

CS-322 Introduction to Database Systems

Project Deliverable #3

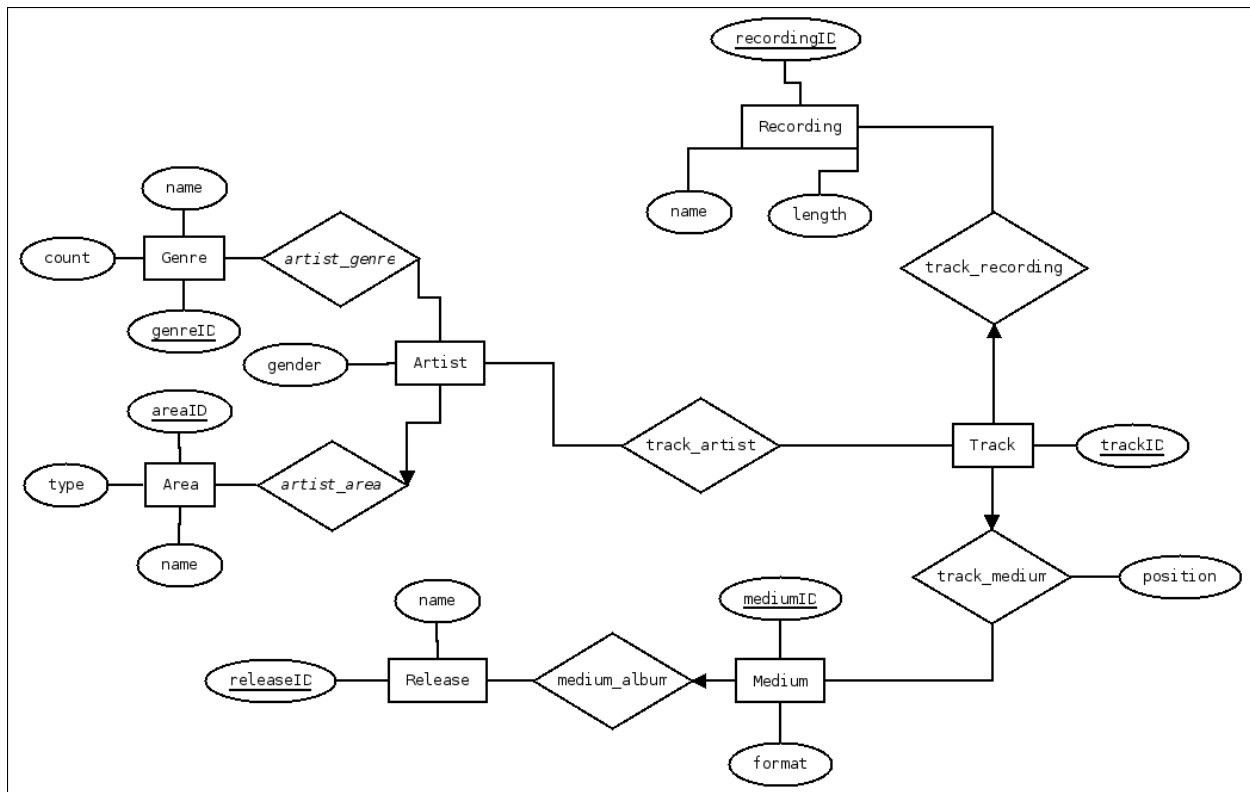
Due on Tuesday, June 3rd, 2014

Group 24
Klay, Ameho, Vinh Mau



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ER model for music database



SQL DDL code for table creations

```
CREATE TABLE Areas (
  areaID INTEGER,
  name VARCHAR(500) NOT NULL,
  type VARCHAR(255),
  PRIMARY KEY (areaID) );
```

```
CREATE TABLE Genres (
  genreID INTEGER,
  name VARCHAR(2000) NOT NULL,
  count INTEGER DEFAULT 0,
  PRIMARY KEY (genreID) );
```

```
CREATE TABLE Artists (
  artistID INTEGER,
  name VARCHAR(2000) NOT NULL,
  areaID INTEGER,
  gender VARCHAR(255),
  type VARCHAR(255),
  PRIMARY KEY (artistID),
  FOREIGN KEY (areaID) REFERENCES Areas );
```

```
CREATE TABLE Recordings (
```

```
25      recordingID INTEGER,
      name VARCHAR(4000) ,
      length INTEGER,
      PRIMARY KEY (recordingID) ) ;

30

CREATE TABLE Releases (
      releaseID INTEGER,
      name VARCHAR(2000) NOT NULL,
      PRIMARY KEY (releaseID) ) ;

35

CREATE TABLE Mediums (
      mediumID INTEGER,
      releaseID INTEGER,
      format VARCHAR(255),
40      PRIMARY KEY (mediumID),
      FOREIGN KEY (releaseID) REFERENCES Releases ) ;

CREATE TABLE Tracks (
45      trackID INTEGER,
      mediumID INTEGER,
      recordingID INTEGER,
      position INTEGER,
      PRIMARY KEY (trackID),
50      FOREIGN KEY (mediumID) REFERENCES Mediums,
      FOREIGN KEY (recordingID) REFERENCES Recordings ) ;
```

SQL script for entities table creation

```
CREATE TABLE Artist_genre (
      artistID INTEGER,
      genreID INTEGER,
      PRIMARY KEY (artistID, genreID),
5      FOREIGN KEY (artistID) REFERENCES Artists,
      FOREIGN KEY (genreID) REFERENCES Genres ) ;

CREATE TABLE Track_artist (
10      artistID INTEGER,
      trackID INTEGER,
      PRIMARY KEY (artistID, trackID),
      FOREIGN KEY (trackID) REFERENCES Tracks ,
      FOREIGN KEY (artistID) REFERENCES Artists ) ;
```

SQL script for relations table creation

Design choices & data constraints

There are three main concepts in our music database : **Song**, **Artist** and **Album**. Both Song and Album were divided between their descriptive data (**Recording**, **Release**) and their physical incarnation (**Track**, **Medium**). Since data is often incomplete, most of the entities 'can be related' but do not have to. We put a NOT NULL constraint on most of the name attributes of the entities, with the exception of **Recording** for the reason just stated. Since they are not required fields to describe music, they should have a valid name when they are in fact used.

- A **Track** is related to:

Recording: A track can be a physical incarnation of a known recording.

Artist: A track can exist without known artists, but can also have several artists to describe collaborations.

Medium: A track can be recorded on some medium. Their relation is characterized by the track position on the medium.

- An **Artist** is defined by a:

Genre: A genre can regroup multiple artists, whereas an artist can be difficult to define as catering to a specific genre, or crossing boundaries between genres nullifying the need for a constraint. We kept the count attribute, choosing small update costs over on-demand higher computation costs.

Area: An artist's location can be pinpointed to a specific creation grounds, hence can be expressed by a foreign key constraint. But several artists can be compelled to share their musical feelings in the same studio.

- A **Release** is the logical aggregation of songs, labeled by a title, and can be recorded on multiple mediums. Conversely, a medium identifies a singular recording of an album, enforced by a foreign key constraint.

The integrity of the count attribute in **Genre**, are not guaranteed by the table creation. It will later be enforced later on by the import and delete data commands.

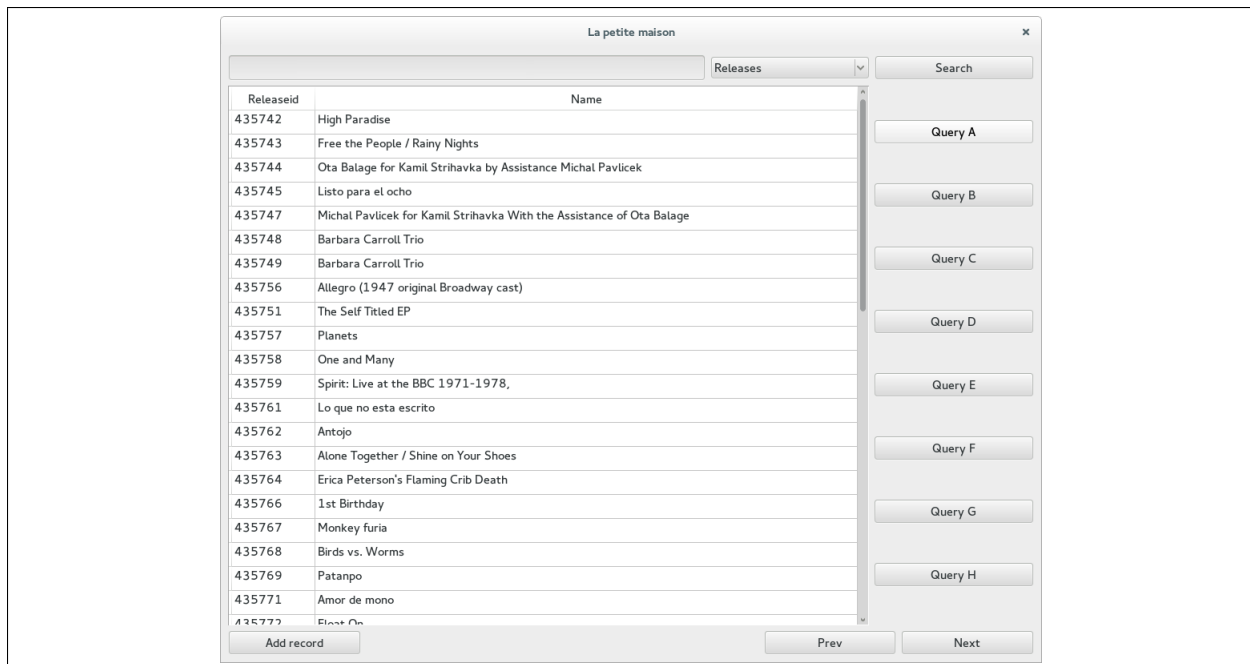
Design changes from deliverable 1

We removed all of our 'at least one' constraints to facilitate the import of new data. For the same reason, we also changed our model to be closer to the given data, so that one can easily add any information into the database and not just track-related information like we initially had in mind. That is, in our first model, everything had to be related to a track to be relevant but now every entities are independent.

About data import

We have not yet be able to import all data into the database. We tried using SQL loader to gain time but we ended up messing up our indexes and we pretty much had to empty all of our tables. In the end we were unable to extend the index of the two last tables, not allowing us to load the full data. This part of the assignment will be completed for the final deliverable. For this one, we worked with a few hundreds entries for each table.

Interface



We chose to use Python to design the software to harness the powerful SQLAlchemy API to interface with the database. Therefore we used PyQt5 to design the GUI itself providing clean and practical tools. SQLAlchemy offers a very useful model as it reflects accurately the actual database schema and makes it easier to map the data to the Model/View offered by Qt. Most of the window is taken over by the table representation presenting the data. It offers contextual research such as getting all records for an artist or for a release.



At the top of the window is a search field. The entry is searched in the field selected in the ComboBox. On the right is a list of buttons to perform the queries requested in the deliverable. The last function is the import of data in the database which is handled quite straightforwardly by a dialog which presents the fields to fill in and generates dynamically the INSERT statement and the needed dynamic parameters, namely the id and the updated count per genre value.

SQL Queries

```
-- 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.
5 SELECT      arti.name
FROM          artists arti, areas area
WHERE         arti.areaID=area.areaID AND area.name='Switzerland' ;
```

SQL script for query A

```

-- 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.

5
-- Area with the most male Artists
Select  artistarea.areaname, artistarea.Type, count(*) "number" from
        (Select * from Artists arti INNER JOIN (
          SELECT  Area.name areaname, area.areaId
10         FROM    Areas area
          WHERE    area.areaId = (
                                SELECT  AreaId areafemale
                                FROM    (
                                  SELECT  AreaId , count(*) c
15                                 FROM    Artists
                                  WHERE    (gender = 'Male')
                                  GROUP BY AreaId
                                  ORDER BY c DESC
                                )
                                WHERE    ROWNUM <=1
          )
        ) toparea
        ON arti.areaId = toparea.areaId) artistarea
GROUP BY artistarea.Type , artistarea.areaname
25 ;

-- Area with the most female Artists
Select  artistarea.areaname, artistarea.Type, count(*) "number" from
        (Select * from Artists arti INNER JOIN (
          SELECT  Area.name areaname, area.areaId
30         FROM    Areas area
          WHERE    area.areaId = (
                                SELECT  AreaId areafemale
                                FROM    (
                                  SELECT  AreaId , count(*) c
35                                 FROM    Artists
                                  WHERE    (gender = 'Female')
                                  GROUP BY AreaId
                                  ORDER BY c DESC
                                )
                                WHERE    ROWNUM <=1
          )
        ) toparea
        ON arti.areaId = toparea.areaId) artistarea
GROUP BY artistarea.Type , artistarea.areaname
45 ;

-- Area with the most female Groups
Select  artistarea.areaname, artistarea.Type, count(*) "number" from
        (Select * from Artists arti INNER JOIN (
          SELECT  Area.name areaname, area.areaId
50         FROM    Areas area
          WHERE    area.areaId = (
                                SELECT  AreaId areafemale
                                FROM    (

```

```

55         SELECT      AreaId , count(*) c
            FROM      Artists
            WHERE      (gender = 'Female') and (type = 'Group')
            GROUP BY   AreaId
            ORDER BY   c DESC
        )
60     WHERE          ROWNUM <=1
    )
    ) toparea
    ON arti.areaid = toparea.areaid) artistarea
GROUP BY artistarea.Type , artistarea.areaname
65 ;

```

SQL script for query B

```

-- List the names of 10 groups with the most recorded tracks.
SELECT      *
FROM (
5     SELECT      Name
        FROM      Artists arti
        INNER JOIN (
            SELECT      ArtistId
            FROM (
10             SELECT ArtistId , count(*) numb
                FROM      TRACK_ARTIST
                GROUP BY   ArtistId
                ORDER BY   numb DESC )
            ) artiId
        ON          arti.ArtistId = artiId.ArtistId
15     WHERE      arti.Type = "GROUP"
    )
WHERE          ROWNUM <=10 ;

```

SQL script for query C

```

-- List the names of 10 groups with the most releases.
SELECT      *
FROM
5 (
    SELECT      arti.name
    FROM      Artists arti
    INNER JOIN (
10         SELECT      ArtistId, COUNT(DISTINCT ReleaseID) num
        FROM          Track_Artist trackarti
        INNER JOIN (
            SELECT      *
            FROM          Tracks track
            INNER JOIN (
15             SELECT      mediums.MediumId, Mediums.AlbumID
                FROM          Mediums mediums
                INNER JOIN      Albums albums
                ON              mediums.AlbumID = albums.AlbumID
            ) media
        )
    )

```

```

                ON track.MediumID = media.Medium
20         ) track
        ON trackarti.TrackID = track.TrackID
        GROUP BY ArtistID
        ORDER BY num DESC
    ) artiId
25 ON arti.ArtistId = artiId.ArtistId
WHERE arti.Type = "Group"
)
WHERE ROWNUM <=10 ;

```

SQL script for query D

```

-- Print the name of a female artist associated with the most genres.
SELECT name
FROM Artists arti
INNER JOIN (
5     SELECT arti.ArtistID, COUNT(DISTINCT genre.GenreID) numb
        FROM Artists arti
        INNER JOIN Artist_Genre genre
        ON arti.ArtistID = genre.ArtistID
        WHERE arti.Gender = 'Female'
10     GROUP BY arti.ArtistID
        ORDER BY numb DESC
    ) artigenre
ON arti.artistid = artigenre.artistid
WHERE ROWNUM <=1 ;

```

SQL script for query E

```

-- List all cities which have more female than male artists.
SELECT areamalefemale.topname
FROM (
5     SELECT city."NAME" topname , city.AreaId,
        count (CASE WHEN arti.gender = 'Female' THEN 1 END) AS females,
        count (CASE WHEN arti.gender = 'Male' THEN 1 END) AS males
        FROM Artists arti
        INNER JOIN Areas city
        ON city.areasid = arti.areasid
10     WHERE city.type = 'City'
        GROUP BY city.areasid , city."NAME"
    ) areamalefemale
WHERE areamalefemale.Females > areamalefemale.Males

```

SQL script for query F

```

-- List the mediums with the highest number of tracks.
SELECT track.mediumid
FROM Tracks track
GROUP BY track.mediumid
5 HAVING COUNT (*) >= ALL (
        SELECT COUNT(*)

```



```
FROM Tracks track
GROUP BY track.mediumid
)
```

SQL script for query G

```
-- 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 art.artistId, MAX ( trackCount )
5 FROM (
    SELECT artistId, COUNT(*) trackCount
```

SQL script for query H

SQL script for query I

SQL script for query J

SQL script for query K

SQL script for query L

SQL script for query M

SQL script for query N

SQL script for query O

SQL script for query P

SQL script for query Q

SQL script for query R

SQL script for query S