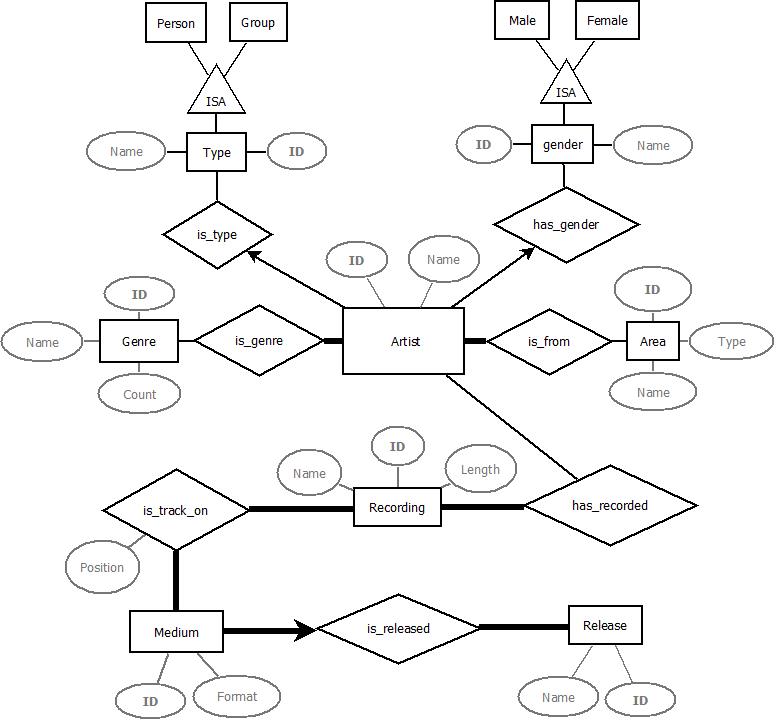
**1 Deliverable 1**

**1.1 ER Diagram**



**1.2 Design choices**

Because we cannot ask customers for their expectations about the database, we did several assumptions listed below. The goal was to maximize the number of caught constraints in our design while avoiding duplication. These are subject to change over time with our design evolution.

1. An artist can have several origins (AREAS) and several genres. However he can only have at most one GENDER and at most one TYPE when those are given.
2. Ideally every artist of type « Group » should have its gender attribute set to « Other ». This is a constraint that we cannot capture in our design.
3. We keep a name field in tables TYPE and GENDER to assure consistency in the beginning of this project, but it would be possible to get rid of them and use only the \_id field to establish an artist’s type or gender by the use of some arbitrary convention.
4. Every recordings in the database should be related to at least one artist, but an artist can have zero recording.
5. All the recordings must appear on at least one medium. A medium must have at least one recording.
6. With the add of a position attribute to the relation is\_track\_on, we capture the definition of a track.
7. A tuple of medium is related to exactly one release.
8. There is no cascade in deletion.

**1.3 SQL DDL code for table creation**

CREATE TABLE Type (

Type\_id INTEGER,

Name CHAR (10),

PRIMARY KEY (Type\_id)) ;

CREATE TABLE Gender (

Gender\_id INTEGER,

Name CHAR (10),

PRIMARY KEY (Gender\_id)) ;

CREATE TABLE Genre (

Genre\_id INTEGER,

Name CHAR (50),

Count INTEGER,

PRIMARY KEY (Genre\_id)) ;

CREATE TABLE Area (

Area\_id INTEGER,

Name CHAR (100),

Type\_of\_area CHAR (100),

PRIMARY KEY (Area\_id)) ;

CREATE TABLE Artist (

Artist\_id INTEGER,

Name CHAR (30),

Type\_id INTEGER,

Gender\_id INTEGER,

Genre\_id INTEGER NOT NULL,

Area\_id INTEGER NOT NULL,

PRIMARY KEY (Artist\_id),

FOREIGN KEY (Type\_id) REFERENCES Type,

FOREIGN KEY (Gender\_id) REFERENCES Gender,

FOREIGN KEY (Genre\_id) REFERENCES Genre,

FOREIGN KEY (Area\_id) REFERENCES Area) ;

CREATE TABLE Recording (

Recording\_id INTEGER,

Name CHAR (100),

Length INTEGER,

PRIMARY KEY (Recording\_id)) ;

CREATE TABLE has\_recorded (

Artist\_id INTEGER, // implicit NOT NULL (used in primary key)

Recording\_id INTEGER, // implicit NOT NULL (idem)

PRIMARY KEY (Artist\_id, Recording\_id),

FOREIGN KEY (Artist\_id) REFERENCES Artist,

FOREIGN KEY (Recording\_id) REFERENCES Recording) ;

CREATE TABLE Medium (

Medium\_id INTEGER,

Format CHAR (30),

Release\_id INTEGER NOT NULL,

PRIMARY KEY (Medium\_id),

FOREIGN KEY (Release\_id) REFERENCES Release) ;

CREATE TABLE Release (

Release\_id INTEGER,

Name CHAR (200),

PRIMARY KEY (Release\_id)) ;

CREATE TABLE is\_track\_on (

Recording\_id INTEGER,

Medium\_id INTEGER,

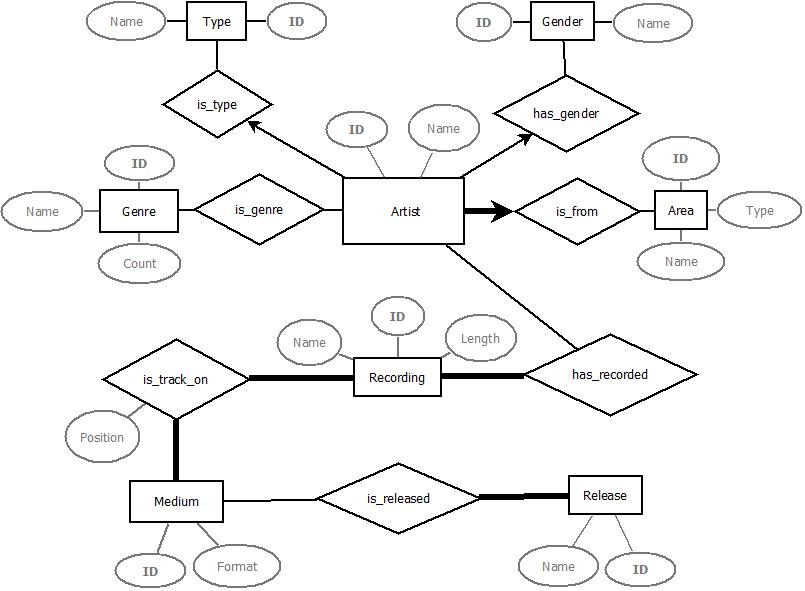
Position INTEGER,

PRIMARY KEY (Recording\_id, Medium\_id, Position),

FOREIGN KEY (Recording\_id) REFERENCES Recording,

FOREIGN KEY (Medium\_id) REFERENCES Medium) ;

**2 Deliverable 2**

**2.1 ER Diagram**

**2.2 Design Choices**

From the feedbacks of the deliverable 1, we made some improvements to our diagram.

First we got rid of the ISAs relationships because they were carrying more complexity than help. Each Type and Gender tables will contain few entries, respectively {‘Person’, ‘Group’} and {‘Male’, ‘Female’, ‘Other’}. This could allow us to add an eventual new Type if needed (not expected in this project though).

We limited the ability for an artist to be from more than one Area as proposed in the feedback. Now an artist must be from exactly one Area.

We allowed an artist to have no genre. Even though the number of genre should cover most needs we thought it was a useless limitation.

The new relation between Medium and Release allow the first to be part of several or none Release (many-to-many relation). We kept the fact that a Release must have at least one Medium.

We can use arbitrary convention for Type and Gender directly in queries, such as 0 for Person and 1 for Group; 0 for Male, 1 for Female and 2 for Other.

**2.3 SQL DDL Code for table creation**

CREATE TABLE Type (

Type\_id INTEGER,

Name CHAR (10),

PRIMARY KEY (Type\_id)) ;

CREATE TABLE Gender (

Gender\_id INTEGER,

Name CHAR (10),

PRIMARY KEY (Gender\_id)) ;

CREATE TABLE Genre (

Genre\_id INTEGER,

Name CHAR (50),

Count INTEGER,

PRIMARY KEY (Genre\_id)) ;

CREATE TABLE Area (

Area\_id INTEGER,

Area\_name CHAR (100),

Type\_of\_area CHAR (100),

PRIMARY KEY (Area\_id)) ;

CREATE TABLE Artist (

Artist\_id INTEGER,

Artist\_name CHAR (30),

Type\_id INTEGER,

Gender\_id INTEGER,

Area\_id INTEGER NOT NULL,

PRIMARY KEY (Artist\_id),

FOREIGN KEY (Type\_id) REFERENCES Type,

FOREIGN KEY (Gender\_id) REFERENCES Gender,

FOREIGN KEY (Area\_id) REFERENCES Area) ;

CREATE TABLE is\_genre (

Artist\_id INTEGER,

Genre\_id INTEGER,

PRIMARY KEY (Artist\_id, Genre\_id),

FOREGIN KEY (Artist\_id) REFERENCES Artist,

FOREIGN KEY (Genre\_id) REFERENCES Genre) ;

CREATE TABLE Recording (

Recording\_id INTEGER,

Name CHAR (100),

Length INTEGER,

PRIMARY KEY (Recording\_id)) ;

CREATE TABLE has\_recorded (

Artist\_id INTEGER, // implicit NOT NULL (used in primary key)

Recording\_id INTEGER, // implicit NOT NULL (idem)

PRIMARY KEY (Artist\_id, Recording\_id),

FOREIGN KEY (Artist\_id) REFERENCES Artist,

FOREIGN KEY (Recording\_id) REFERENCES Recording) ;

CREATE TABLE Medium (

Medium\_id INTEGER,

Format CHAR (30),

PRIMARY KEY (Medium\_id)) ;

CREATE TABLE Release (

Release\_id INTEGER,

Name CHAR (200),

PRIMARY KEY (Release\_id)) ;

CREATE TABLE is\_track\_on (

Recording\_id INTEGER,

Medium\_id INTEGER,

Position INTEGER,

PRIMARY KEY (Recording\_id, Medium\_id, Position),

FOREIGN KEY (Recording\_id) REFERENCES Recording,

FOREIGN KEY (Medium\_id) REFERENCES Medium) ;

CREATE TABLE is\_released (

Medium\_id INTEGER,

Release\_id INTEGER,

PRIMARY KEY (Medium\_id, Release\_id),

FOREIGN KEY (Medium\_id) REFERENCES Medium,

FOREIGN KEY (Release\_id) REFERENCES Release) ;

**2.4 Import of the Datas**

We created a table for each CSV in which we imported them. We used queries to extract the wanted datas in our own table. For instance, some datas such as ‘\N’ in recording.csv were changed to ‘-1’ or simply ignored when not convenient to our table definition.

**2.5 Queries**

A. Print the names of artists from Switzerland, i.e., artists whose area is Switzerland. You should not include the names of the artists associated with individual cantons and towns in Switzerland.

SELECT Artist\_name

FROM ( SELECT \*

FROM Area

WHERE Area\_name='Switzerland') NATURAL JOIN Artist

B. Print the names of areas with the highest number male artists, female artists and groups. For each of these 3 areas, print the number of artists of each of the three types in the area.

C. List the names of 10 groups with the most recorded tracks.

SELECT Artist.name

FROM ( SELECT Artist\_id, COUNT(Recording\_id) AS c1

FROM has\_recorded, Artist

WHERE Type\_id = '0'

GROUP BY Artist\_id ) NATURAL JOIN Artist

ORDER BY c1 DESC

LIMIT 10

D. List the names of 10 groups with the most releases.

SELECT Artist.name

FROM ( TODO )

ORDER BY c1 DESC

LIMIT 10

E. Print the name of a female artist associated with the most genres.

SELECT Artist\_name

FROM ( SELECT Artist\_id, COUNT(Genre\_id) AS G

FROM ( SELECT \*

FROM ( SELECT \*

FROM Artist

WHERE Gender\_id=1 ) NATURAL JOIN is\_genre)

GROUP BY Artist\_id )

ORDER BY G DESC

LIMIT 1

**2.6 Screenshots**

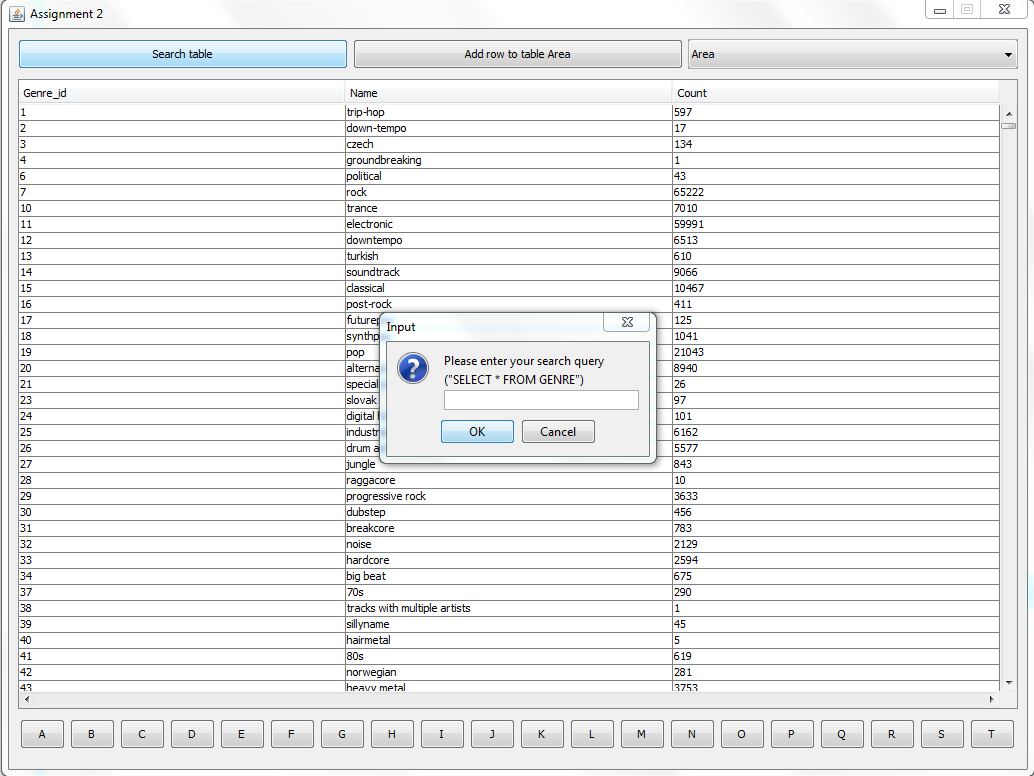
Here are a few screenshots of our interface. We chose to make a Java solution.

Figure 1 Search Functionality

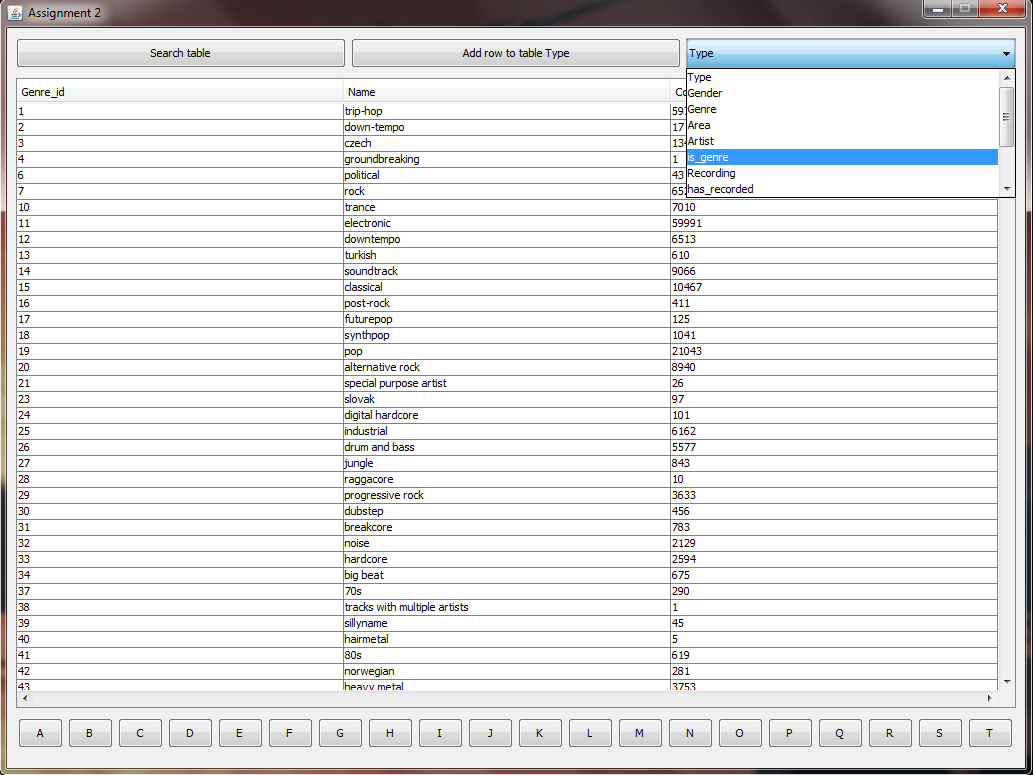


Figure 2 Tables View

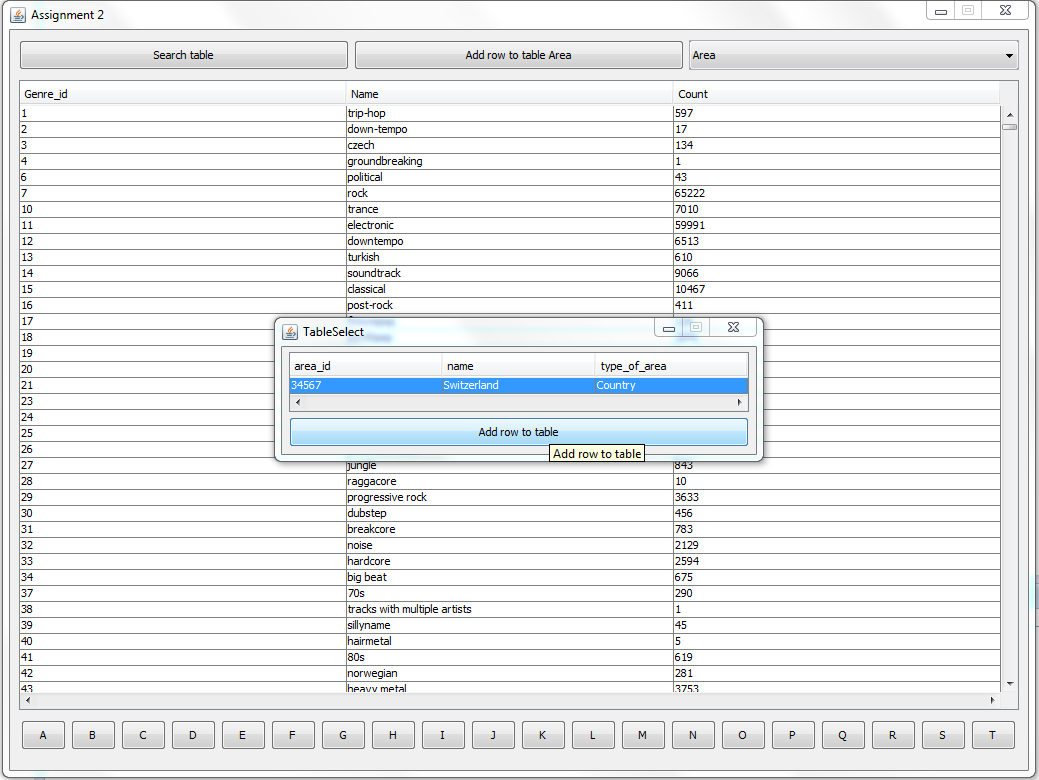
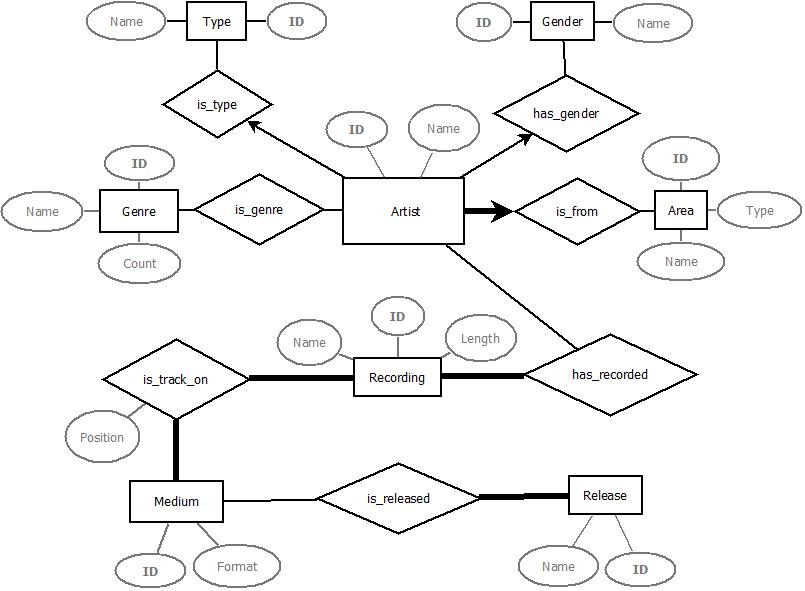


Figure 3 Data Entry Functionality

**3 Deliverable 3**

**3.1 ER Diagram**

**3.2 Design Choices**

Identical to deliverable 2,

**3.3 SQL DDL Code for table creation**

Identical to deliverable 2.

**3.4 Software**

Although we did not change the structural design we reorganized the way we are storing queries which gives us more freedom to add new ones and to test them directly.

We store each query in a file.sql and when clicking on the button we load them from the disk and execute them.

**3.5 Import of the Datas**

For each file.csv we created a corresponding table having the same specifications. In a first step we imported all the file.csv into these tables. As a second step we ran several queries which are stored in fillTables.sql to fill our defined tables.

/\* Type \*/

INSERT INTO Type (Type\_id, Name) VALUES (0, 'Person')

INSERT INTO Type (Type\_id, Name) VALUES (1, 'Groupe')

/\* Gender \*/

INSERT INTO Gender (Gender\_id, Name) VALUES (0, 'Male')

INSERT INTO Gender (Gender\_id, Name) VALUES (1, 'Female')

INSERT INTO Gender (Gender\_id, Name) VALUES (2, 'Ohter')

/\* Genre \*/

INSERT INTO Genre (Genre\_id, Name, Count) SELECT \* FROM Genre\_csv

/\* Area \*/

INSERT INTO Area (Area\_id, Name, Type\_of\_area) SELECT \* FROM Area\_csv

/\* Artist \*/

INSERT INTO Artist (Artist\_id, Name, Type\_id, Gender\_id, Area\_id) SELECT \* FROM Artist\_csv

/\* is\_genre \*/

INSERT INTO is\_genre (Artist\_id, Genre\_id) SELECT \* FROM Artist\_genre\_csv

/\* Recording \*/

INSERT INTO Recording (Recording\_id, Name, Length) SELECT \* FROM Recording\_csv

/\* has\_recorded \*/

INSERT INTO has\_recorded (Artist\_id, Recording\_id) SELECT DISTINCT ArtistID, TrackID FROM artist\_track\_csv NATURAL JOIN Track\_csv

/\* Medium \*/

INSERT INTO Medium (Medium\_id, Format) SELECT ID, Format FROM Medium\_csv

/\* Release \*/

INSERT INTO Release (Release\_id, Name) SELECT \* FROM Release\_csv

/\* is\_track\_on \*/

INSERT INTO is\_track\_on (Recording\_id, Medium\_id, Position) SELECT RecordingID, MediumID, Position FROM Track\_csv

/\* is\_released \*/

INSERT INTO is\_released (Medium\_id, Release\_id) SELECT ID, ReleaseID FROM Medium\_csv

**3.6 Queries**

To get a complete overview of all the queries we add all of them in this chapter:

A: "Print the names of artists from Switzerland."

SELECT Name

FROM (

SELECT Area\_id

FROM Area

WHERE Name = 'Switzerland'

)

NATURAL JOIN Artist

B: "Print the names of areas with the highest number male artists, female artists and groups, as well as the number of artists of each type in each of the three areas."

SELECT a1.Name, male, female, groups

FROM Area a1, (

SELECT \*

FROM(

SELECT Area\_id, male, female, groups

FROM (

SELECT a1.Area\_id,

COUNT(CASE WHEN a2.Gender\_id = 'Male' THEN 1 ELSE NULL END) AS male,

COUNT(CASE WHEN a2.Gender\_id = 'Female' THEN 1 ELSE NULL END) AS female,

COUNT(CASE WHEN a2.Type\_id = 'Group' THEN 1 ELSE NULL END) AS groups

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

WHERE male IN (

SELECT MAX(male)

FROM(

SELECT a1.Area\_id,

COUNT(CASE WHEN a2.Gender\_id = 'Male' THEN 1 ELSE NULL END) AS male,

COUNT(CASE WHEN a2.Gender\_id = 'Female' THEN 1 ELSE NULL END) AS female,

COUNT(CASE WHEN a2.Type\_id = 'Group' THEN 1 ELSE NULL END) AS groups

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

)

LIMIT 1

)

UNION ALL

SELECT \*

FROM(

SELECT Area\_id, male, female, groups

FROM (

SELECT a1.Area\_id,

COUNT(CASE WHEN a2.Gender\_id = 'Male' THEN 1 ELSE NULL END) AS male,

COUNT(CASE WHEN a2.Gender\_id = 'Female' THEN 1 ELSE NULL END) AS female,

COUNT(CASE WHEN a2.Type\_id = 'Group' THEN 1 ELSE NULL END) AS groups

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

WHERE female IN (

SELECT MAX(female)

FROM(

SELECT a1.Area\_id,

COUNT(CASE WHEN a2.Gender\_id = 'Male' THEN 1 ELSE NULL END) AS male,

COUNT(CASE WHEN a2.Gender\_id = 'Female' THEN 1 ELSE NULL END) AS female,

COUNT(CASE WHEN a2.Type\_id = 'Group' THEN 1 ELSE NULL END) AS groups

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

)

LIMIT 1

)

UNION ALL

SELECT \*

FROM(

SELECT Area\_id, male, female, groups

FROM (

SELECT a1.Area\_id,

COUNT(CASE WHEN a2.Gender\_id = 'Male' THEN 1 ELSE NULL END) AS male,

COUNT(CASE WHEN a2.Gender\_id = 'Female' THEN 1 ELSE NULL END) AS female,

COUNT(CASE WHEN a2.Type\_id = 'Group' THEN 1 ELSE NULL END) AS groups

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

WHERE groups IN (

SELECT MAX(groups)

FROM(

SELECT a1.Area\_id,

COUNT(CASE WHEN a2.Gender\_id = 'Male' THEN 1 ELSE NULL END) AS male,

COUNT(CASE WHEN a2.Gender\_id = 'Female' THEN 1 ELSE NULL END) AS female,

COUNT(CASE WHEN a2.Type\_id = 'Group' THEN 1 ELSE NULL END) AS groups

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

)

LIMIT 1

)

) a2

WHERE a1.Area\_id = a2.Area\_id

C: "List the names of 10 groups with the most recorded tracks."

SELECT Name

FROM (

SELECT a.Name, COUNT( i.Medium\_id ) AS Records

FROM has\_recorded h, Artist a, Recording r, is\_track\_on i

WHERE a.Type\_id = 'Group' AND a.Artist\_id = h.Artist\_id AND h.Recording\_id = r.Recording\_id AND r.Recording\_id = i.Recording\_id

GROUP BY a.Artist\_id

)

ORDER BY Records DESC LIMIT 10

D: "List the names of 10 groups with the most releases."

SELECT Name

FROM (

SELECT a.Name, COUNT( DISTINCT r1.Release\_id ) AS Releases

FROM Release r1, is\_released ir, is\_track\_on t, has\_recorded h, Artist a, Medium m, Recording r2

WHERE a.Type\_id = 'Group' AND a.Artist\_id = h.Artist\_id AND h.Recording\_id = r2.Recording\_id AND r2.Recording\_id = t.Recording\_id AND t.Medium\_id = m.Medium\_id AND m.Medium\_id = ir.Medium\_id AND ir.Release\_id = r1.Release\_id

GROUP BY a.Artist\_id

)

ORDER BY Releases DESC LIMIT 10

E: "Print the name of a female artist associated with the most genres."

SELECT Name

FROM (

SELECT a.Name, COUNT(g.Genre\_id) as c

FROM Artist a, is\_genre i, Genre g

WHERE a.Gender\_id = 'Female' AND a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id

GROUP BY a.Artist\_id

)

ORDER BY c DESC LIMIT 1

F: "List all cities which have more female than male artists."

SELECT name

FROM (

SELECT Area\_id, name, COUNT(Gender\_id) as C1

FROM (

(SELECT Gender\_id, Area\_id

FROM Artist)

NATURAL JOIN Area

)

WHERE Gender\_id = 'Male' AND Type\_of\_Area='City'

GROUP BY Area\_id

)

NATURAL JOIN (

SELECT Area\_id, name, COUNT(Gender\_id) as C2

FROM (

(SELECT Gender\_id, Area\_id

FROM Artist)

NATURAL JOIN Area

)

WHERE Gender\_id = 'Female' AND Type\_of\_Area='City'

GROUP BY Area\_id

)

WHERE C2 > C1

G: "List the releases with the highest number of tracks."

SELECT Name

FROM Release

WHERE Release\_id IN (

SELECT Release\_id

FROM (

SELECT r.Release\_id, COUNT(i2.Recording\_id) AS a

FROM Release r, is\_released i1, is\_track\_on i2

WHERE r.Release\_id = i1.Release\_id AND i1.Medium\_id = i2.Medium\_id

GROUP BY r.Release\_id )

WHERE a IN (

SELECT MAX(b)

FROM (

SELECT r.Release\_id, COUNT(i2.Recording\_id) AS b

FROM Release r, is\_released i1, is\_track\_on i2

WHERE r.Release\_id = i1.Release\_id AND i1.Medium\_id = i2.Medium\_id

GROUP BY r.Release\_id

)

)

)

H: "For each area that has more than 30 artists, list the male artist, the female artist and the group with the most tracks recorded."

SELECT Area\_id , artists

FROM(

SELECT a1.Area\_id, COUNT(a2.Artist\_id) as artists

FROM Area a1, Artist a2

WHERE a1.Area\_id = a2.Area\_id

GROUP BY a1.Area\_id

)

WHERE artists > 30

I: "List the name of the release with the most tracks."

SELECT Name

FROM(

SELECT r.Name, r.Release\_id, COUNT(t.Recording\_id) as tracks

FROM Release r, is\_released i, Medium m, is\_track\_on t

WHERE r.Release\_id = i.Release\_id AND i.Medium\_id = m.Medium\_id AND m.Medium\_id = t.Medium\_id

GROUP BY r.Release\_id

)

ORDER BY tracks LIMIT 1

J: "American metal group Metallica is asking its fans to choose the setlist for its upcoming concert in Switzerland. Assuming that the Metallica fans will choose the songs that have appeared on the highest number of mediums, list the top 25 songs."

SELECT Name

FROM(

SELECT r.Name, COUNT( m.Medium\_id ) med

FROM has\_recorded h, Recording r, is\_track\_on t, Medium m, (

SELECT a.Artist\_id

FROM Artist a

WHERE a.Name = 'Metallica'

) a

WHERE a.Artist\_id = h.Artist\_id AND h.Recording\_id = t.Recording\_id AND t.Medium\_id = m.Medium\_id

GROUP BY r.Recording\_id

)

ORDER BY med LIMIT 25

K: "For each of the 10 genres with the most artists, list the most popular female artist."

L: "List all genres that have no female artists, all genres that have no male artists and all genres that have no groups."

SELECT Name

FROM Genre

WHERE Genre\_id NOT IN (

SELECT g.Genre\_id

FROM Artist a, is\_genre i, Genre g

WHERE a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id AND a.Gender\_id = 'Male'

)

UNION

SELECT Name

FROM Genre

WHERE Genre\_id NOT IN (

SELECT g.Genre\_id

FROM Artist a, is\_genre i, Genre g

WHERE a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id AND a.Gender\_id = 'Female'

)

UNION

SELECT Name

FROM Genre

WHERE Genre\_id NOT IN (

SELECT g.Genre\_id

FROM Artist a, is\_genre i, Genre g

WHERE a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id AND a.Type\_id = 'Group'

)

M: "For each area with more than 10 groups, list the 5 male artists that have recorded the highest number of tracks."

N: "List the 10 groups with the highest number of tracks that appear on compilations."

O: "List the top 10 releases with the most collaborations, i.e., releases where one artist is performing all songs and the highest number of different guest artists contribute to the album."

P: "List the release which is associated with the most mediums. If there are more than one such release, list all such releases."

SELECT Name

FROM Release

WHERE Release\_id IN (

SELECT Release\_id

FROM(

SELECT i.Release\_id, COUNT(i.Medium\_id) as med1

FROM is\_released i, Release r, Medium m

WHERE i.Release\_id = r.Release\_id AND m.Medium\_id = i.Medium\_id

GROUP BY i.Release\_id

)

WHERE med1 IN (

SELECT MAX(med2)

FROM(

SELECT i.Release\_id, COUNT(i.Medium\_id) as med2

FROM is\_released i, Release r, Medium m

WHERE i.Release\_id = r.Release\_id AND m.Medium\_id = i.Medium\_id

GROUP BY i.Release\_id

)

)

)

Q: "List the most popular genre among the groups which are associated with at least 3 genres."

SELECT Name

FROM Genre

WHERE Genre\_id IN (

SELECT Genre\_id

FROM (

SELECT g.Genre\_id, COUNT(a.Artist\_id) as art1

FROM Artist a, is\_genre i, Genre g

WHERE a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id AND a.Artist\_id IN (

SELECT Artist\_id

FROM (

SELECT a.Artist\_id, COUNT(g.Genre\_id) as gen

FROM Artist a, is\_genre i, Genre g

WHERE a.Type\_id = 'Group' AND a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id

GROUP BY a.Artist\_id

)

WHERE gen > 2

)

GROUP BY g.Genre\_id

)

WHERE art1 IN (

SELECT MAX(art2)

FROM(

SELECT g.Genre\_id, COUNT(a.Artist\_id) as art2

FROM Artist a, is\_genre i, Genre g

WHERE a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id AND a.Artist\_id IN (

SELECT Artist\_id

FROM (

SELECT a.Artist\_id, COUNT(g.Genre\_id) as gen

FROM Artist a, is\_genre i, Genre g

WHERE a.Type\_id = 'Group' AND a.Artist\_id = i.Artist\_id AND i.Genre\_id = g.Genre\_id

GROUP BY a.Artist\_id

)

WHERE gen > 2

)

GROUP BY g.Genre\_id

)

)

)

R: "List the 5 most popular song titles along with the number of songs that share such title."

SELECT name, COUNT(Recording\_id) as records

FROM Recording

GROUP BY name

ORDER by records DESC LIMIT 5

S: "List the top 10 artists according to their track to release ratio."

T: "The concert hit index is a measure of probability that the artist can attract enough fans to fill a football stadium. We define the “hit artist” as one that has more than 10 songs that appear on more than 100 mediums and measure \"hit ability\" as the average number of mediums that a top 10 song appears on. List all “hit artists” according to their \"hit ability\"."

**3.7 Running Time**

To measure the execution time of each query we added a new static class Timing with two simple methods you can call from any program. Before executing a query you call startCounter() which gets the current milliseconds of the computer and stores it locally in a variable counter in Timing.java. After the query is executed you call endCounter() which again gets the current milliseconds and compares it to its local variable counter and prints on the console the exact execution time in milliseconds.

We measured the following delays:

A: 9ms

B: 2581ms

C: 0ms

D: 15777ms

E: 575ms

F: 322ms

G: 284007ms

H: 409ms

I: 72210ms

J: 74340ms

K: -

L: 4857ms

M: -

N: -

O: -

P: 13225ms

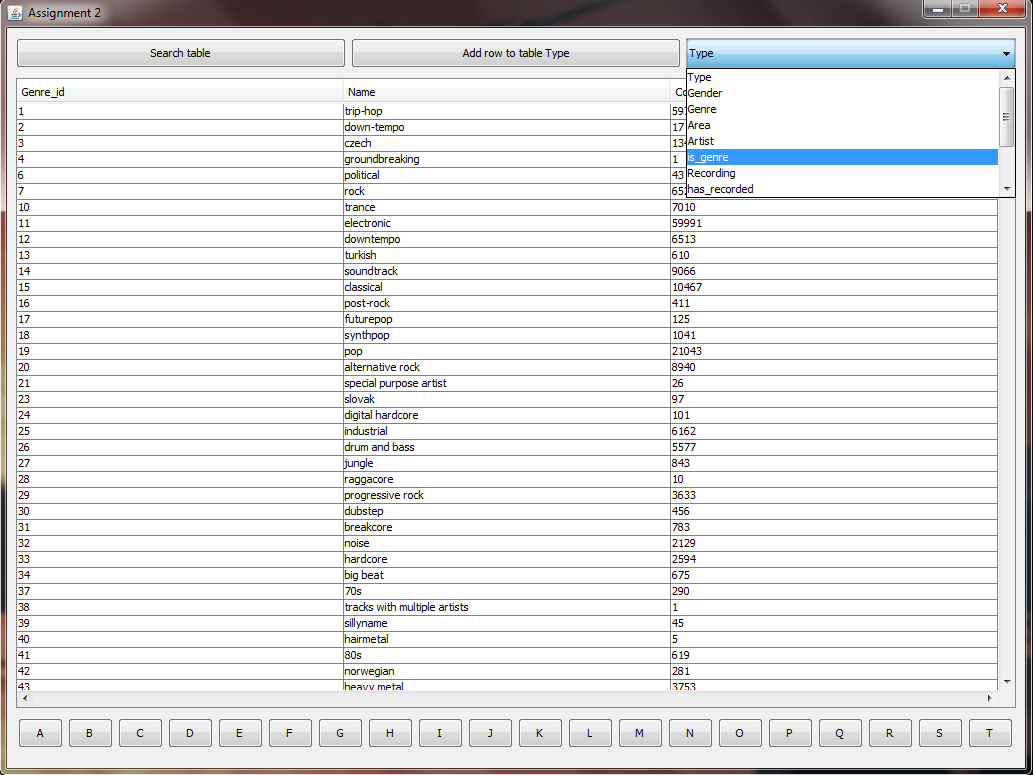
Q: 684ms

R: 45759ms

S: -

T: -

**3.8 Screenshots**

Here are a few screenshots of our interface. We chose to make a Java solution.

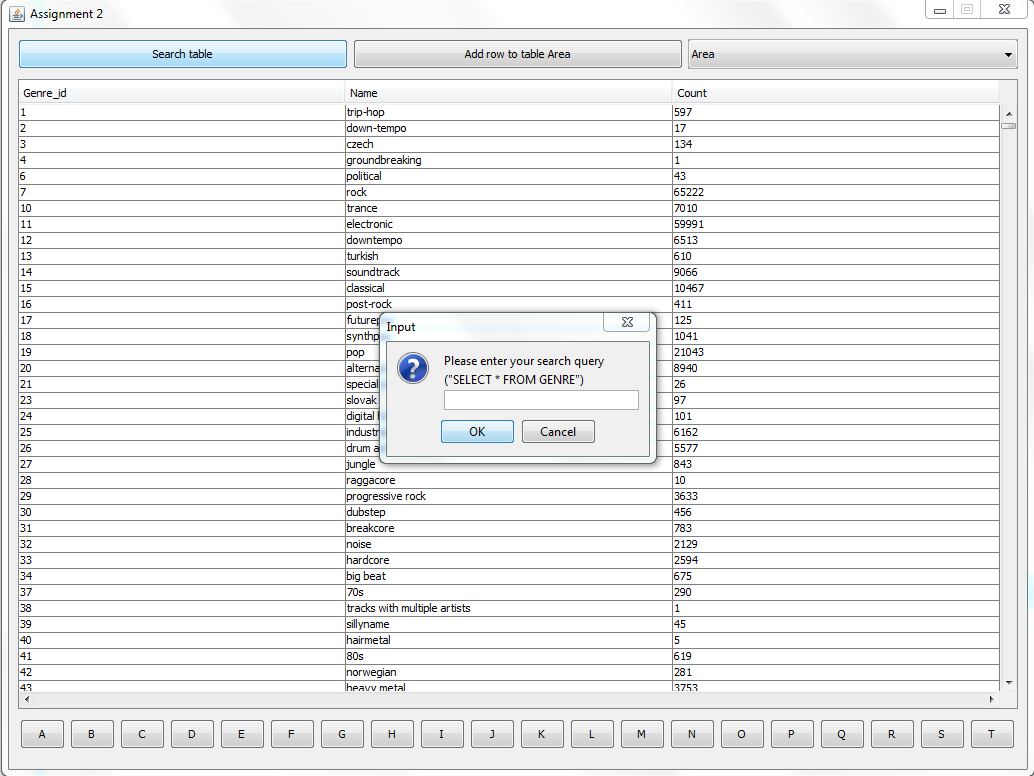


Figure 1 Search Functionality

Figure 2 Tables View

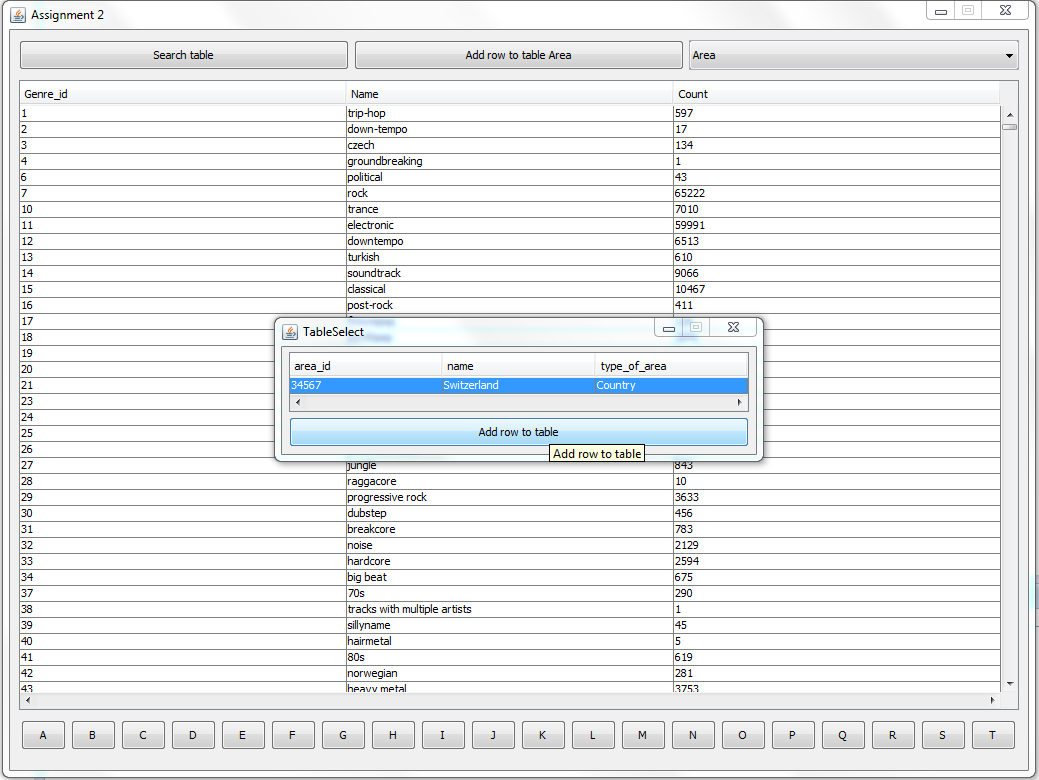
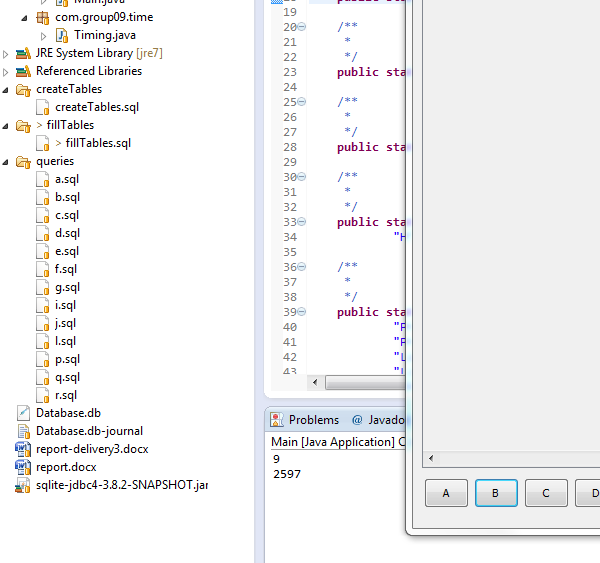
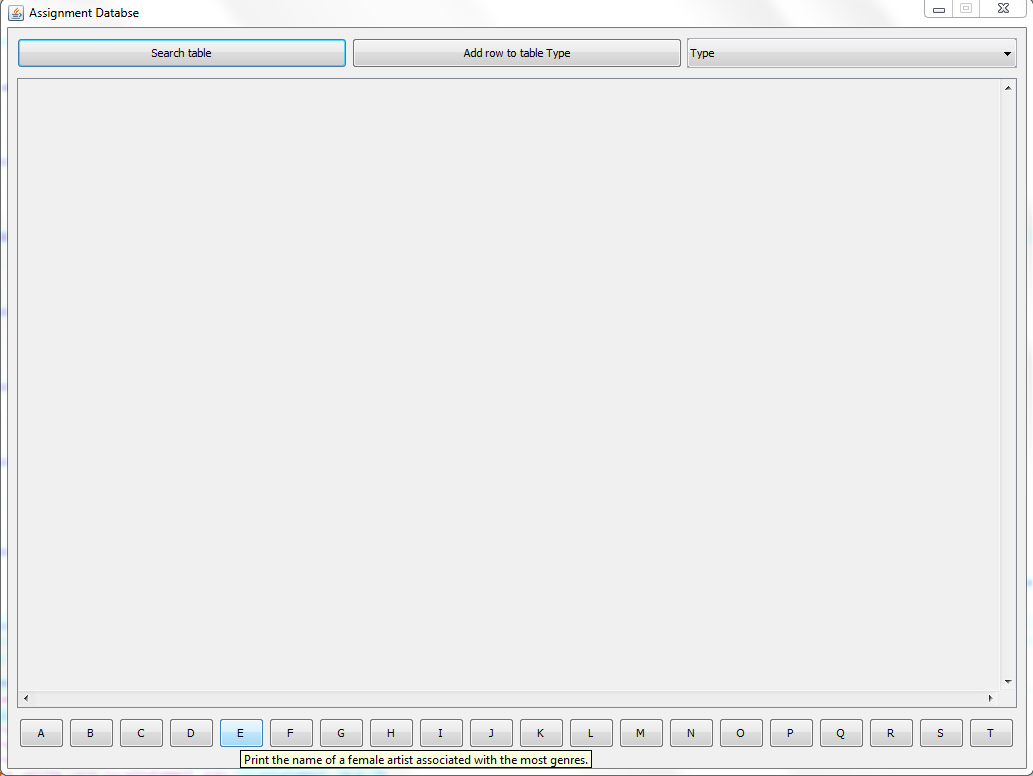


Figure 3 Data Entry Functionality





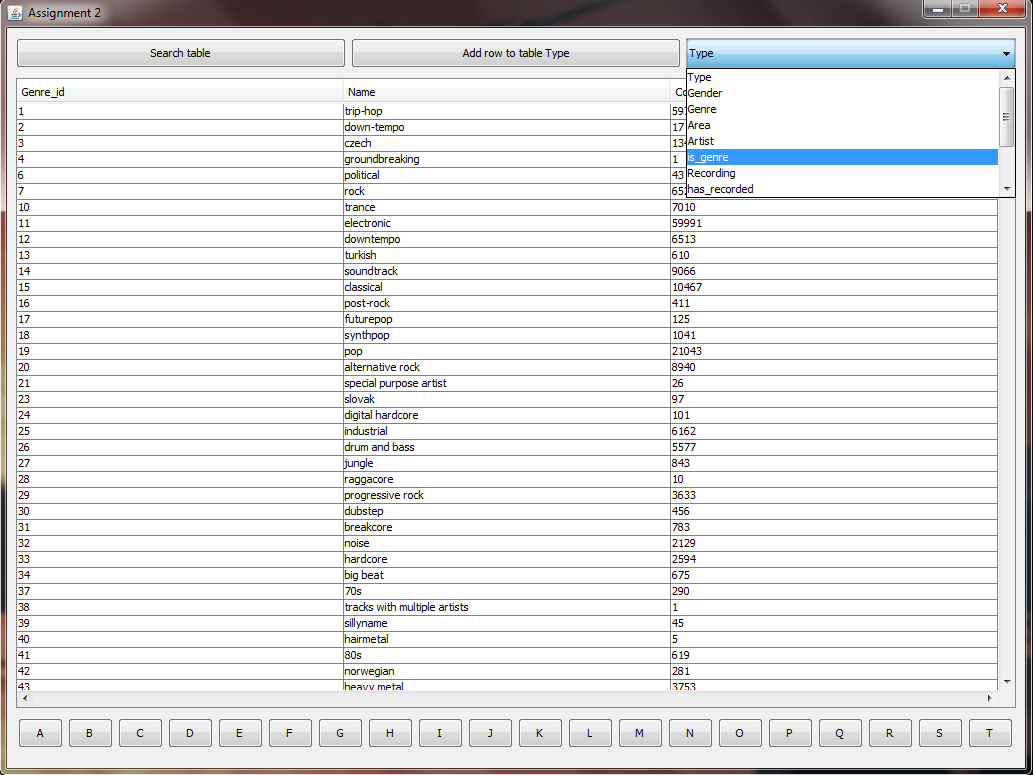


Figure 2 Tables View

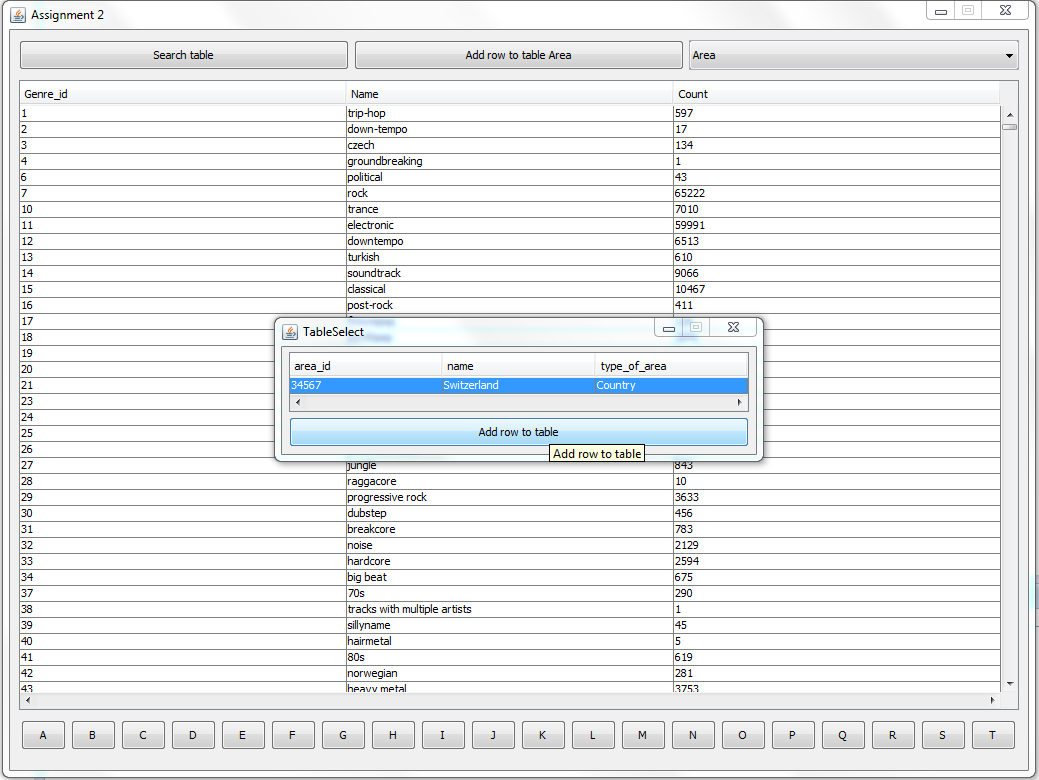


Figure 3 Data Entry Functionality