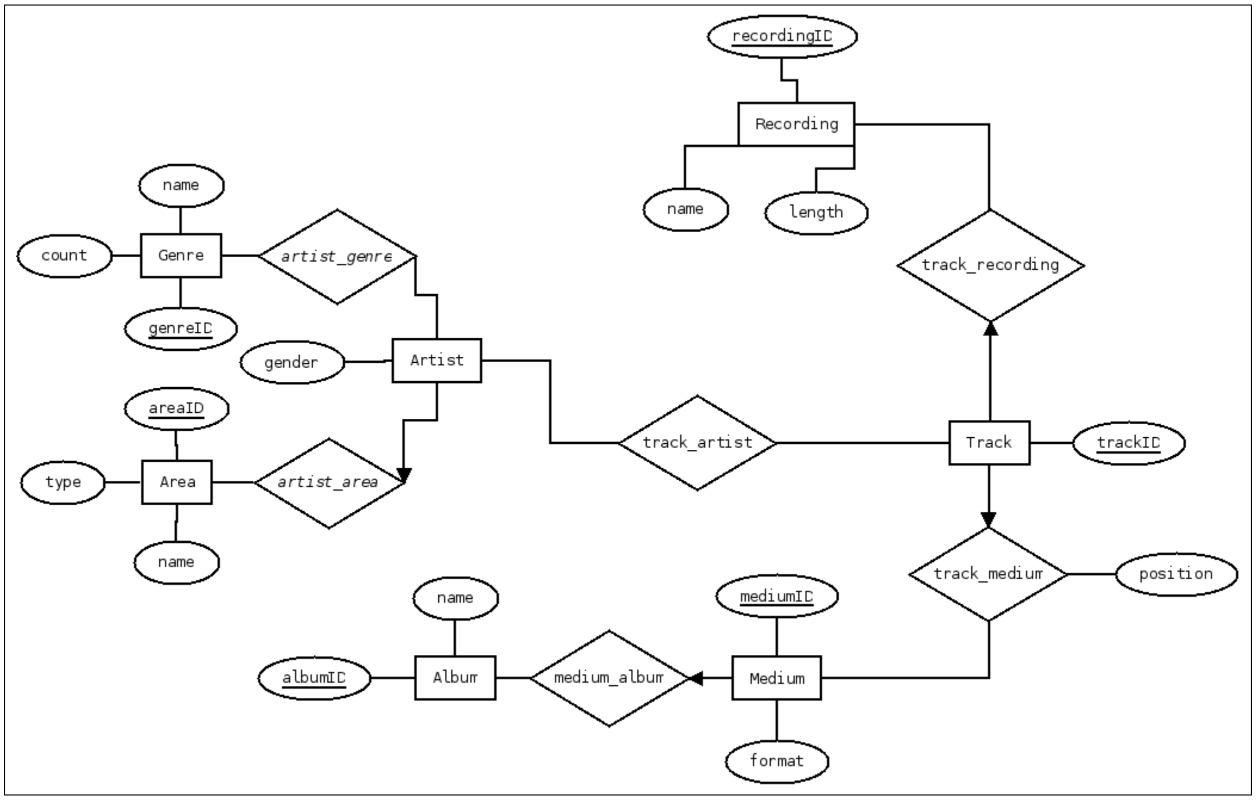
CS-322 Introduction to Database Systems
Project Deliverable #2

Due on Sunday, May 4th, 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(255) NOT NULL,
  type VARCHAR(255),
 PRIMARY KEY (areaID) ) ;

CREATE TABLE Genres (
  genreID INTEGER,
  name VARCHAR(255) NOT NULL,
  count INTEGER DEFAULT 0,
 PRIMARY KEY (genreID) ) ;

CREATE TABLE Artists (
  artistID INTEGER,
  name VARCHAR(255) NOT NULL,
  areaID INTEGER,
  gender CHAR(1),
 PRIMARY KEY (artistID),
 FOREIGN KEY (areaID) REFERENCES Areas ) ;


CREATE TABLE Recordings (
  recordingID INTEGER,
  name VARCHAR(255) ,
  length INTEGER,
  PRIMARY KEY (recordingID) ) ;



CREATE TABLE Albums (
  albumID INTEGER,
  name VARCHAR(255) NOT NULL,
  PRIMARY KEY (albumID) ) ;

CREATE TABLE Mediums (
  mediumID INTEGER,
  albumID INTEGER,
  format VARCHAR(255),
  PRIMARY KEY (mediumID),
  FOREIGN KEY (albumID) REFERENCES Albums ) ;


CREATE TABLE Tracks (
  trackID INTEGER,
  mediumID INTEGER,
  recordingID INTEGER,
  position INTEGER,
  PRIMARY KEY (trackID),
  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),
  FOREIGN KEY (artistID) REFERENCES Artists,
  FOREIGN KEY (genreID) REFERENCES Genres ) ;


CREATE TABLE Track_artist (
  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 and their physical incarnation. We decided to enforce **Song** as the only necessary information to describe music, emulating the approach taken by most popular music player softwares since data can be incomplete. We put a NOT NULL constraint on most of the name attributes of the entities, with the exception of **Song** 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 **Song** is related to:

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

  Medium: Though a song is not necessarily part of an album, it has to be recorded on some medium. There is therefore a participation constraint of **Song** in **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, but makes no sense as an empty container, thus triggering a participation constraint, 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, and a place which doesn't house such a creative conundrum isn't worth keeping track of in a music database.

- An **Album** is the logical aggregation of songs, labeled by a title, and can be recorded on multiple — at least one, participation constraint— media. Conversely, a medium identifies a singular recording of an album, enforced by a foreign key constraint.

Some constraints, the integrity of the count attribute in **Genre** and the several "at least one" constraints, are not guaranteed by the table creation. They will later be enforced later on by the import and delete data commands.

SQL Queries

```sql
SELECT a.name
FROM artists a, areas l
WHERE a.areaID=l.areaID AND l.name='Switzerland' ;
```

SQL script for query A

```sql
SELECT area.name
FROM Areas area
WHERE area.areaid = (    SELECT AreaId areafemale
                              FROM (    SELECT AreaId , count(*) c
                                        FROM Artists
                                        WHERE (gender = 'Male')
                                        GROUP BY AreaId
                                        ORDER BY c DESC                          )
                                WHERE ROWNUM <=1                                     ) ;
```

SQL script for query B

```
SELECT Name
FROM Artists arti
INNER JOIN   (  SELECT ArtistId
                    FROM (    SELECT ArtistId  , count(*) numb
                              FROM TRACK_ARTIST
                              GROUP BY ArtistId
                              ORDER BY numb DESC )) artiId
ON arti.ArtistId = artiId.ArtistId
WHERE ROWNUM <=10  ;
```

SQL script for query C

```
SELECT name
FROM Artists arti
INNER JOIN   (  SELECT ArtistId, COUNT(DISTINCT AlbumId) num
                    FROM track_artist trackarti
                    INNER JOIN   (  SELECT *
                                      FROM Tracks track
                                      INNER JOIN   (  SELECT mediums.mediumid, mediums.albumid
                                                        FROM Mediums
                                                        INNER JOIN Albums
                                                        ON mediums.albumid = albums.albumid
                                      ON track.mediumid = medi.mediumid
                    ON trackarti.trackid = track.trackid
                    GROUP BY ArtistId
                    ORDER BY num DESC
ON arti.ArtistId = artiId.ArtistId
WHERE ROWNUM <=10  ;
```

SQL script for query D

```
SELECT name
FROM Artists arti
INNER JOIN   (  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'
                    GROUP BY arti.ArtistId
                    ORDER BY numb DESC                                               ) artigenre
ON arti.artistid = artigenre.artistid
WHERE ROWNUM <=1  ;
```

SQL script for query E

```
SELECT AreaId
FROM  (   SELECT city.areaid, count(CASE WHEN arti.gender = 'Female' THEN 1 END) AS females, count(CA
          FROM Artists arti
          INNER JOIN Areas city
          ON city.areaid = arti.areaId
          WHERE city.type = 'City'
          GROUP BY city.areaId ) areamalefemale
WHERE areamalefemale.Females > areamalefemale.Males
```

SQL script for query F

```
SELECT medi.albumid, count(DISTINCT track.trackid)
FROM Tracks track
INNER JOIN  (  SELECT mediums.mediumid ,mediums.albumid
                   FROM Mediums
                   INNER JOIN Albums ON mediums.albumid = albums.albumid   ) medi
ON track.mediumid = medi.mediumid
GROUP BY medi.albumid ;
```

SQL script for query G

Interface