**Importation**

We were confronted to some problems during the import part. The « real » data which we have at our disposal are far from the idealized schema that we made for the first part. To reduce those differences, some modifications had to be made.

New ER-Model

Because some data are missing, artist can be associated with one or zero area and release can be associated with one or zero medium.



Changes in Tables Creation

Weaken constraints :

Some constraints had to be weaken by deleting some FOREIGN KEY. Indeed, some constraints that seem natural during the design part of the database, were finally not well adapted to the available data. For example, in the artist table the FOREIGN KEY on the area ID had to be suppressed because some artists are associated with an area ID that does not exist in the area table. For the same reason the FOREIGN KEY on the artist ID in the artist\_genre table had to be suppressed, as well as the FOREIGN KEY on the track ID in the artist\_track table and the FOREIGN KEY on the Medium ID in the Track table.

Merges for participation constraints :

To capture the constraints that an artist is associated with at most one area, the artist\_area table is merged with the artist table. The same is made for the physical\_song and the medium.

Fields types more accurate :

We also decided to change every field type for the tables creation (before every fields were just CHAR).

Every field for the creation has to be modified. IDs are only integer, thus use integer instead of CHAR are more suitable. For other fields, VARCHAR are used instead of CHAR because the lengths of the field are really variable thus VARCHAR is more appropriated for the real values of the data.

New table creation sql code :

**CREATE** **TABLE** Area(

ID\_Area INT,

Name VARCHAR2(1000),

**Type** VARCHAR2(60),

**PRIMARY** **KEY**(ID\_Area)

);

**CREATE** **TABLE** Genre

(ID\_Genre INT,

Name VARCHAR2(1000),

**Count** INT,

**PRIMARY** **KEY**(ID\_Genre));

**CREATE** **TABLE** Recording

(ID\_Recording INT,

Name VARCHAR2(2000),

**Length** INT,

**PRIMARY** **KEY**(ID\_Recording));

**CREATE** **TABLE** Release

(ID\_Release INT,

Name VARCHAR2(1000),

**PRIMARY** **KEY**(ID\_Release));

**CREATE** **TABLE** Artist

(ID\_Artist INT,

Name VARCHAR2(1000),

**Type** VARCHAR2(60),

Gender VARCHAR2(20),

ID\_Area INT,

**PRIMARY** **KEY**(ID\_Artist)

);

**CREATE** **TABLE** Medium

(ID\_Medium INT,

Format VARCHAR2(60),

ID\_Release INT,

**PRIMARY** **KEY**(ID\_Medium)

);

**CREATE** **TABLE** Track

(ID\_Track INT,

**Position** INT,

ID\_Medium INT,

ID\_Recording INT,

**PRIMARY** **KEY**(ID\_Track),

**FOREIGN** **KEY**(ID\_Recording) **REFERENCES** Recording

);

**CREATE** **TABLE** Artist\_Genre

(ID\_Artist INT,

ID\_Genre INT,

**PRIMARY** **KEY**(ID\_Artist, ID\_Genre),

**FOREIGN** **KEY**(ID\_Artist) **REFERENCES** Artist,

**FOREIGN** **KEY**(ID\_Genre) **REFERENCES** Genre);

**CREATE** **TABLE** Artist\_Track

(ID\_Artist INT,

ID\_Track INT,

**PRIMARY** **KEY**(ID\_Artist, ID\_Track),

**FOREIGN** **KEY**(ID\_Artist) **REFERENCES** Artist);

**Search Functionality**

We decided to implement the search functionality as follow:

1. We chose not to allow the user to search on any column of a table. The reason is we thought all the columns were not really interesting to run queries on. For example in the GENRE table there is a count column, but we believe this is not a useful query to be run for a lambda user. That’s why the only search that can be done on our UI is about the NAME column of the table.
2. We chose to allow the user only to search in table where there is a NAME column. The reason follow from the discussion in the first point previously mentioned. Thus the available tables for search are: AREA, ARTIST, GENRE, RECORDING and RELEASE. In the UI, this table selection can be done through a drop-down list (in order to not let the user write junk as a table name).
3. We construct our queries by using the LIKE operator of SQL using the keyword the user inputted. More formally the query was designed as follow:

"SELECT \* FROM "**+**table**+**" WHERE LOWER(name) LIKE lower('%" **+** keyword **+** "%')"

1. The resulting rows of the simple search query are displayed in a HTML table. The HTML columns are the SQL columns of the table that was searched without id related columns. For example if the search was made on the ARTIST table, the HTML columns are NAME, TYPE and GENDER.
2. For the follow up queries we decided that the user can have more infos about a row by clicking on it. We decided to do so since the resulting queries are made upon the id of the row and not about a specific column.
3. The follow up queries are made using the id of the clicked row on chosen tables that seemed logical to be linked to for us.

More precisely the tables are linked like that:

|  |  |
| --- | --- |
| **ID from table** | **Linked with table(s)** |
| AREA | ARTIST |
| ARTIST | AREA, GENRE, RELEASE |
| GENRE | ARTIST |
| RECORDING | ARTIST, RELEASE |
| RELEASE | ARTIST, RECORDING |

These choices were obvious for us since we thought of it as if we were a user of the UI and we talk between us about “What more infos would want the user x if he choose to search a keyword y?”

1. Concerning the displaying we chose to use tabs. We display one of the linked tables and then the user can click on the other tabs to go to the other linked tables. When the user click on one row of one of the linked table the same behaviour occurs as if these row resulted from a simple search query.

**SQL Queries**

We have still some problems for the SQL queries. We test all the following queries with a dummy database that is smaller than the music database and the queries give us the expected results. But with the real database some of the queries take a lot of time to be executed (we kill the process after 5 minutes and when we let the execution run the following error can appear…). Those queries are the query B, C, D, F and G. Thus we think that this problem comes from a lack of optimisation of those queries.



Figure 1: error when the query E is run during a long time

Queries A-G:

--A print the name of artist from switzerland

SELECT A.name

FROM Artist A, Area B

WHERE A.ID\_AREA = B.ID\_AREA AND B.name= 'Switzerland'

--B print the name and the number of female, male and group of the

--area that have the most female, male or group artists.

SELECT \*

FROM

(SELECT B.name AS NAME,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.GENDER='Female' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTF,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.GENDER='Male' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTM,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.TYPE='Group' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTG

FROM AREA B

ORDER BY COUNTF DESC)

WHERE ROWNUM=1

UNION

SELECT \*

FROM

(SELECT B.name AS NAME,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.GENDER='Female' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTF,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.GENDER='Male' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTM,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.TYPE='Group' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTG

FROM AREA B

ORDER BY COUNTM DESC)

WHERE ROWNUM=1

UNION

SELECT \*

FROM

(SELECT B.name AS NAME,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.GENDER='Female' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTF,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.GENDER='Male' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTM,

(SELECT COUNT(DISTINCT A.ID\_ARTIST)

        FROM ARTIST A

        WHERE A.TYPE='Group' AND B.ID\_AREA=A.ID\_AREA)

        AS COUNTG

FROM AREA B

ORDER BY COUNTG DESC)

WHERE ROWNUM=1;

--C List the name of 10 groups with the most recorded track

SELECT \*

FROM(SELECT A.NAME

FROM  Artist A, Artist\_Track S

WHERE A.ID\_ARTIST=S.ID\_ARTIST AND A.TYPE='Group'

GROUP BY A.ID\_ARTIST, A.NAME

ORDER BY count(S.ID\_TRACK) DESC)

WHERE ROWNUM <=10

--D List the name of 10 groups with the most release

SELECT \*

FROM(SELECT **A.NAME**

FROM Artist A, Track T, Artist\_Track S, Medium M, Release R

WHERE A.ID\_ARTIST =S.ID\_ARTIST AND T.ID\_TRACK=S.ID\_TRACK

AND T.ID\_MEDIUM=M.ID\_MEDIUM AND R.ID\_RELEASE = M.ID\_RELEASE

AND A.TYPE='Group'

GROUP BY A.ID\_ARTIST, **A.NAME**

ORDER BY count(DISTINCT **R.ID\_RELEASE**) DESC)

WHERE ROWNUM <=10

--E Print the name of female artist that have the most genres

SELECT NAME

FROM (SELECT A.NAME AS NAME,

COUNT(DISTINCT G.ID\_GENRE) AS COUNT\_GENRE

FROM ARTIST\_GENRE G, ARTIST A

WHERE A.ID\_ARTIST= G.ID\_ARTIST AND A.GENDER='Female'

GROUP BY A.ID\_ARTIST, A.NAME

ORDER BY COUNT\_GENRE DESC)

WHERE ROWNUM=1

--F Print the name of the cities that have more female artist than

--male artist

SELECT B.name

FROM Area B

WHERE B.type='City'

AND (SELECT Count(\*)

FROM Artist A

WHERE A.gender='Female' AND A.ID\_AREA= B.ID\_AREA)

     > (SELECT Count(\*)

FROM Artist A1

WHERE A1.gender='Male' AND A1.ID\_AREA=B.ID\_AREA)

--G List the mediums with the most number of tracks

SELECT MID

FROM

(SELECT M.ID\_MEDIUM AS MID,

COUNT(DISTINCT T.ID\_TRACK) AS COUNT\_TRACK

FROM TRACK T, MEDIUM M

WHERE M.ID\_MEDIUM = T.ID\_MEDIUM

GROUP BY M.ID\_MEDIUM)

WHERE COUNT\_TRACK = (SELECT MAX(COUNT\_TRACK)

FROM

(SELECT

COUNT(DISTINCT T.ID\_TRACK) AS COUNT\_TRACK

FROM TRACK T, MEDIUM M

WHERE M.ID\_MEDIUM=T.ID\_MEDIUM

GROUP BY M.ID\_MEDIUM)

);

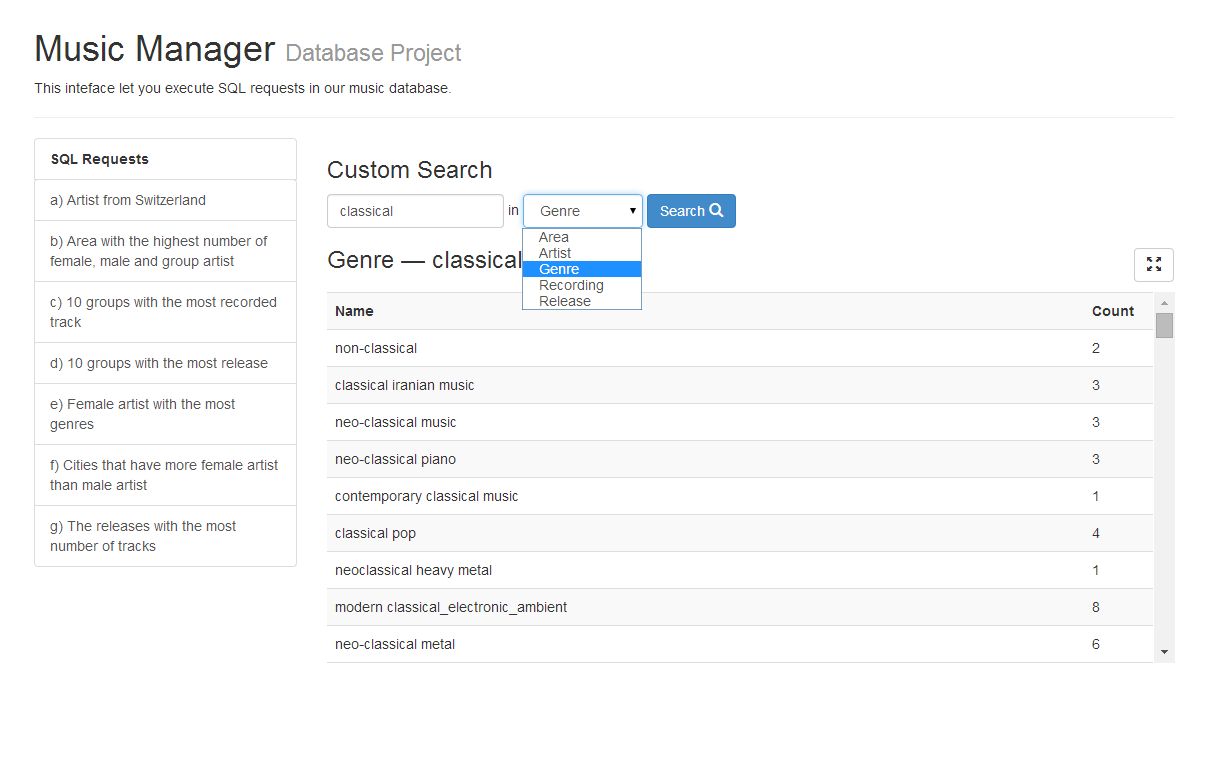
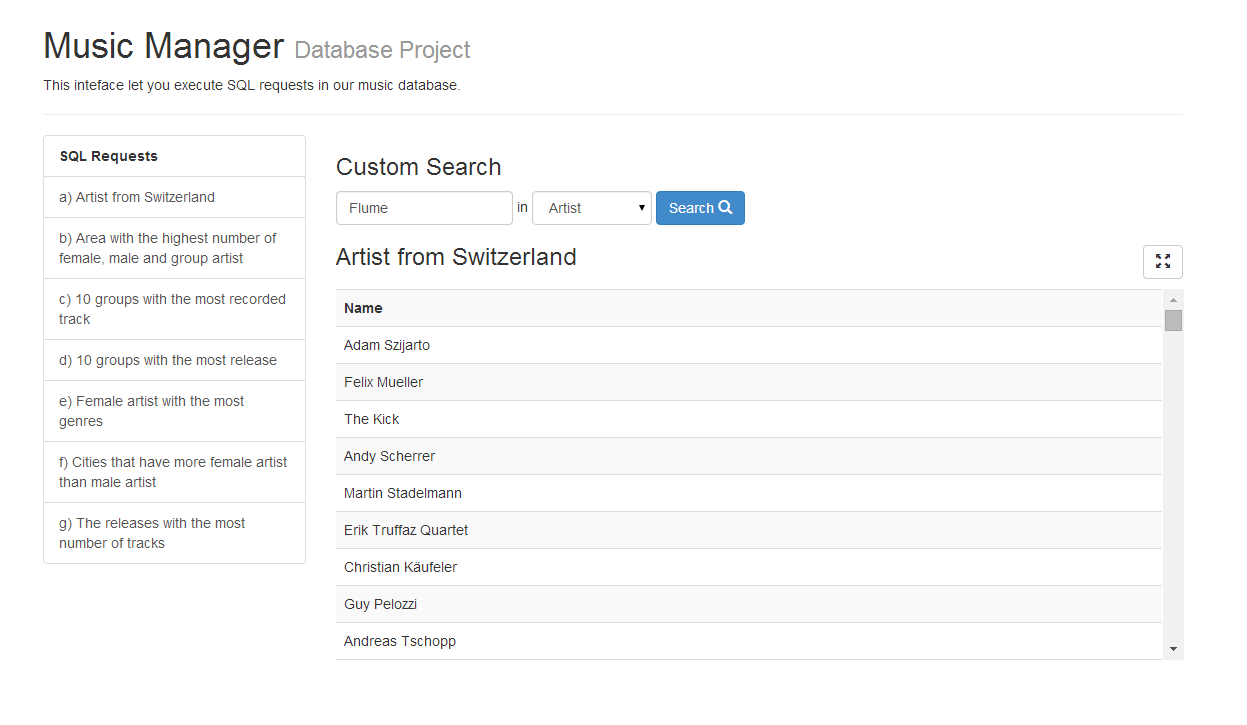
**Interface**

Figure 2: simple search: the user can choose in which table he wants to do the search

Figure 3 SQL queries: the left table allows the user to select a query and displays the result

****

Figure 4: the user clicked on the row of an artist and obtained different complementary information. Different information are available using the tabs

Figure 5: the user executed a search on an artist

Figure 6: Mode full screen of the results