SAE 2.04 – Livrable 1 de Bases de données

Modèle entité/association

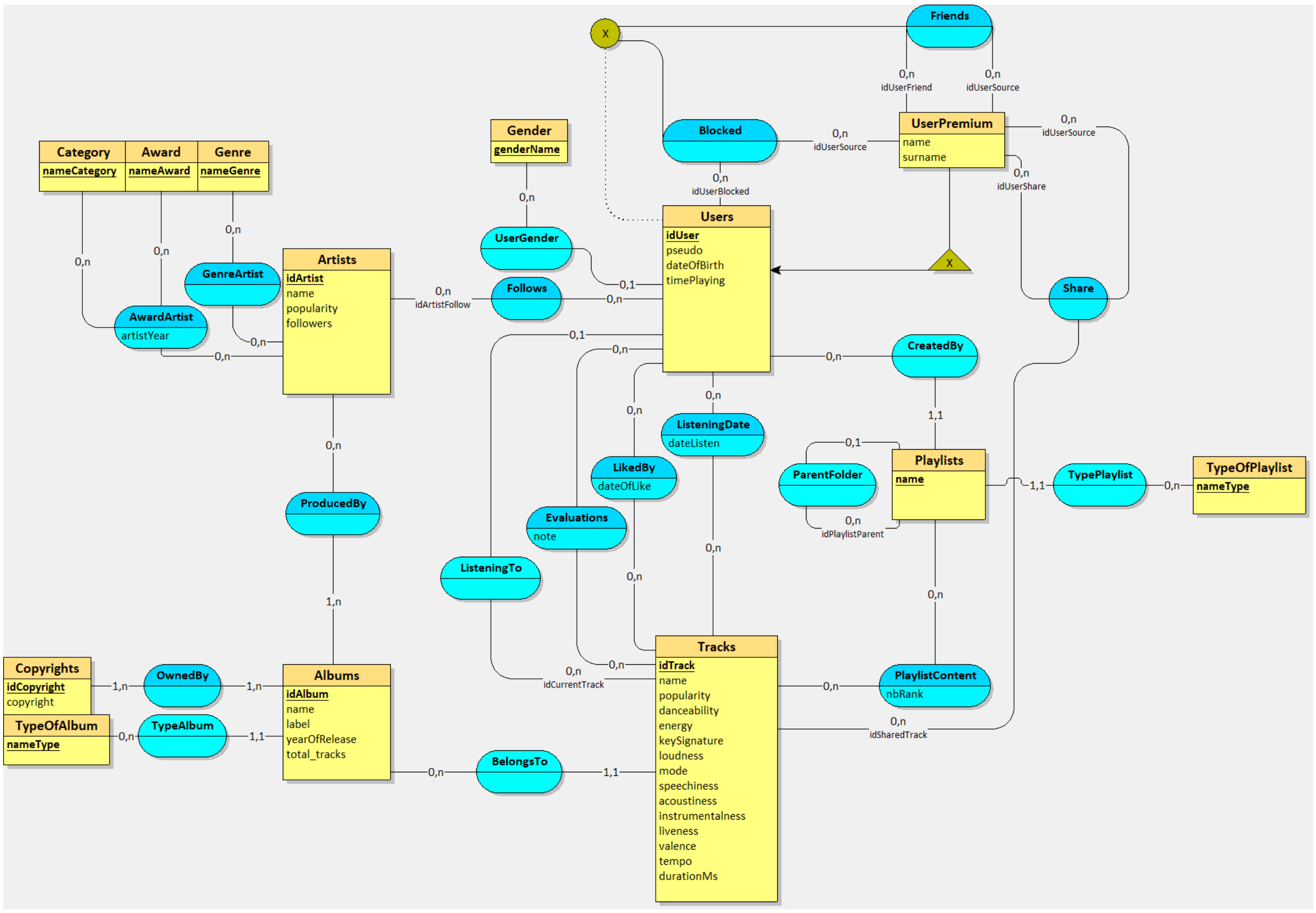

Schéma Relationnel de la base de données

Copyrights (idCopyright, copyright)

Artists (idArtist, name, popularity, followers)

Category (nameCategory);

Genre (nameGenre);

Award (nameAward)

TypeOfAlbum (nameType)

TypeOfPlaylist (nameType)

Gender (genderName)

Albums (idAlbum, name, label, yearOfRelease, total_tracks, #nameType)

Tracks (idTrack, name, popularity, danceability, energy, keySignature, loudness, mode, speechiness, acoustiness, instrumentalness, liveness, valence, tempo, durationMs, #idAlbum)

Users (idUser, pseudo, dateOfBirth, timePlaying, #genderName, #idCurrentTrack)

Playlists (name, #nameType, #idUser, #idPlaylistParent)

UserPremium (#idUser, name, surname)

Follows (#idArtistFollow, #idUser)

ProducedBy (#idAlbum, #idArtist)

OwnedBy (#idAlbum, #idCopyright)

Evaluations (#idUser, #idTrack, note)

LikedBy (#idUser, #idTrack, dateOfLike)

PlaylistContent (#idTrack, #name, nbRank)

Share (#idSharedTrack, #(#idUser_idUserShare), #(#idUser_idUserSource))

Blocked (#idUserBlocked, #(#idUser_idUserSource))

GenreArtist (#idArtist, #nameGenre)

AwardArtist (#idArtist, #nameCategory, #nameAward, artistYear)

Friends (#(#idUser_idUserSource), #(#idUser_idUserFriend))

ListeningDate (#idUser, #idTrack, dateListen)

Requêtes SQL de création/insertion des données dans les tables

Création des tables

```
SET AUTOCOMMIT ON;

CREATE TABLE Copyrights (
    idCopyright INT,
    copyright   VARCHAR(50),
    CONSTRAINT pk_Copyrights PRIMARY KEY (idCopyright)
);

CREATE TABLE Artists (
    idArtist   CHAR(22),
    name       VARCHAR(50),
    popularity INT NOT NULL,
    followers  INT NOT NULL,
    CONSTRAINT pk_Artists PRIMARY KEY (idArtist)
);

CREATE TABLE Category (
    nameCategory VARCHAR(50),
    CONSTRAINT pk_Category PRIMARY KEY (nameCategory)
);

CREATE TABLE Genre (
    nameGenre VARCHAR(50),
    CONSTRAINT pk_Genre PRIMARY KEY (nameGenre)
);

CREATE TABLE Award (
    nameAward VARCHAR(50),
    CONSTRAINT pk_Award PRIMARY KEY (nameAward)
);

CREATE TABLE TypeOfAlbum (
    nameType VARCHAR(50),
    CONSTRAINT pk_TypeOfAlbum PRIMARY KEY (nameType)
);

CREATE TABLE TypeOfPlaylist (
    nameType VARCHAR(50),
    CONSTRAINT pk_TypeOfPlaylist PRIMARY KEY (nameType)
);

CREATE TABLE Gender (
    genderName CHAR(1),
    CONSTRAINT pk_Gender PRIMARY KEY (genderName)
);


-- DEBUT DES FOREIGN KEY--


CREATE TABLE Albums (
    idAlbum       CHAR(22),
    name          VARCHAR(50) NOT NULL,
    label         VARCHAR(50) NOT NULL,
    yearOfRelease INT         NOT NULL,
    total_tracks  INT         NOT NULL,
    nameType      VARCHAR(50) NOT NULL,
    CONSTRAINT pk_Albums PRIMARY KEY (idAlbum),
    CONSTRAINT fk_Albums_TypeOfAlbum FOREIGN KEY (nameType) REFERENCES TypeOfAlbum (nameType)
);

CREATE TABLE Tracks (
    idTrack          CHAR(22),
    name             VARCHAR(50) NOT NULL,
    popularity       INT         NOT NULL,
    danceability     REAL,
    energy           REAL,
    keySignature     INT,
    loudness         REAL,
    modeTrack        NUMBER(1),
    speechiness      REAL,
    acoustiness      REAL,
    instrumentalness REAL,
    liveness         REAL,
    valence          REAL,
    tempo            REAL,
    durationMs       INT,
    idAlbum          CHAR(22)    NOT NULL,
    CONSTRAINT pk_Tracks PRIMARY KEY (idTrack),
    CONSTRAINT fk_Tracks_Albums FOREIGN KEY (idAlbum) REFERENCES Albums (idAlbum),
    CONSTRAINT mode_constraints CHECK (modeTrack = 0 OR modeTrack = 1)
);

CREATE TABLE Users (
    idUser        INT,
    pseudo        VARCHAR(50) NOT NULL,
    dateOfBirth   DATE        NOT NULL,
    timePlaying   INT,
    genderName    CHAR(1),
    idCurrentTrack CHAR(22),
    CONSTRAINT pk_Users PRIMARY KEY (idUser),
    CONSTRAINT fk_Users_Gender FOREIGN KEY (genderName) REFERENCES Gender (genderName),
    CONSTRAINT fk_Users_Tracks FOREIGN KEY (idCurrentTrack) REFERENCES Tracks (idTrack)
);

CREATE TABLE Playlists (
    idUser          INT         NOT NULL,
    name            VARCHAR(50) NOT NULL,
    nameType        VARCHAR(50) NOT NULL,
    nameElementParent VARCHAR(50),
    nameTypeParent    VARCHAR(50),
    CONSTRAINT pk_Playlists PRIMARY KEY (idUser, name, nameType),
        CONSTRAINT fk_Playlists_TypeOfPlaylist FOREIGN KEY (nameType) REFERENCES TypeOfPlaylist
(nameType),
        CONSTRAINT fk_Playlists_Playlists FOREIGN KEY (idUser,nameElementParent,nameTypeParent)
REFERENCES Playlists (idUser, name, nameType),
    CONSTRAINT fk_Playlists_Users FOREIGN KEY (idUser) REFERENCES Users (idUser),
    CONSTRAINT ParentChecker CHECK (nameTypeParent = 'Folder' OR nameTypeParent IS NULL)
);

CREATE TABLE UserPremium (
    idUser  INT,
    name    VARCHAR(50),
    surname VARCHAR(50),
    CONSTRAINT pk_UserPremium PRIMARY KEY (idUser),
    CONSTRAINT fk_UserPremium_Users FOREIGN KEY (idUser) REFERENCES Users (idUser)
);

CREATE TABLE Follows (
    idArtistFollow CHAR(22),
    idUser        INT,
    CONSTRAINT pk_Follows PRIMARY KEY (idArtistFollow, idUser),
    CONSTRAINT fk_Follows_Artists FOREIGN KEY (idArtistFollow) REFERENCES Artists (idArtist),
    CONSTRAINT fk_Follows_Users FOREIGN KEY (idUser) REFERENCES Users (idUser)
);

CREATE TABLE ProducedBy (
    idAlbum  CHAR(22),
    idArtist CHAR(22),
    CONSTRAINT pk_ProducedBy PRIMARY KEY (idAlbum, idArtist),
    CONSTRAINT fk_ProducedBy_Albums FOREIGN KEY (idAlbum) REFERENCES Albums (idAlbum),
    CONSTRAINT fk_ProducedBy_Artists FOREIGN KEY (idArtist) REFERENCES Artists (idArtist)
);

CREATE TABLE OwnedBy (
    idAlbum     CHAR(22),
    idCopyright INT,
    CONSTRAINT pk_OwnedBy PRIMARY KEY (idAlbum, idCopyright),
    CONSTRAINT fk_OwnedBy_Albums FOREIGN KEY (idAlbum) REFERENCES Albums (idAlbum),
    CONSTRAINT fk_OwnedBy_Copyrights FOREIGN KEY (idCopyright) REFERENCES Copyrights (idCopyright)
);

CREATE TABLE Evaluations (
    idUser  INT,
    idTrack CHAR(22),
    note    INT NOT NULL,
    CONSTRAINT pk_Evaluations PRIMARY KEY (idUser, idTrack,note),
    CONSTRAINT fk_Evaluations_Users FOREIGN KEY (idUser) REFERENCES Users (idUser),
    CONSTRAINT fk_Evaluations_Tracks FOREIGN KEY (idTrack) REFERENCES Tracks (idTrack),
    CONSTRAINT check_note_between_0_and_20 CHECK (note >= 0 AND note <= 20)
);

CREATE TABLE LikedBy (
    idUser     INT,
    idTrack    CHAR(22),
    dateOfLike DATE NOT NULL,
    CONSTRAINT pk_LikedBy PRIMARY KEY (idUser, idTrack),
    CONSTRAINT fk_LikedBy_Users FOREIGN KEY (idUser) REFERENCES Users (idUser),
    CONSTRAINT fk_LikedBy_Tracks FOREIGN KEY (idTrack) REFERENCES Tracks (idTrack)
);

CREATE TABLE PlaylistContent (
    idTrack        CHAR(22),
    nbRank         INT,
    idUserPlaylist INT        NOT NULL,
    playlistName   Varchar(50) NOT NULL,
    playlistType   Varchar(50) NOT NULL,
    CONSTRAINT pk_PlaylistContent PRIMARY KEY (idTrack, idUserPlaylist, playlistName, playlistType),
    CONSTRAINT fk_PlaylistContent_Tracks FOREIGN KEY (idTrack) REFERENCES Tracks (idTrack),
       CONSTRAINT fk_PlaylistContent_Playlist FOREIGN KEY (idUserPlaylist, playlistName, playlistType)
REFERENCES Playlists (idUser, name, nameType)
);

CREATE TABLE ShareTrack (
    idSharedTrack CHAR(22),
    idUserShare   INT,
    idUserSource  INT,
    CONSTRAINT pk_ShareTrack PRIMARY KEY (idSharedTrack, idUserShare, idUserSource),
    CONSTRAINT fk_ShareTrack_Tracks FOREIGN KEY (idSharedTrack) REFERENCES Tracks (idTrack),
    CONSTRAINT fk_ShareTrack_UserShare FOREIGN KEY (idUserShare) REFERENCES UserPremium (idUser),
    CONSTRAINT fk_ShareTrack_UserSource FOREIGN KEY (idUserSource) REFERENCES UserPremium (idUser)
);

CREATE TABLE Blocked (
    idUserBlocked INT,
    idUserSource  INT,
    CONSTRAINT pk_Blocked PRIMARY KEY (idUserBlocked, idUserSource),
    CONSTRAINT fk_Blocked_UserBlocked FOREIGN KEY (idUserBlocked) REFERENCES Users (idUser),
    CONSTRAINT fk_Blocked_UserPremiumSource FOREIGN KEY (idUserSource) REFERENCES UserPremium
(idUser)
);

CREATE TABLE GenreArtist (
    idArtist  CHAR(22),
    nameGenre VARCHAR(50),
    CONSTRAINT pk_GenreArtist PRIMARY KEY (idArtist, nameGenre),
    CONSTRAINT fk_GenreArtists_Artists FOREIGN KEY (idArtist) REFERENCES Artists (idArtist),
    CONSTRAINT fk_GenreArtists_Genre FOREIGN KEY (nameGenre) REFERENCES Genre (nameGenre)
);


CREATE TABLE AwardArtist (
    idArtist     CHAR(22),
    nameCategory VARCHAR(50),
    nameAward    VARCHAR(50),
    artistYear   INT,
    CONSTRAINT pk_AwardArtist PRIMARY KEY (idArtist, nameCategory, nameAward, artistYear),
    CONSTRAINT fk_AwardArtist_Artists FOREIGN KEY (idArtist) REFERENCES Artists (idArtist),
    CONSTRAINT fk_AwardArtist_Category FOREIGN KEY (nameCategory) REFERENCES Category
(nameCategory),
    CONSTRAINT fk_AwardArtist_Award FOREIGN KEY (nameAward) REFERENCES Award (nameAward)
);


CREATE TABLE Friends (
    idUserSource INT,
    idUserFriend INT,
    CONSTRAINT pk_Friends PRIMARY KEY (idUserSource, idUserFriend),
    CONSTRAINT fk_Friends_UserPremiumSource FOREIGN KEY (idUserSource) REFERENCES UserPremium
(idUser),
    CONSTRAINT fk_Friends_UserFriend FOREIGN KEY (idUserFriend) REFERENCES Users (idUser)
);


CREATE TABLE ListeningDate (
    idUser     INT,
    idTrack    CHAR(22),
    dateListen TIMESTAMP NOT NULL,
    CONSTRAINT pk_ListeningDate PRIMARY KEY (idUser, idTrack, dateListen),
    CONSTRAINT fk_ListeningDate_Users FOREIGN KEY (idUser) REFERENCES Users (idUser),
    CONSTRAINT fk_ListeningDate_Tracks FOREIGN KEY (idTrack) REFERENCES Tracks (idTrack)
```

```
);


-- Début des triggers --


CREATE OR REPLACE TRIGGER trg_single_has_only_1_song
    BEFORE INSERT OR UPDATE
    ON Tracks
    FOR EACH ROW
DECLARE
    albumType   VARCHAR(50);
    track_count INT;
BEGIN
    SELECT nameType
    INTO albumType
    FROM ALBUMS
    WHERE idAlbum = :NEW.idAlbum;

    IF (albumType = 'Single') THEN
        SELECT COUNT(*)
        INTO track_count
        FROM TRACKS
        WHERE idAlbum = :NEW.idAlbum;

        IF (track_count >= 1) THEN
            RAISE_APPLICATION_ERROR(-20001, 'A Single album cannot have more than one track');
        END IF;
    END IF;
END;

--

CREATE
    OR REPLACE TRIGGER trg_Block
    BEFORE INSERT
    ON Blocked
    FOR EACH ROW
DECLARE
    friend_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO friend_count
    FROM Friends
    WHERE :NEW.idUserSource = Friends.idUserSource
      AND Friends.idUserFriend = :NEW.idUserBlocked;
```

```
  IF
    friend_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'L''utilisateur est dans votre liste amis.');
  END IF;
END;

--

CREATE
  OR REPLACE TRIGGER trg_AddFriend
  BEFORE INSERT
  ON Friends
  FOR EACH ROW
DECLARE
  block_count INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO block_count
  FROM Blocked
  WHERE :NEW.idUserSource = Blocked.idUserSource
   AND Blocked.idUserBlocked = :NEW.idUserFriend;

  IF
    block_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'L''utilisateur est bloqué.');
  END IF;
END;

--

CREATE OR REPLACE TRIGGER increment_total_tracks
AFTER INSERT ON Tracks
FOR EACH ROW
BEGIN
  UPDATE Albums
  SET total_tracks = total_tracks + 1
  WHERE idAlbum = :NEW.idAlbum;
END;

--

CREATE OR REPLACE TRIGGER increment_followers
  AFTER INSERT ON Follows
  FOR EACH ROW
BEGIN
  UPDATE Artists
  SET followers = Artists.followers + 1
```

```
    WHERE idArtist = :NEW.idArtistFollow;
END;

--

CREATE OR REPLACE TRIGGER trg_BlockFriendExclusion
    BEFORE INSERT OR UPDATE ON Blocked
    FOR EACH ROW
DECLARE
    friend_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO friend_exists
    FROM Friends
    WHERE (Friends.idUserSource = :NEW.idUserSource AND Friends.idUserFriend = :NEW.idUserBlocked);

    IF friend_exists > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'An user cannot be both blocked and a friend.');
    END IF;
END;

--

CREATE OR REPLACE TRIGGER trg_FriendBlockExclusion
    BEFORE INSERT OR UPDATE ON Friends
    FOR EACH ROW
DECLARE
    blocked_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO blocked_exists
    FROM Blocked
    WHERE (Blocked.idUserSource = :NEW.idUserSource AND Blocked.idUserBlocked = :NEW.idUserFriend);

    IF blocked_exists > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'An user cannot be both a friend and blocked.');
    END IF;
END;

COMMIT;
```

Insertions des données dans les tables

```
SET AUTOCOMMIT ON;

INSERT INTO COPYRIGHTS (IDCOPYRIGHT,COPYRIGHT)
SELECT DISTINCT idCopyright, Copyright
FROM TEMP_ALBUMS
ORDER BY IDCOPYRIGHT;
```

```
--SELECT * FROM Copyrights;

INSERT INTO ARTISTS
SELECT DISTINCT IDARTIST, NAME, POPULARITY,0
FROM TEMP_ARTISTS
ORDER BY NAME;

--SELECT * FROM Artists

INSERT INTO CATEGORY
SELECT DISTINCT NAMECATEGORY
FROM TEMP_ARTISTS
WHERE NAMECATEGORY IS NOT NULL
ORDER BY NAMECATEGORY;

--SELECT * FROM Category

INSERT INTO GENRE
SELECT DISTINCT nameGenre
FROM TEMP_ARTISTS
WHERE nameGenre IS NOT NULL
ORDER BY NAMEGENRE;

--SELECT * FROM Genre

INSERT INTO Award
   SELECT DISTINCT NAMEAWARD
   FROM TEMP_ARTISTS
   WHERE NAMEAWARD IS NOT NULL

--SELECT * FROM Award

INSERT INTO TypeOfAlbum
   SELECT DISTINCT NAMETYPE
   FROM TEMP_ALBUMS
   WHERE NAMETYPE IS NOT NULL

--SELECT * FROM TypeOfAlbum

INSERT INTO TypeOfPlaylist
   SELECT DISTINCT TYPE
   FROM TEMP_PLAYLIST
   WHERE TYPE IS NOT NULL

--SELECT * FROM TypeOfPlaylist
```

```
INSERT INTO Gender
   SELECT DISTINCT GENDERNAME
   FROM TEMP_USERS
   WHERE GENDERNAME IS NOT NULL

--SELECT * FROM Gender

INSERT INTO Albums (IDALBUM, NAME, LABEL, YEAROFRELEASE, TOTAL_TRACKS, NAMETYPE)
   SELECT DISTINCT IDALBUM, NAME, LABEL, YEAROFRELEASE, 0, NAMETYPE
   FROM TEMP_ALBUMS
   ORDER BY NAME;

--SELECT * FROM Albums

INSERT INTO Tracks
   SELECT DISTINCT idtrack, name, popularity, danceability, energy, keysignature, loudness,
                  "MODE", speechiness, acoustiness, instrumentalness, liveness, valence, tempo, durationms,
idalbum
   FROM TEMP_TRACKS
   ORDER BY name;

--SELECT * FROM Tracks

INSERT INTO Users
   SELECT DISTINCT iduser, pseudo, dateofbirth, timeplaying, gendername, idcurrenttrack
   FROM TEMP_USERS
   ORDER BY idUser;

--SELECT * FROM Users

INSERT INTO Playlists (iduser, name, nametype, nameelementparent)
   SELECT IDUSER, NAME, TYPE, NAMEPARENTELEMENT
   FROM TEMP_PLAYLIST
   GROUP BY IDUSER, NAME, TYPE, NAMEPARENTELEMENT
   ORDER BY IDUSER;

UPDATE PLAYLISTS
SET NAMETYPEPARENT = 'Folder'
WHERE NAMEELEMENTPARENT IS NOT NULL;

--SELECT * FROM Playlists

INSERT INTO UserPremium
   SELECT DISTINCT IDUSER, name, surname
   FROM TEMP_USERS
   WHERE name IS NOT NULL
   ORDER BY IDUSER;
```

```
--SELECT * FROM UserPremium

INSERT INTO Follows
   SELECT IDARTISTFOLLOW, IDUSER
   FROM TEMP_USERS
   WHERE IDARTISTFOLLOW IS NOT NULL
   GROUP BY IDARTISTFOLLOW, IDUSER
   ORDER BY idUser;

--SELECT * FROM Follows

INSERT INTO ProducedBy
   SELECT IDALBUM, IDARTIST
   FROM TEMP_ALBUMS
   GROUP BY IDALBUM, IDARTIST

--SELECT * FROM ProducedBy

INSERT INTO OwnedBy
   SELECT IDALBUM, IDCOPYRIGHT
   FROM TEMP_ALBUMS
   GROUP BY IDALBUM, IDCOPYRIGHT
   ORDER BY IDCOPYRIGHT;

--SELECT * FROM OwnedBy

INSERT INTO Evaluations
   SELECT idUser, idTrack, note
   FROM TEMP_EVALUATION
   GROUP BY idUser, idTrack, note
   ORDER BY IDUSER;



--SELECT * FROM Evaluations

INSERT INTO LikedBy
   SELECT idUser, IDTRACK, DATELIKE
   FROM TEMP_LIKES
   ORDER BY IDUSER;

--SELECT * FROM LikedBy


INSERT INTO PlaylistContent
   SELECT IDTRACK, "ORDER", IDUSER, NAME, TYPE
```

```sql
    FROM TEMP_PLAYLIST
    WHERE idTrack IS NOT NULL
    ORDER BY IDUSER;

--SELECT * FROM PlaylistContent


INSERT INTO ShareTrack
SELECT IDTRACK, IDUSERSHARE, IDUSER
    FROM TEMP_SHARE
    ORDER BY IDUSER;

--SELECT * FROM ShareTrack


INSERT INTO Blocked
    SELECT IDUSERBLOCKED, IDUSERPREMIUM
    FROM TEMP_USERS_BLOCKED
    ORDER BY IDUSERPREMIUM;

--SELECT * FROM Blocked

INSERT INTO GenreArtist
    SELECT IDARTIST, NAMEGENRE
    FROM TEMP_ARTISTS
    WHERE nameGenre IS NOT NULL
    GROUP BY IDARTIST, NAMEGENRE
    ORDER BY NAMEGENRE;

--SELECT * FROM GenreArtist

INSERT INTO AwardArtist
    SELECT IDARTIST, NAMECATEGORY, NAMEAWARD, ARTISTYEAR
    FROM TEMP_ARTISTS
    WHERE NAMECATEGORY IS NOT NULL
    GROUP BY IDARTIST, NAMECATEGORY, NAMEAWARD, ARTISTYEAR
    ORDER BY NAMECATEGORY;

--SELECT * FROM AwardArtist

INSERT INTO Friends
    SELECT IDUSERPREMIUM, IDUSERFRIEND
    FROM TEMP_FRIENDS
    ORDER BY IDUSERPREMIUM;

--SELECT * FROM Friends
```

```
INSERT INTO ListeningDate
    SELECT IDUSER, IDTRACK, DATELISTEN
    FROM TEMP_TRACKS
    WHERE idUser IS NOT NULL
    GROUP BY IDUSER, IDTRACK, DATELISTEN
    ORDER BY IDUSER;

--SELECT * FROM ListeningDate
```

Suppression des tables

```
DROP TABLE ListeningDate;
DROP TABLE Friends;
DROP TABLE AwardArtist;
DROP TABLE GenreArtist;
DROP TABLE Blocked;
DROP TABLE ShareTrack;
DROP TABLE PlaylistContent;
DROP TABLE LikedBy;
DROP TABLE Evaluations;
DROP TABLE OwnedBy;
DROP TABLE ProducedBy;
DROP TABLE Follows;
DROP TABLE UserPremium;
DROP TABLE Playlists;
DROP TABLE Users;
DROP TABLE Tracks;
DROP TABLE Albums;
DROP TABLE Gender;
DROP TABLE TypeOfPlaylist;
DROP TABLE TypeOfAlbum;
DROP TABLE Award;
DROP TABLE Genre;
DROP TABLE Category;
DROP TABLE Artists;
DROP TABLE Copyrights;
```

Nom du schéma sur Oracle

Pour la personne du groupe qui possède les tables ainsi que les données des fichiers CSV, ce n’est autre que : **DEBOISSESONT**