

# Introduction To Database

---

## Project Report

Group 3 :

Joanna Salathé  
Eliéva Pignat  
Sacha Bron

This document contains the final report of the project. It contains all the subjects of the 3 deliverables

**Table des matières**

Modification of deliverable 2 .....	2
ER-Model .....	2
Constraints Explanation .....	3
Importation .....	3
Areas .....	3
Tracks .....	3
Recordings .....	3
Releases .....	3
Table Creation .....	4
Merge for participation constraints .....	4
SQL Code .....	4
Queries .....	6
Queries A to G .....	6
Queries H to S .....	13
Index performance .....	26
Importance of indexes based .....	26
Query Plans .....	26
Running Time .....	30
User Interface .....	31
Choices .....	31
Search Functionality .....	31
Server side .....	32
Why Java .....	32
Web server .....	32
Database Access .....	33
Client Side .....	34
Pre-written Requests .....	34
Custom Search .....	34
Clickable Results .....	34
Insertion .....	34
Deletion .....	34
Fullscreen Mode .....	34
Technical Details .....	35
Interface Screenshots .....	36

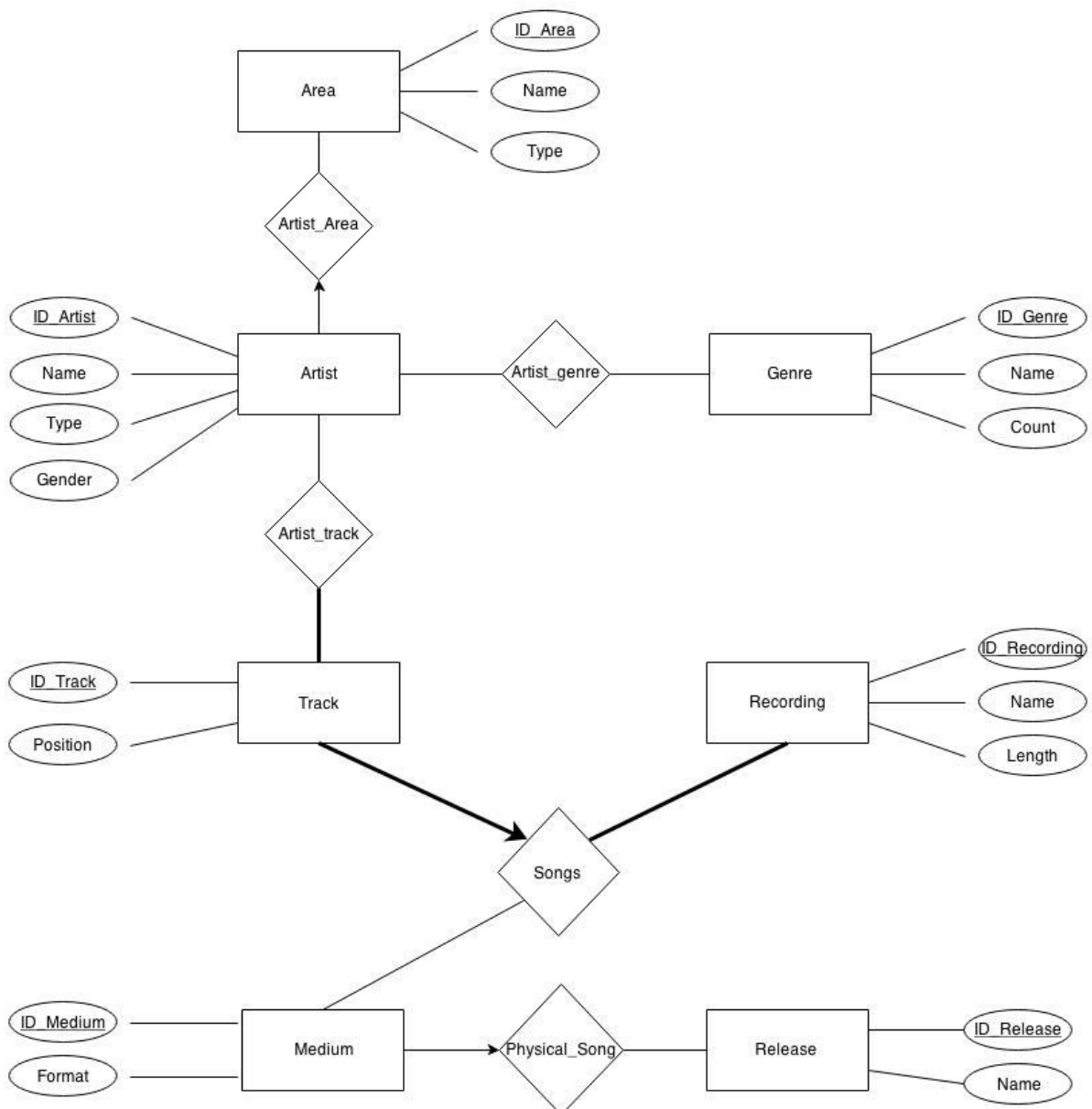
## Modification of deliverable 2

After the feedback, the content of the deliverable 2 was modified in the following way:

- 1) Some queries had to be modified and corrected. For example we add DISTINCT when we count. And we GROUP BY first by ID and after by NAME
- 2) The foreign key on artist of the artist\_track table was removed to be consistent with our database.
- 3) A short explanation of the queries A to G was added

## ER-Model

This schema fits best with the data structures..



## Constraints Explanation

There are some complex constraints between entities because the musical universe is also complex: collaborations (multiple artists per track), compilations (multiple artists per album), etc. But because of the reality of the data (for instance information is missed). A lot of constraints cannot be strong. Some constraints had to be weakened by deleting some FOREIGN KEY. Indeed, some constraints that stand on the schema 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.

## 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. The next chapter presents the ER-Model and the SQL code for the table creation. Some choices don't seem logic but are adapted to the available data. An explanation is given for those choices.

## Areas

An artist can have at most one area. This constraint is given by the available data. Indeed, instinctively we may think that one artist have to come from one or more than one area (for example: New-York (*City*) and America (*Continent*)). But in the given data, the situation is different: an artist can have only one area or no area if the data is missing.

## Tracks

Each track is unique. It contains only one recording and is on one medium exactly.

## Recordings

A recording must be on a track but can be on several different tracks. It makes no sense to have a recording that is not on a track (a logical song need a support, so participate to the song relationship).

## Releases

Because some data is missed, release can be associated with one or zero medium.

## Table Creation

### Merge 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.

### 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  
    ON DELETE CASCADE);
```

```
CREATE TABLE Artist_Genre  
(ID_Artist INT,  
ID_Genre INT,  
PRIMARY KEY(ID_Artist, ID_Genre),  
FOREIGN KEY(ID_Genre) REFERENCES Genre  
    ON DELETE CASCADE);
```

```
CREATE TABLE Artist_Track  
(ID_Artist INT,  
ID_Track INT,  
PRIMARY KEY(ID_Artist, ID_Track),  
FOREIGN KEY(ID_Artist) REFERENCES Artist  
    ON DELETE CASCADE);
```

## Queries

### Queries A to G

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, F. Thus we think that this problem comes from a lack of optimisation of those queries (too much join or too much nested loop).

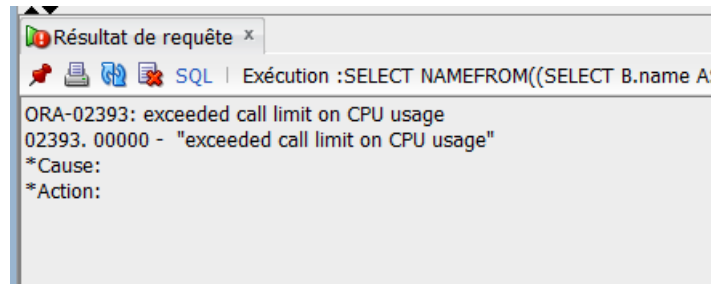


Figure 1: error when the query E is run during a long time

Query A :

--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'

NAME	NAME	NAME
1 Adam Szijarto	77 Protest	218 Charles Panzéra
2 Felix Mueller	78 Reto Ardour	219 Adolf Wölfli
3 The Kick	79 Franz Gisler	220 Marylane
4 Andy Scherrer	80 Manon	221 Housi Wittlin
5 Martin Stadelmann	81 Petia Kaufman	222 Brian Allen
6 Erik Truffaz Quartet	82 Polo Hofer	223 Susanna Hofmann
7 Christian Käufeler	83 Chœur de Chambre Romand	224 Barbara Wiss
8 Guy Pelozzi	84 Rudolf Rosen	225 Lisa Neurohr
9 Andreas Tschopp	85 So Wonderful	226 Patricia Mair
10 Peter Wagner	86 Gilbert Cotting	227 Susi Schaufelberger
11 Pascal Nägeli	87 BAK XIII	228 Andrea Schröder
12 Cenk	88 Jan Stehle	229 Robert Rüdisüli
13 Hugo Cotting	89 Christian Addor	230 Redeem
14 Morph	90 Mr. P!nk	231 Kolback
15 Michel Cleis	91 Michel	232 Brainless
16 Liz Stoussi	92 Dma-Sc	233 Pat Charvet
17 Cabaretduo Divertimento	93 Hazy Osterwald	234 Henri Huber
18 Cabaret Marcocello	94 Peeping Tom	235 Samael
19 Cabaret Rotstift	95 Hansruedi Egli	236 Vorph
20 Arsi Montsenigos	96 Klaus Widmer	237 Michael Locher
21 Myrto Joannidis	97 Dave Scherler	238 Frédéric Minuti
22 Duo Meiermoser	98 Willi Grimm	239 Xytras
23 Edgar Schmid	99 Cornelia Kraft	240 Alexandre Locher
24 Martin Schumacher	100 Anna Golob	241 Christophe Mermod
25 Christoph Gantert	101 Erwin Cotting	242 Yvan & Dan Daniel
26 Roman Camenzind	102 René Burri	243 Gran Purismo
27 Domenic Janett	103 Bill Weilenmann	244 Alois Betschart
28 Michel Willi	104 Yvan	245 Dezmond Dez
29 Joy Frempong	105 Eric Cotting	246 Serge Weber
30 Julian Sartorius	106 Dieter Meier	247 Full Lemon
31 Grauzone	107 Steine Für Den Frieden	248 Tommy Vercetti
32 Autopsy	108 Paul Fahm	249 Marc Erbetta
33 Walter Keiser	109 Cabaret Bärner Rohrs...	250 Heidi Happy

Figure 2 result of query A



## Query B :

--B print the name and the number of female, male and group of the  
--area that have the most female, male or group artists.  
/\*The first part of the query create a table with the following  
column : AREA| female count| male count| group count and in

```
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;
```

Query C :

--C List the name of 10 groups with the most recorded track  
/\*order by the number of tracks and select the top 10 for the  
group\*/

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

	NAME
1	U2
2	The Beatles
3	Grateful Dead
4	Nirvana
5	Pearl Jam
6	Metallica
7	Bob Marley & The Wailers
8	The Rolling Stones
9	Pink Floyd
10	Duke Ellington & His Orchestra

Figure 3 result of query C

Query D :

```
--D List the name of 10 groups with the most release
/*Order by the number of releases and select the top 10 for the
group*/
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
```

	NAME
1	U2
2	Depeche Mode
3	The Beach Boys
4	Queen
5	Scooter
6	Blondie
7	Kool & The Gang
8	The Prodigy
9	Duran Duran
10	Underworld

Figure 4 result of query D

Query E :

```
--E Print the name of female artist that have the most genres
/*Order by the number of genre of female artist and select the top 1
artist*/
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
```

	NAME
1	Cornelia Dahlgren

Figure 5 result of query E

Query F :

```
--F Print the name of the cities that have more female artist than
--male artist
/*Compare the count of the female artist and the count of male
artist for the city. And select the cites 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)
```

Query G :

```
--G List the mediums with the most number of tracks
/*Order the mediums by their track count and select the medium that
have their track count egal to the max track count.*/
SELECT ID, FORMAT, COUNT_TRACK
FROM
(SELECT M.ID_MEDIUM AS ID, M.FORMAT AS FORMAT,
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, M.FORMAT)
WHERE COUNT_TRACK = (SELECT MAX(COUNT_TRACK)
                     FROM
                     (SELECT M.ID_MEDIUM AS ID, M.FORMAT AS FORMAT,
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, M.FORMAT)
);
```

	MID
1	785558

Figure 6 result of query G

## Queries H to S

The queries H, M, N and S don't give us any results. They never end (we stop after more than 1 hour). But they work (terminate and give consistent result) when we compute them on our « mini » database (a dummy database with the same schema than the project database but with less entries).

We also try to reduce the number of joins (that was really big). But reducing the multiple join introduced nested loops or/and multiple views and the queries still don't give us any result.

For the others queries, you can find in this report the SQL codes, an explanation and (a part of) the results.

### Query H :

```
--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.
-- The query H does not stop
/*We do three different views. One to obtain the male artist, one
for the female and the last one for the group by areas.*/
```

```
SELECT A.NAME, M.ART_NAME, F.ART_NAME, G.ART_NAME
FROM AREA A,
(SELECT AREA_ID, ART_NAME
FROM
(SELECT AM.ID_AREA AS AREA_ID, AM.NAME AS ART_NAME,
COUNT(DISTINCT ATM.ID_TRACK) AS TCOUNT,
ROW_NUMBER()OVER (PARTITION BY AM.ID_AREA
ORDER BY COUNT(DISTINCT ATM.ID_TRACK)DESC) AS RN
FROM ARTIST AM, ARTIST_TRACK ATM
WHERE AM.GENDER='Male' AND AM.ID_ARTIST=ATM.ID_ARTIST
AND AM.ID_AREA IN (SELECT AR2.ID_AREA
FROM AREA AR2, ARTIST A
WHERE A.ID_AREA= AR2.ID_AREA
GROUP BY AR2.ID_AREA
HAVING COUNT(DISTINCT A.ID_ARTIST)>=30)
GROUP BY AM.ID_AREA, AM.ID_ARTIST, AM.NAME
)WHERE RN<=1) M,
(SELECT AREA_ID, ART_NAME
FROM
(SELECT AM.ID_AREA AS AREA_ID, AM.NAME AS ART_NAME,
COUNT(DISTINCT ATM.ID_TRACK) AS TCOUNT,
ROW_NUMBER()OVER (PARTITION BY AM.ID_AREA
ORDER BY COUNT(DISTINCT ATM.ID_TRACK)DESC) AS RN
```

```
FROM ARTIST AM, ARTIST_TRACK ATM
WHERE AM.GENDER='Female' AND AM.ID_ARTIST=ATM.ID_ARTIST
AND AM.ID_AREA IN (SELECT AR2.ID_AREA
                    FROM AREA AR2, ARTIST A
                    WHERE A.ID_AREA= AR2.ID_AREA
                    GROUP BY AR2.ID_AREA
                    HAVING COUNT(DISTINCT A.ID_ARTIST)>=30)
GROUP BY AM.ID_AREA, AM.ID_ARTIST, AM.NAME
)WHERE RN<=1) F,
(SELECT AREA_ID, ART_NAME
FROM
(SELECT AM.ID_AREA AS AREA_ID, AM.NAME AS ART_NAME,
      COUNT(DISTINCT ATM.ID_TRACK) AS TCOUNT,
      ROW_NUMBER()OVER (PARTITION BY AM.ID_AREA
      ORDER BY COUNT(DISTINCT ATM.ID_TRACK)DESC) AS RN
FROM ARTIST AM, ARTIST_TRACK ATM
WHERE AM.TYPE='Group' AND AM.ID_ARTIST=ATM.ID_ARTIST
AND AM.ID_AREA IN (SELECT AR2.ID_AREA
                    FROM AREA AR2, ARTIST A
                    WHERE A.ID_AREA= AR2.ID_AREA
                    GROUP BY AR2.ID_AREA
                    HAVING COUNT(DISTINCT A.ID_ARTIST)>=30)
GROUP BY AM.ID_AREA, AM.ID_ARTIST, AM.NAME
)WHERE RN<=1) G
WHERE A.ID_AREA=M.AREA_ID AND A.ID_AREA=G.AREA_ID
AND A.ID_AREA=F.AREA_ID;
```

## Query I :

```
-- I: American metal group Metallica is asking its fans to choose
-- the set list 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.
/*Order the recordings of Metallica by the count of their mediums
apparitions in the track table and select the top 25 of them */
```

```
SELECT DISTINCT *
FROM
(SELECT R.NAME, COUNT(DISTINCT T.ID_MEDIUM)
FROM TRACK T, RECORDING R, ARTIST_TRACK AT, ARTIST A
WHERE A.NAME = 'Metallica' AND A.ID_ARTIST = AT.ID_ARTIST
AND AT.ID_TRACK = T.ID_TRACK AND T.ID_RECORDING = R.ID_RECORDING
GROUP BY R.ID_RECORDING, R.NAME
ORDER BY COUNT(DISTINCT T.ID_MEDIUM) DESC)
WHERE ROWNUM <= 25;
```

	NAME
1	Whiplash
2	For Whom the Bell Tolls
3	Motorbreath
4	Seek & Destroy
5	Jump in the Fire
6	Hit the Lights
7	Welcome Home (Sanitarium)
8	Phantom Lord
9	Trapped Under Ice
10	The Call of Ktulu
11	Frantic
12	Fade to Black
13	Master of Puppets
14	Ride the Lightning
15	(Anesthesia)-Pulling Teeth
16	Metal Militia
17	Nothing Else Matters
18	Fight Fire With Fire
19	Escape
20	Battery
21	Creeping Death
22	The Four Horsemen
23	No Remorse
24	One
25	Damage, Inc.

Figure 7 result of query I



Query J :

```
--J : For each of the 10 genres with the most artists, list the
--female artist that has recorded the highest number of tracks.
/*First we order the genre by the count of their artists and select
the top 10 of them. Then for each genre that belongs to this subset
we check if it has female artists in it. If so we order its female
artists by counting the number of track they did with an over
partition operator and select the first row for each area.*/
```

```
SELECT GENRE_ID, NAME
FROM(
  SELECT AG.ID_GENRE AS GENRE_ID, A.NAME AS NAME,
    COUNT(DISTINCT AT.ID_TRACK),
    ROW_NUMBER()OVER (PARTITION BY AG.ID_GENRE
      ORDER BY COUNT(DISTINCT AT.ID_TRACK) DESC) AS RN
  FROM ARTIST A, ARTIST_TRACK AT, ARTIST_GENRE AG
  WHERE A.ID_ARTIST= AT.ID_ARTIST AND A.GENDER='Female'
  AND A.ID_ARTIST=AG.ID_ARTIST
  AND AG.ID_GENRE IN (SELECT *
    FROM
      (SELECT G.ID_GENRE
        FROM ARTIST_GENRE G
        GROUP BY G.ID_GENRE
        ORDER BY COUNT(DISTINCT G.ID_ARTIST) DESC)
    WHERE ROWNUM <=10)
  GROUP BY AG.ID_GENRE, A.ID_ARTIST, A.NAME
)WHERE RN=1;
```

	GENRE_ID	NAME
1	7	Tori Amos
2	75	Nina Simone
3	273	Madonna
4	1753	Kate Bush
5	3598	Nora Bumbiere
6	111	Ella Fitzgerald
7	171	J.K. Rowling
8	88	Avril Lavigne
9	237	J.K. Rowling
10	359	Rachel Portman

Figure 8 result of query J

## Query K :

--K : List all genres that have no female artists, all genres that  
--have no male artists and all genres that have no groups.  
/\*Select the genres that are not present in the result of the  
selection of genres that have female artist.  
Select the genres that are not present in the result of the  
selection of genres that have male artist.  
Select the genres that are not present in the result of the  
selection of genres that have group artist.  
Union the results of each of these sub-queries.\*/

```
SELECT G.ID_GENRE
FROM GENRE G
WHERE G.ID_GENRE
NOT IN (SELECT AG.ID_GENRE
        FROM ARTIST_GENRE AG, ARTIST A
        WHERE AG.ID_ARTIST = A.ID_ARTIST AND A.GENDER = 'Female')
UNION
SELECT G.ID_GENRE
FROM GENRE G
WHERE G.ID_GENRE
NOT IN (SELECT AG.ID_GENRE
        FROM ARTIST_GENRE AG, ARTIST A
        WHERE AG.ID_ARTIST = A.ID_ARTIST AND A.GENDER = 'Male')
UNION
SELECT G.ID_GENRE
FROM GENRE G
WHERE G.ID_GENRE
NOT IN (SELECT AG.ID_GENRE
        FROM ARTIST_GENRE AG, ARTIST A
        WHERE AG.ID_ARTIST = A.ID_ARTIST AND A.TYPE = 'Group') ;
```

ID_GENRE	ID_GENRE	ID_GENRE	ID_GENRE
1	2	244	523
2	4	245	524
3	6	246	525
4	17	247	526
5	21	248	531
6	24	249	532
7	28	250	533
8	31	251	534
9	32	252	536
10	33	253	538
11	34	254	541
		255	542
		256	543
		257	544
		258	547
		259	549
		260	550
		261	551
		262	562
		263	563
		264	565
		265	566
		266	572
		267	573
		268	574
		269	575
		270	576
		271	577
		272	578
		273	579
		274	580
		275	581
		276	582
		1469	2387
		1470	2388
		1471	2389
		1472	2390
		1473	2391
		1474	2392
		1475	2393
		1476	2394
		1477	2395
		1478	2396
		1479	2397
		1480	2398
		1481	2399
		1482	2400
		1483	2401
		1484	2405
		1485	2406
		1486	2408
		1487	2409
		1488	2410
		1489	2412
		1490	2413
		1491	2414
		1492	2415
		1493	2416
		1494	2417
		1495	2418
		1496	2419
		1497	2420
		1498	2422
		1499	2423
		1500	2424

Figure 9 result of query K

Query L :

```
--L : For each area with more than 10 groups, list the 5 male
--artists that have recorded the highest number of tracks.
/*Select the areas that have more than 10 groups and for each of
those areas we order the artist of this area by their count record
and select the 5 highest of each area */
```

```
SELECT AREA_NAME, ART_NAME
FROM
(SELECT ARE.NAME AS AREA_NAME, A.NAME AS ART_NAME,
COUNT(DISTINCT AT.ID_TRACK) AS TRACKCOUNT,
ROW_NUMBER()OVER(PARTITION BY ARE.ID_AREA, ARE.NAME
ORDER BY COUNT(DISTINCT AT.ID_TRACK) DESC) AS RN
FROM ARTIST A, ARTIST_TRACK AT, AREA ARE
WHERE A.ID_ARTIST = AT.ID_ARTIST AND A.ID_AREA=ARE.ID_AREA
AND ARE.ID_AREA IN
(SELECT AR.ID_AREA
FROM AREA AR, ARTIST A2
WHERE AR.ID_AREA = A2.ID_AREA AND A2.TYPE = 'Group'
GROUP BY AR.ID_AREA
HAVING COUNT(DISTINCT A2.ID_ARTIST) > 10)
GROUP BY ARE.ID_AREA, ARE.NAME, A.ID_ARTIST, A.NAME)
WHERE RN <= 5;
```

AREA_NAME	ART_NAME	AREA_NAME	ART_NAME	AREA_NAME	ART_NAME
1 Angola	Bonga Kuenda	69 Chile	Quilapayún	241 Latvia	Raimonds Pauls
2 Angola	Waldemar Bastos	70 Chile	ThePlasmas	242 Latvia	Prāta vētra
3 Angola	Sofia Rosa	71 China	alan	243 Latvia	Jumprava
4 Angola	Conjunto Ngonguenha	72 China	女子十二乐坊	244 Latvia	Līvi
5 Angola	Lilly Tchiumba	73 China	谭盾	245 Latvia	Iļģi
6 Argentina	Ástor Piazzolla	74 China	Nawang Khechog	246 Lebanon	Gabriel Yared
7 Argentina	Lalo Schifrin	75 China	甘雅丹	247 Lebanon	فيروز
8 Argentina	Carlos Gardel	76 Colombia	Shakira	248 Lebanon	مرسيل خليفة
9 Argentina	The Cherry Blues Project	77 Colombia	Joe Arroyo	249 Lebanon	Rabih Abou-Khalil
10 Argentina	Los Fabulosos Cadillacs	78 Colombia	Diomedes Diaz	250 Lebanon	Elissa
11 Australia	AC/DC	79 Colombia	Juanes	251 Lithuania	Skylė
12 Australia	Kylie Minogue	80 Colombia	Carlos Vives	252 Lithuania	Foje
13 Australia	Nick Cave & The Bad Seeds	81 Congo	Reddy Amisi	253 Lithuania	Andrius Mamontovas
14 Australia	INXS	82 Congo	Les Bantous de la Capitale	254 Lithuania	Sel
15 Australia	Dead Can Dance	83 Congo	Diblo Dibala	255 Lithuania	Mango
16 Austria	Wolfgang Amadeus Mozart	84 Congo	René Lokua	256 Luxembourg	Rome
17 Austria	Joseph Haydn	85 Congo	Kanyoka	257 Luxembourg	LUXUS
18 Austria	Franz Schubert	86 Costa Rica	Malpais	258 Luxembourg	Sun Glitters
19 Austria	Gustav Mahler	87 Costa Rica	Gandhi	259 Luxembourg	Camillo Felgen
20 Austria	Johann Strauss II	88 Costa Rica	Humberto Vargas	260 Luxembourg	Petrograd
21 Bangladesh	S.D. Burman	89 Costa Rica	Billy The Kid	261 Macedonia	Леб и сол
22 Bangladesh	Arnob	90 Costa Rica	Pseudostratified Epith...	262 Macedonia	Тоше Проески
23 Bangladesh	Miles	91 Croatia	Azra	263 Macedonia	Кочани Оркестар
24 Bangladesh	LRB	92 Croatia	Oliver Dragojević	264 Macedonia	Esma Redžepova
25 Bangladesh	Habib Wahid	93 Croatia	Parni valjak	265 Macedonia	Мизар
26 Belarus	Ляпис Трубецкой	94 Croatia	Arsen Dedić	266 Madagascar	Rajery
27 Belarus	Ambassador21	95 Croatia	Let 3	267 Madagascar	D'Gary
28 Belarus	Серебряная свадьба	96 Cuba	Celia Cruz	268 Madagascar	Rossy
29 Belarus	Садъ	97 Cuba	Pérez Prado	269 Madagascar	Tarika Sammy
30 Belarus	Віктар Шалкевіч	98 Cuba	Compay Segundo	270 Madagascar	Régis Gizavo
31 Belgium	Jacques Brel	99 Cuba	Mongo Santamaria	271 Malaysia	梁靜茹
32 Belgium	Front 242			272 Malaysia	光良

Figure 10 result of query L

Query M :

```
--M :List the 10 groups with the highest number of tracks that  
--appear on compilations. A compilation is a medium that contains  
--tracks associated with more than one artist.
```

```
--Query M does not stop
```

```
/*Count the number of tracks of an artist if the track is on a  
medium that contains at least another track create by another  
artist. Order the artists by the number of those track and finally  
select the top 10*/
```

```
SELECT *
```

```
FROM
```

```
(SELECT A.NAME AS NAME, COUNT(DISTINCT T.ID_TRACK)
```

```
FROM ARTIST A, ARTIST_TRACK AT, TRACK T
```

```
WHERE A.TYPE='Group' AND A.ID_ARTIST=AT.ID_ARTIST
```

```
AND AT.ID_TRACK=T.ID_TRACK AND T.ID_MEDIUM
```

```
IN (SELECT DISTINCT T2.ID_MEDIUM
```

```
FROM ARTIST_TRACK AT2, TRACK T2, ARTIST_TRACK AT3, TRACK T3
```

```
WHERE AT2.ID_TRACK=T2.ID_TRACK AND AT3.ID_TRACK =T3.ID_TRACK
```

```
AND T2.ID_TRACK <>T3.ID_TRACK AND AT3.ID_ARTIST <>AT2.ID_ARTIST
```

```
AND T2.ID_MEDIUM=T3.ID_MEDIUM)
```

```
GROUP BY A.ID_ARTIST, A.NAME
```

```
ORDER BY COUNT(DISTINCT T.ID_TRACK)DESC
```

```
)WHERE ROWNUM<=10;
```

Query N :

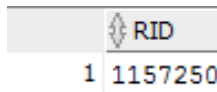
--N :List the top 10 releases with the most collaborations, i.e.,  
--releases where one artist is performing all songs and the highest  
--number of different guest artists contribute to the album.  
--Query N : does not give a result. There is big joins and then the  
--computation is too much heavy and slow

```
SELECT *
FROM
(SELECT B.RELEASE_ID, C.COUNT_ART
FROM
  --# OF TRACKS PER ARTIST PER RELEASE
  (SELECT M.ID_RELEASE AS RELEASE_ID,
    COUNT(DISTINCT T.ID_TRACK) AS COUNT_TRACK
  FROM ARTIST A, TRACK T, MEDIUM M, ARTIST_TRACK AT
  WHERE A.ID_ARTIST=AT.ID_ARTIST AND AT.ID_TRACK=T.ID_TRACK
  AND T.ID_MEDIUM=M.ID_MEDIUM
  GROUP BY M.ID_RELEASE, A.ID_ARTIST) A,
  --# OF TRACK PER RELEASE
  (SELECT M.ID_RELEASE AS RELEASE_ID,
    COUNT(DISTINCT T.ID_TRACK)AS COUNT_TRACK
  FROM MEDIUM M, TRACK T
  WHERE T.ID_MEDIUM=M.ID_MEDIUM
  GROUP BY M.ID_RELEASE) B,
  --# OF DIFFERENT ARTISTS PER RELEASE
  (SELECT M.ID_RELEASE AS RELEASE_ID,
    COUNT(DISTINCT A.ID_ARTIST)AS COUNT_ART,
    COUNT(DISTINCT T.ID_TRACK) AS COUNT_TRACK
  FROM ARTIST A, TRACK T, MEDIUM M, ARTIST_TRACK AT
  WHERE A.ID_ARTIST=AT.ID_ARTIST
  AND AT.ID_TRACK=T.ID_TRACK AND T.ID_MEDIUM=M.ID_MEDIUM
  GROUP BY M.ID_RELEASE) C
WHERE B.RELEASE_ID=A.RELEASE_ID
AND A.COUNT_TRACK=B.COUNT_TRACK AND B.RELEASE_ID=C.RELEASE_ID
GROUP BY B.RELEASE_ID, C.COUNT_ART
ORDER BY C.COUNT_ART DESC
)WHERE ROWNUM<=10;
```

Query O :

```
--O : List the release which is associated with the most mediums. If
--there are more than one such release, list all such releases.
/*For each release we count the number of medium they are associated
to and we order these releases by this count.
Then we inject this result table in one from of another query. In
this outer query we check for all releases if the count is equal to
the max count of all releases.*/
--The code given here is not the code use for testing the index
--performance but it is a new implementation that is really more
--fast
```

```
SELECT RID
FROM
(SELECT M.ID_RELEASE AS RID, COUNT(DISTINCT M.ID_MEDIUM) AS COUNTM
FROM MEDIUM M
GROUP BY M.ID_RELEASE
ORDER BY COUNT(DISTINCT M.ID_MEDIUM) DESC)
WHERE COUNTM = (SELECT MAX(COUNTM)
FROM
(SELECT COUNT(DISTINCT M.ID_MEDIUM) AS COUNTM
FROM MEDIUM M
GROUP BY M.ID_RELEASE
ORDER BY COUNT(DISTINCT M.ID_MEDIUM) DESC));
```



	RID
1	1157250

Figure 11 result of query O

Query P :

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

/\*We select the groups that are associated with at least 3 genres and count the number of total artist for the genre associated with the previous artists selection .\*/

```
SELECT GNAME
FROM
(SELECT G.NAME AS GNAME
 FROM GENRE G, ARTIST_GENRE AG
 WHERE G.ID_GENRE=AG.ID_GENRE AND AG.ID_ARTIST IN
      (SELECT AG.ID_ARTIST
       FROM ARTIST A, ARTIST_GENRE AG
       WHERE A.TYPE = 'Group' AND AG.ID_ARTIST = A.ID_ARTIST
       GROUP BY AG.ID_ARTIST
       HAVING COUNT(DISTINCT AG.ID_GENRE)>=3)
 GROUP BY G.ID_GENRE, G.NAME
 ORDER BY COUNT(DISTINCT AG.ID_ARTIST) DESC
)WHERE ROWNUM <= 1;
```

GNAME
1 punk

Figure 12 result of query P

Query Q:

-- Q : List the 5 titles that are associated with the most different --songs (recordings) along with the number of songs that share such --title.

/\*We select the name of the titles and for each of them we count the number of recording that has this name and we order them by this count. Finally we select the top 5 of them.\*/

```
SELECT *
FROM
(SELECT R.NAME, COUNT(R.ID_RECORDING)
 FROM RECORDING R
 GROUP BY R.NAME
 ORDER BY COUNT(R.ID_RECORDING) DESC)
WHERE ROWNUM <=5;
```

	NAME	COUNT(R.ID_RECORDING)
1	[untitled]	42889
2	Intro	19441
3	[silence]	8581
4	[unknown]	7830
5	Outro	4881

Figure 13 result of query Q



Query R :

```
--R : List the top 10 artists according to their track-to-release
--ratio. This ratio is computed by dividing the number of tracks an
--artist is associated with by the number of releases this artist
--as contributed a track to.
/*We create 2 views. The first one with the count of tracks of each
artist and the second one with the count of releases per artist and
then compute the ratio and select the top 10.*/
```

```
SELECT *
FROM
(SELECT A.NAME, COUNT_TRACK/COUNT_RELEASE AS RATIO
FROM
(SELECT AT.ID_ARTIST AS ART_ID,
      COUNT(DISTINCT AT.ID_TRACK) AS COUNT_TRACK
FROM ARTIST_TRACK AT
GROUP BY AT.ID_ARTIST) TR,
(SELECT AT.ID_ARTIST AS ART_ID,
      COUNT(DISTINCT M.ID_RELEASE) AS COUNT_RELEASE
FROM ARTIST_TRACK AT, TRACK T, MEDIUM M
WHERE AT.ID_TRACK=T.ID_TRACK AND T.ID_MEDIUM=M.ID_MEDIUM
GROUP BY AT.ID_ARTIST) ME,
ARTIST A
WHERE A.ID_ARTIST=TR.ART_ID AND A.ID_ARTIST =ME.ART_ID
ORDER BY RATIO DESC
)WHERE ROWNUM <=10;
```

	NAME	RATIO
1	Zondervan Publishing	1471
2	Charles Taylor	1272
3	Mary Roach	946
4	Jack Higgins	887
5	Susan La Rosa Maehre	861
6	David Timson	844
7	Groove Addicts	686
8	Arthur C. Clarke	657
9	DJ Epic	630
10	David Weber	615.5

Figure 14 result of query R

Query S :

```
--S :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".
--S does not stop
/*select songs and associated artist that appear on more than 100
medium and then compute the average for artist that have at least 10
song. This don't works because of the big join (artist_track,
recording and track) .*/
```

```
SELECT ART_ID, AVG(COUNTM)
FROM
(SELECT AT.ID_ARTIST AS ART_ID, R.NAME AS RECORD_NAME,
      COUNT(DISTINCT T.ID_MEDIUM) AS COUNTM
 FROM ARTIST_TRACK AT, RECORDING R, TRACK T
 WHERE T.ID_RECORDING=R.ID_RECORDING AND AT.ID_TRACK=T.ID_TRACK
 GROUP BY AT.ID_ARTIST, R.NAME
 HAVING COUNT(DISTINCT T.ID_MEDIUM)>=100
 )GROUP BY ART_ID
 HAVING COUNT(RECORD_NAME)>=10;
```

## Index performance

### Importance of indexes based

As seen during the lessons, the indexes can considerably improve the time requested by a query. And it is more important in our case when we come with big data. Indeed, some queries take several minutes to be executed. We can win significant amount of time. The indexes are a trade-off between time and space. More precisely, we use space to stock the indexes but we win time of execution. Thus we cannot add indexes for everything. We have to choose carefully the indexes that we need.

The Oracle Optimizer determines the most efficient query plan. It chooses the plan with the lowest cost. The plan query can help to choose indexes. We can see how the time is distributed and with this information we can see how and where we can win execution time.

### Query Plans

The selected queries are the queries : I, J and O.

Plan for I :

We add an index for the artist name because it must find all Metallica's tracks.

Here is the plan before the indexes :

Id	Operation	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT	25	525	1172 (4)	00 :00 :15
1	HASH UNIQUE	25	525	1172 (4)	00 :00 :15
2	COUNT STOPKEY				
3	VIEW	47	987	1171 (4)	00 :00 :15
4	SORT ORDER BY STOPKEY	47	3572	1171 (4)	00 :00 :15
5	HASH GROUP BY	47	3572	1171 (4)	00 :00 :15
6	NESTED LOOPS	47	3572	1169 (4)	00 :00 :15
7	NESTED LOOPS	47	2303	1075 (4)	00 :00 :13
8	NESTED LOOPS	50	1550	978 (4)	00 :00 :12
9	TABLE ACCESS FULL	1	21	976 (4)	00 :00 :12
10	INDEX RANGE SCAN	47	470	2 (0)	00 :00 :01
11	TABLE ACCESS BY INDEX ROWID	1	18	2 (0)	00 :00 :01
12	INDEX UNIQUE SCAN	1		1 (0)	00 :00 :01
13	TABLE ACCESS BY INDEX ROWID	1	27	2 (0)	00 :00 :01
14	INDEX UNIQUE SCAN	1	1	1 (0)	00 :00 :01

Predicate Information (identified by operation id):

```
-----
2 - filter(ROWNUM<=25)
4 - filter(ROWNUM<=25)
9 - filter("A"."NAME"='Metallica')
10 - access("A"."ID_ARTIST"="AT"."ID_ARTIST")
12 - access("AT"."ID_TRACK"="T"."ID_TRACK")
14 - access("T"."ID_RECORDING"="R"."ID_RECORDING")
```

Here is the plan of the query I after the indexing:

Id	Operation	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT	25	525	201 (2)	00 :00 :03
1	HASH UNIQUE	25	525	201 (2)	00 :00 :03
2	COUNT STOPKEY				
3	VIEW	47	987	200 (1)	00 :00 :03
4	SORT ORDER BY STOPKEY	47	3572	200 (1)	00 :00 :03
5	HASH GROUP BY	47	3572	200 (1)	00 :00 :03
6	NESTED LOOPS	47	3572	198 (0)	00 :00 :03
7	NESTED LOOPS	47	2303	104 (0)	00 :00 :02
8	NESTED LOOPS	50	1550	7 (0)	00 :00 :01
9	TABLE ACCESS BY INDEX ROWID	1	21	5 (0)	00 :00 :01
10	INDEX RANGE SCAN	1		3 (0)	00 :00 :01
11	INDEX RANGE SCAN	47	470	2 (0)	00 :00 :01
12	TABLE ACCESS BY INDEX ROWID	1	18	2 (0)	00 :00 :01
13	INDEX UNIQUE SCAN	1		1 (0)	00 :00 :01
14	TABLE ACCESS BY INDEX ROWID	1	27	2 (0)	00 :00 :01
15	INDEX UNIQUE SCAN	1		1 (0)	00 :00 :01

Predicate  
Informatio

n (identified by operation id):

- 
- 2 - filter(ROWNUM<=25)
  - 4 - filter(ROWNUM<=25)
  - 10 - access("A"."NAME"='Metallica')
  - 11 - access("A"."ID\_ARTIST"="AT"."ID\_ARTIST")
  - 13 - access("AT"."ID\_TRACK"="T"."ID\_TRACK")
  - 15 - access("T"."ID\_RECORDING"="R"."ID\_RECORDING")

We can observe that the expected time and CPU use is really lower with the indexes.

For the queries O and J the improvement is not significant we cannot see it with the plan because it cannot use the index instead of doing full table scan.

Plan of query J :

Id	Operation	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT	37800	18M	3809 (4)	00 :00 :46
1	VIEW	37800	18M	3809 (4)	00 :00 :46
2	WINDOW SORT PUSHED RANK	37800	2030K	3809 (4)	00 :00 :46
3	SORT GROUP BY	37800	2030K	3809 (4)	00 :00 :46
4	NESTED LOOP	37800	2030K	2776 (4)	00 :00 :34
5	HASH JOIN	797	35865	1179 (9)	00 :00 :15
6	HASH JOIN	797	9564	191 (28)	00 :00 :03
7	VIEW	10	40	115 (40)	00 :00 :02
8	COUNT STOPKEY				
9	VIEW	1725	6900	115 (40)	00 :00 :02
10	SORT ORDER BY STOPKEY	1725	13800	115 (40)	00 :00 :02
11	SORT GROUP BY	1725	13800	115 (40)	00 :00 :02
12	TABLE ACCESS FULL	137K	1074K	74 (7)	00 :00 :01
13	TABLE ACCESS FULL	137K	1074K	74 (7)	00 :00 :01
14	TABLE ACCESS FULL	217K	8759K	982 (5)	00 :00 :12
15	INDEX RANGE SCAN	47	470	2(0)	00 :00 :01

Predicate Information (identified by operation id):

- 
- 1 - filter("RN"=1)
  - 2 - filter(ROW\_NUMBER() OVER ( PARTITION BY "AG"."ID\_GENRE" ORDER BY COUNT(DISTINCT "AT"."ID\_TRACK") DESC )<=1)
  - 5 - access("A"."ID\_ARTIST"="AG"."ID\_ARTIST")
  - 6 - access("AG"."ID\_GENRE"="ID\_GENRE")
  - 8 - filter(ROWNUM<=10)
  - 10 - filter(ROWNUM<=10)
  - 14 - filter("A"."GENDER"='Female')
  - 15 - access("A"."ID\_ARTIST"="AT"."ID\_ARTIST")

The part that we would like to reduce is the full table access that is done on the artist\_track and artist tables. There is an index on the artist gender and on the artist\_genre for the id\_genre.

But the new query plan doesn't select the new index. It may be that in this case the use of the index cost more. Even the query seems to be slightly slower. It may be because the optimizer must compute more possibilities than without indexes and finally choose the same plan than before.

Plan of query O :

Id	Operation	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT	923K	22M	15844 (4)	00 :03 :11
1	VIEW	923K	22M	7922 (4)	00 :01 :36
2	SORT ORDER BY	923K	10M	7922 (4)	00 :01 :36
3	SORT GROUP BY NOSORT	923K	10M	7922 (4)	00 :01 :36
4	MERGE JOIN	1047K	11M	7922 (4)	00 :01 :36
5	INDEX FULL SCAN	1236K	7243K	3667 (2)	00 :00 :45
6	SORT JOIN	1044K	6121K	4255 (6)	00 :00 :52
7	TABLE ACCESS FULL	1044K	6121K	716 (6)	00 :00 :09
8	SORT AGGREGATE	1	13		
9	VIEW	923K	11M	7922 (4)	00 :01 :36
10	SORT GROUP BY	923K	10M	7922 (4)	00 :01 :36
11	MERGE JOIN	1047K	11M	7922 (4)	00 :01 :36
12	INDEX FULL SCAN	1236K	7243K	3667 (2)	00 :00 :45
13	SORT JOIN	1044K	6121K	4255 (6)	00 :00 :52
14	TABLE ACCESS FULL	1044K	6121K	716 (6)	00 :00 :09

Predicate Information (identified by operation id):

```

-----
1 - filter("COUNTM"= (SELECT MAX("COUNTM") FROM (SELECT COUNT(*) "COUNTM" FROM
    "MEDIUM" "M","RELEASE" "R" WHERE "M"."ID_RELEASE"="R"."ID_RELEASE" GROUP BY
    "R"."ID_RELEASE") "from$_subquery$_004"))
6 - access("R"."ID_RELEASE"="M"."ID_RELEASE")
    filter("R"."ID_RELEASE"="M"."ID_RELEASE")
13 - access("M"."ID_RELEASE"="R"."ID_RELEASE")
    filter("M"."ID_RELEASE"="R"."ID_RELEASE")

```

We must access the full medium table twice for the id\_release. Thus we add an index on the id\_release on the medium table but the new plan does not use it. We can suppose that the use of the index cost more.

### Running Time

The running time can depend of random factors like the number of people that use the server at the same time.

A : 5.026 seconds

B : no end

C : 15 seconds

D : 568 seconds

E : 0.567 seconds

F : no end

G : 838 seconds

H : no end

I : 30.157 seconds before added indexes, 0.4 seconds after.

J : 18.64 seconds before added indexes, 19.223 seconds (not better) after.

K : 2.09 seconds

L : 2162.6 seconds

M : no end

N : no end

O : 109.714 seconds before added indexes, 108 seconds after.

The query O was reimplemented in a more performed way : now the execution time is : 23 seconds

P : 2.59 seconds

Q : 249.877 seconds

R : 5175 seconds

S : no end

## User Interface

The user interface is one of the big part of the project. We had a lot of choices to make during the development of the interface.

This section will explain how the interface work and why we made those choices.

## Choices

First of all, we choose to develop the user interface as Web App because it's an easy way to design the interface we want.

## 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 + "%')"
```
- 4) 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.
- 5) 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.
- 6) 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?”

- 7) 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.

## Server side

The server is a simple multi-threaded HTTP server that also accesses the EPFL's Oracle Database.

## Why Java

We choose to code it in Java because it's almost the only language we all know well. Another advantage is that it handles pretty well Oracle's database with the JDBC lib. Furthermore, Java is powered by Oracle too. It had to be well supported and it is.

## Web server

The web server is pretty simple. It just handles basic HTTP GET requests.

When the server is running, a listener is called. This listener wait for a HTTP request and, when a request is caught, it spawns a worker new thread. Then, it waits for the next request.

The Worker is a little bit more complicated.

First of all, the worker will handle the stream of data coming on the port 80. It will extract the path and GET parameters from the header.

Then, it will try to open the file gave in the path and stream it to the client in the response.

The MIME type is automatically detected and sent in the HTTP response headers.

If the requested path is `/do-sql`, the behaviour will be different. The worker will try to extract a SQL query from the GET parameters. Then, it will ask the Database handler to execute the SQL query and to return a JSON with the result (or an error).

## Database Access

The database access is handled by a **Database** class.

It manages the connection and can execute some queries on the Oracle database. To achieve that, we use the JDBC driver.

After executing a query, the response from the Oracle database will be transformed in a JSON.

For instance, if you search an artist called "Flume", the response sent to the client will be:

```
{
  "status": "OK",
  "data": [
    {
      "NAME": "Liza Flume",
      "GENDER": "Female",
      "ID_AREA": 0,
      "ID_ARTIST": 1057910,
      "TYPE": "Person"
    },
    {
      "NAME": "Flume",
      "GENDER": "Male",
      "ID_AREA": 13,
      "ID_ARTIST": 835335,
      "TYPE": "Person"
    },
    {
      "NAME": "The Flumes",
      "GENDER": "Other",
      "ID_AREA": 0,
      "ID_ARTIST": 837196,
      "TYPE": "Other"
    },
    {
      "NAME": "DJ Michael Flume",
      "GENDER": "Other",
      "ID_AREA": 0,
      "ID_ARTIST": 195126,
      "TYPE": "Other"
    }
  ]
}
```

## Client Side

The client is a web app. So, to access it, you must first, run the java server, and then, access `http://localhost:7123/`

The interface is pretty simple and user-friendly. This chapter contains a description of every features of the interface. And the next chapter contains some screenshots of the interface.

## Pre-written Requests

Pre-written queries are easily runnable by clicking on buttons on the left of the interface.

Once a button is clicked, the corresponding SQL query is sent to the server and a JSON is returned as a response. This response will be show as a table in the center of the user interface.

## Custom Search

The custom search allows you to search things with a case-insensitive keyword. For instance, if you search ``beetroots``, the result ``The Bloody Beetroots`` will appear.

The request behind it is pretty simple:

```
SELECT * FROM artist WHERE LOWER(name) LIKE lower('%beetroots%').
```

Note that characters like ``%`` (percent), ``_`` (underscore) and `` `` (single quote) are escaped.

You can refer to the Search Functionality chapter for more details on the choices of this implementation.

## Clickable Results

When a non-empty result is displayed, each row can be clicked to show more information about the chosen result.

When the row is clicked, some tabs will appear to display new results.

You can refer to the Search Functionality chapter for more details

## Insertion

For each table displayed, an insertion form will appear. This allows you to add some more data in the database.

If a field is left empty, the value will be `null`.

Note that the line will not directly appear in the already displayed table for technical reason (the ID has to be defined by the database so the row can be clickable).

## Deletion

To delete a data, just click on the red cross on the right of the row. The deletion can be do after a search query (can delete a row).

## Fullscreen Mode

Above the results, on the right, a fullscreen button can be clicked to display results on a bigger view. This is pretty practical when the table has a lot of row.

### Technical Details

When a button is clicked, when a search is done or when a row is clicked, the same function is called to render the table. This function will execute an **AJAX** request to the server and get a **JSON** containing the result. The result will then be transformed to a **HTML** table.

Note that technical information like references to other **SQL** tables will be hidden to the user.

When errors occur, there are logged in the **Debug Console** at the bottom of the page. This is useful if you are disconnected from the **EPFL** network for instance.

## Interface Screenshots

### Music Manager Database Project

This interface let you execute SQL requests in our music database.

**SQL Requests**

- A) Artist from Switzerland
- B) Area with the highest number of female, male and group artist
- C) 10 groups with the most recorded track
- D) 10 groups with the most release
- E) Female artist with the most genres
- F) Cities that have more female artist than male artist
- G) The releases with the most number of tracks
- H) For each area with more than 30 artists list the male, female and group with highest tracks
- I) 25 most famous track of metallica

**Custom Search**

in
Artist

Search: Artist — dark

Name	Gender	Type	
Afterdark Inc.	Other	Group	✕
The Darkhouse Family	Other	Other	✕
Deities of Darkness	Other	Group	✕
Dark Denim	Other	Other	✕
Darkvizion	Other	Other	✕
DARK UNDERCOVER	Other	Person	✕
Dark Orchid	Other	Group	✕
Darkjet	Other	Group	✕
Darkness Profanum	Other	Group	✕

Figure 15: After a Search there is the possibility to delete rows with the red cross

### Music Manager Database Project

This interface let you execute SQL requests in our music database.

**SQL Requests**

- A) Artist from Switzerland
- B) Area with the highest number of female, male and group artist
- C) 10 groups with the most recorded track
- D) 10 groups with the most release
- E) Female artist with the most genres
- F) Cities that have more female artist than male artist
- G) The releases with the most number of tracks
- H) For each area with more than 30 artists list the male, female and

**Custom Search**

in
Artist

Search: Artist — flume

Name	Gender	Type	
Liza Flume	Female	Person	✕
Flume	Male	Person	✕
The Flumes	Other	Other	✕
DJ Michael Flume	Other	Other	✕

**Insert New Data**

Figure 16: Data can be insert in a Table

Name	Gender	Type	
Liza Flume	Female	Person	✕
Flume	Male	Person	✕
The Flumes	Other	Other	✕
DJ Michael Flume	Other	Other	✕

Figure 17: Fullscreen Mode

## Music Manager Database Project

This interface let you execute SQL requests in our music database.

**SQL Requests**

A) Artist from Switzerland

B) Area with the highest number of female, male and group artist

C) 10 groups with the most recorded track

Custom Search

in 

Area

Search

Output

The table is empty. There is no data.

Figure 18: Search functionality

## Music Manager Database Project

This interface let you execute SQL requests in our music database.

**SQL Requests**

a) Artist from Switzerland

b) Area with the highest number of female, male and group artist

c) 10 groups with the most recorded track

d) 10 groups with the most release

e) Female artist with the most genres

f) Cities that have more female artist than male artist

g) The releases with the most number of tracks

Custom Search

in 

Artist

Search

Artist — Flume

Area

Genre

Release

Name	Type
Australia	Country

Figure 19: Find more information by clicking on a row

## Music Manager Database Project

This interface let you execute SQL requests in our music database.

**SQL Requests**

a) Artist from Switzerland

b) Area with the highest number of female, male and group artist

c) 10 groups with the most recorded track

d) 10 groups with the most release

e) Female artist with the most genres

f) Cities that have more female artist than male artist

g) The releases with the most number of tracks

Custom Search

in 

Artist

Search

Artist from Switzerland

Name

Adam Sziarto

Felix Mueller

The Kick

Andy Scherrer

Martin Stadelmann

Erik Truffaz Quartet

Christian Käufeler

Guy Pelozzi

Andreas Tschopp

Figure 20: Left buttons does SQL queries