιτυχίας.

**3. Μέθοδος jButton1ActionPerformed**

Αυτή η μέθοδος καλείται όταν ο χρήστης πατήσει το κουμπί "Run".

* **Διαδικασία**:
  1. Λαμβάνει την είσοδο του χρήστη από το jTextArea1.
  2. Χωρίζει την είσοδο σε λέξεις χρησιμοποιώντας τη μέθοδο split("\\s+").
  3. Ελέγχει τον τύπο της SQL εντολής (INSERT, UPDATE, DELETE) βασισμένο στη μεταβλητή action και στον πρώτο όρο της εντολής.
  4. Επικυρώνει αν ο πίνακας (tableName) στον οποίο αναφέρεται η εντολή είναι ο σωστός.
  5. Καλεί τις αντίστοιχες μεθόδους του PROJECT για να εκτελέσει την SQL εντολή.
* **Σφάλματα**:
  1. Αν η εντολή δεν είναι συμβατή με τη δράση (action), εμφανίζει σφάλμα.
  2. Αν ο πίνακας (tableName) είναι λανθασμένος, εμφανίζεται μήνυμα σφάλματος.

**2. Σενάριο Χρήσης**

**Περιγραφή Σεναρίου**

Ο χρήστης ανοίγει την εφαρμογή για να διαχειριστεί δεδομένα μιας μουσικής εταιρείας. Στο GUI, μπορεί:

1. Να εισάγει δεδομένα καλλιτεχνών ή άλμπουμ.
2. Να προβάλει δεδομένα με το πάτημα κουμπιού πίνακα που καλεί τη μέθοδο DisplayTable.
3. Να ενημερώσει ή διαγράψει υπάρχουσες εγγραφές με τις αντίστοιχες εντολές.

**Ενδεικτικά Βήματα**

1. Ο χρήστης πατά κουμπί που αντιστοιχεί στον πίνακα που θέλει να εμφανίσει και βλέπει όλα τα δεδομένα ενός πίνακα στο text Area του main GUI.
2. Πατάει το drop down menu "action" και επιλέγει εντολή (insert, update, delete).
3. Πληκτρολογεί μια SQL INSERT/ UPDATE/ DELETE εντολή στο πεδίο κειμένου του popup GUI και πατά "run".

**3. Screenshots και Εξήγηση**

1. **Αρχική Οθόνη του GUI**: Που δείχνει τα κουμπιά εμφάνισης πινάκων και τα drop down menu για προβολή δεδομένων.

A screenshot of a computer

Description automatically generated

1. **Εμφάνιση Δεδομένων**: Μετά το πάτημα κουμπιού (κλήση της DisplayTable), εμφανίζονται τα δεδομένα στο text Area.

A screenshot of a computer

Description automatically generated

1. **Εισαγωγή Δεδομένων**: Μετά το πάτημα insert στο action drop down menu, γράψε την εντολή INSERT.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **Ενημέρωση/Διαγραφή Δεδομένων**: Πάτα το κουμπί που αντιστοιχεί στον πίνακα που έγινε η προσθήκη/ ενημέρωση/ διαγραφή.

A screenshot of a computer

Description automatically generated

# 5. Πρόσθετη λειτουργικότητα

Ακολουθεί αναλυτική τεκμηρίωση του κώδικα:

**Περιγραφή Κλάσης**

Η κλάση getvenuesGUI είναι ένα pop up window που παρέχει τη δυνατότητα στον χρήστη να εισάγει ConsertID και Required Capacity και να εμφανίζει στο text area το venue με τον μεγαλυτερο βαθμό.

**Κύρια Σημεία Κώδικα**

1. **Πεδία Κλάσης**:
   * int conid: Αποθηκεύει το ID της συναυλίας (ConcertID).
   * int reqcap: Αποθηκεύει την απαιτούμενη χωρητικότητα.
2. **Μέθοδος initComponents()**:
   * Αρχικοποιεί και διαμορφώνει τα στοιχεία του GUI, όπως κείμενα εισόδου (conidfield, reqcapfield), την περιοχή αποτελεσμάτων (venuefield) και το κουμπί (jButton1).
3. **Διαχείριση Γεγονότων (jButton1MouseClicked)**:
   * Εκτελείται όταν ο χρήστης κάνει κλικ στο κουμπί "Get Venues".
   * Ελέγχει την εγκυρότητα της εισόδου (με χρήση Integer.parseInt) και εμφανίζει μήνυμα σε περίπτωση σφάλματος.
   * Καλεί δύο stored procedures στη βάση δεδομένων:
     + CALL GetVenueData(?, ?, ?, ?);: Επιστρέφει το ID και τη χωρητικότητα ενός κατάλληλου χώρου.
     + CALL getscore(?, ?);: Επιστρέφει το score ενός συγκεκριμένου χώρου.
4. **Λειτουργία Stored Procedures**:
   * Οι παράμετροι εισόδου και εξόδου διαχειρίζονται με τη χρήση CallableStatement.
   * Τα δεδομένα εξόδου ανακτώνται με τη χρήση getInt.
5. **Εμφάνιση Αποτελεσμάτων**:
   * Αν βρεθεί κατάλληλος χώρος, εμφανίζονται το ID, η χωρητικότητα και το score.
   * Αν δεν βρεθεί, εμφανίζεται το μήνυμα "No suitable venue found.".
6. **Διαχείριση Σφαλμάτων**:
   * Τυχόν SQLException καταγράφεται και εμφανίζεται μήνυμα λάθους στην περιοχή κειμένου.
7. **Screenshots και Εξήγηση**

Main GUI στο οποίο φαίνονται οι εγγραφές του πίνακα venue:

A screenshot of a computer

Description automatically generated

Πατώντας το menu κουμπί “Get venues for consert” του main GUI, εμφανίζεται το pop up window:

A white rectangular object with a gray border

Description automatically generated

Επιλέγω ConsertID: 3 και απαιτούμενη χωριτικότητα: 10000.

Η μέθοδος GetVenueData καλείται και επιστρέφει το venue με τον μεγαλύτερο βαθμό.

A screenshot of a computer

Description automatically generated

Αν δεν βρεθεί ιδανικό venue επιστρέφει το μήνυμα:

A screen shot of a computer

Description automatically generated