

SAE 2.04 : Exploitation d'une Base de Données

Livrable 1 de Bases de Données

Conception et Création de la Base de Données à partir d'un jeu de données

Par Milwenn **FRANCEUS-COINTREL**, Julien **RENAUD**, Héroïse **RIGAUX**,
Alexandre **DESCHANEL**



**UNIVERSITÉ DE
MONTPELLIER**

Année Universitaire 2023-2024 | Groupe S1-4

Table des matières

Table des matières.....	2
INTRODUCTION.....	3
MODÈLE ENTITÉ/ASSOCIATION.....	4
SCHÉMA RELATIONNEL.....	5
REQUÊTES SQL DE CRÉATION DES TABLES ET D'INSERTION DE DONNÉES.....	6
Script de création des tables.....	6
Script d'insertion des données.....	20
Script d'altération et suppression.....	24

INTRODUCTION

Ce livrable présente une vue d'ensemble détaillée de notre base de données pour une application d'écoute de musique en ligne, faite à partir de fichiers CSV fournis en amont.

L'objectif est de structurer et d'organiser les données de façon efficace, de réduire au maximum la charge sur la base de données et son temps de réponse, chose faite grâce à la minimisation de la redondance d'information et à l'optimisation des requêtes SQL.

Dans ce document vous retrouverez notamment le modèle entité/association, le schéma relationnel et les scripts de création et altération de la base de données.

NB : Toutes les tables peuvent être retrouvées sur le compte étudiant **franceusm**.

Les tables se terminant par **_TEMP** sont des tables temporaires ayant servi au jeu de données.

Veuillez ne pas les prendre en compte.

SCHÉMA RELATIONNEL

Artistes = (idArtiste, nomArtiste, populariteArtiste, nbFollowers) ;

Genres = (idGenre, nomGenre) ;

[etreClasser] GenresArtistes = (idArtiste#, idGenre#) ;

[etre] CategoriesPrix = (nomCategorie) ;

Prix = (idPrix, nomPrix, *nomCategorie#*) ;

[gagner] PrixArtistes = (idPrix#, anneePrix, idArtiste#) ;

Albums = (idAlbum, nomAlbum, labelAlbum, typeAlbum, anneeAlbum, totalTracks) ;

Copyrights = (idCopyright, descriptionCopyright) ;

[protéger] AlbumsCopyrights = (idAlbum#, idCopyright#) ;

Tracks = (idTrack, nomTrack, idAlbum#, populariteTrack, danceAbilite, energie, cleSignature, intensiteSonore, tonalite, elocution, acoustique, instrumentalite, vivacite, ratioAppreciation, tempo, durationMs) ;

[interpreter] TracksInterpretes = (idTrack#, idArtiste#) ;

[participer] AlbumsParticipants = (idAlbum#, idArtiste#) ;

Utilisateurs = (idUtilisateur, pseudo, dateNaissance, sexe, tempsMsTrackLast, *idTrackLast#*) ;

Classiques = (idUtilisateur#) ;

Premiums = (idUtilisateur#, nomPremium, surnomPremium) ;

[suivre] UtilisateursSuivre = (idUtilisateur#, idArtiste#) ;

[etreAmis] PremiumsAmis = ((idUtilisateur#)#, idUtilisateurAmi#) ;

[etreBloque] PremiumsBloques = ((idUtilisateur#)#, idUtilisateurBloque#) ;

[partager] PremiumsPartage = ((idUtilisateur#)#, (idUtilisateurPartage#)#, idTrack#) ;

[ecouter] UtilisateursHistorique = (idUtilisateur#, idTrack#, dateEcoule) ;

[aimer] UtilisateursLike = (idUtilisateur#, idTrack#, dateLike) ;

UtilisateursEvaluations = (idEvaluation, idUtilisateur#, idTrack#, note) ;

ConteneursPlaylists = (idConteneur, nomConteneur, typeConteneur, idDossierParent#, idUtilisateur#) ;

Playlists = (idPlaylist#) ;

DossiersPlaylists = (idDossier#) ;

[ajouter] PlaylistsTracks = ((idPlaylist#)#, ordre, idTrack#) ;

REQUÊTES SQL DE CRÉATION DES TABLES ET D'INSERTION DE DONNÉES

Script de création des tables

```
/*
Relations Artistes
*/

CREATE TABLE Artistes (
    idArtiste CHAR(22),
    nomArtiste VARCHAR(50) NOT NULL,
    populariteArtiste INTEGER DEFAULT 0,
    nbFollowers INTEGER DEFAULT 0,
    CONSTRAINT pk_artistes PRIMARY KEY (idArtiste),
    CONSTRAINT check_artistes_popularite CHECK (populariteArtiste >= 0
AND populariteArtiste <= 100),
    CONSTRAINT check_artistes_nbFollowers CHECK (nbFollowers >= 0)
);

CREATE TABLE Genres (
    idGenre INTEGER,
    nomGenre VARCHAR(50) NOT NULL,
    CONSTRAINT pk_genres PRIMARY KEY (idGenre)
);

-- Association etreClasser
CREATE TABLE GenresArtistes (
    idArtiste CHAR(22),
    idGenre INTEGER,
    CONSTRAINT pk_genresArtistes PRIMARY KEY (idArtiste, idGenre),
    CONSTRAINT fk_genresArtistes_idA_vers_ast FOREIGN KEY (idArtiste)
REFERENCES Artistes(idArtiste),
    CONSTRAINT fk_genresArtistes_idG_vers_gre FOREIGN KEY (idGenre)
REFERENCES Genres(idGenre)
);

CREATE TABLE CategoriesPrix (
    nomCategorie VARCHAR(50),
    CONSTRAINT pk_categoriesPrix PRIMARY KEY (nomCategorie)
);
```

```

CREATE TABLE Prix (
    idPrix INTEGER,
    nomPrix VARCHAR(50) NOT NULL,
    nomCategorie VARCHAR(50) NOT NULL,
    CONSTRAINT pk_prix PRIMARY KEY (idPrix),
    CONSTRAINT fk_prix_nomCateg_vers_ctgPrix FOREIGN KEY (nomCategorie)
REFERENCES CategoriesPrix(nomCategorie)
);

-- Association gagner
CREATE TABLE PrixArtistes (
    idPrix INTEGER,
    anneePrix INTEGER,
    idArtiste CHAR(22) NOT NULL,
    CONSTRAINT pk_artistesPrix PRIMARY KEY (idPrix, anneePrix),
    CONSTRAINT fk_artistesPrix_idAst_vers_ast FOREIGN KEY (idArtiste)
REFERENCES Artistes(idArtiste),
    CONSTRAINT fk_artistesPrix_idP_vers_prix FOREIGN KEY (idPrix)
REFERENCES Prix(idPrix),
    CONSTRAINT check_artistesPrix_anneePrix CHECK (anneePrix > 0)
);

/*
Relations Albums
*/

CREATE TABLE Albums (
    idAlbum CHAR(22),
    nomAlbum VARCHAR(150) NOT NULL,
    labelAlbum VARCHAR(150) NOT NULL,
    typeAlbum VARCHAR(11) NOT NULL,
    anneeAlbum INTEGER NOT NULL,
    totalTracks INTEGER DEFAULT 0,
    CONSTRAINT pk_albums PRIMARY KEY (idAlbum),
    CONSTRAINT check_albums_type CHECK (typeAlbum IN ('compilation',
'single', 'album')),
    CONSTRAINT check_albums_anneeAlbum CHECK (anneeAlbum > 0),
    CONSTRAINT check_albums_totalTracks CHECK (totalTracks >= 0)
);

CREATE TABLE Copyrights (

```

```

        idCopyright INTEGER,
        descriptionCopyright VARCHAR(300) NOT NULL,
        CONSTRAINT pk_copyrights PRIMARY KEY (idCopyright)
    );

-- Association proteger
CREATE TABLE AlbumsCopyrights (
    idAlbum CHAR(22),
    idCopyright INTEGER,
    CONSTRAINT pk_albumsCopyrights PRIMARY KEY (idAlbum, idCopyright),
    CONSTRAINT fk_albumsCopyrht_idA_vers_Alb FOREIGN KEY (idAlbum)
REFERENCES Albums(idAlbum),
    CONSTRAINT fk_albumsCopyrht_idC_vers_Copy FOREIGN KEY (idCopyright)
REFERENCES Copyrights(idCopyright)
);

/*
Relations Tracks
*/

CREATE TABLE Tracks (
    idTrack CHAR(22),
    nomTrack VARCHAR(150) NOT NULL,
    idAlbum CHAR(22) NOT NULL,
    populariteTrack INTEGER NOT NULL,
    danceAbilite DECIMAL(4, 3),
    energie DECIMAL(4, 3),
    cleSignature INTEGER,
    intensiteSonore DECIMAL(5, 3),
    tonalite INTEGER,
    elocution DECIMAL(4, 3),
    acoustique DECIMAL(4, 1),
    instrumentalite DECIMAL(10, 9),
    vivacite DECIMAL(4, 3),
    ratioAppreciation DECIMAL(5, 4),
    tempo DECIMAL(6, 3),
    durationMs INTEGER,
    CONSTRAINT pk_tracks PRIMARY KEY (idTrack),
    CONSTRAINT fk_tracks_idAlbum_vers_albums FOREIGN KEY (idAlbum)
REFERENCES Albums(idAlbum),
    CONSTRAINT check_tracks_popularite CHECK (populariteTrack >= 0 AND
populariteTrack <= 100),

```



```

        CONSTRAINT check_tracks_danceAbilite CHECK (danceAbilite >= 0 AND
danceAbilite <= 1),
        CONSTRAINT check_tracks_energie CHECK (energie >= 0 AND energie <=
1),
        CONSTRAINT check_tracks_cleSignature CHECK (cleSignature >= 0),
        CONSTRAINT check_tracks_intensiteSonore CHECK (intensiteSonore >=
-60 AND intensiteSonore <= 0),
        CONSTRAINT check_tracks_tonalite CHECK (tonalite >= 0 AND tonalite
<= 1),
        CONSTRAINT check_tracks_elocution CHECK (elocution >= 0 AND
elocution <= 1),
        CONSTRAINT check_tracks_acoustique CHECK (acoustique >= 0 AND
acoustique <= 1),
        CONSTRAINT check_tracks_instrumentalite CHECK (instrumentalite >= 0
AND instrumentalite <= 1),
        CONSTRAINT check_tracks_vivacite CHECK (vivacite >= 0 AND vivacite
<= 1),
        CONSTRAINT check_tracks_ratioAppreciation CHECK (ratioAppreciation
>= 0 AND ratioAppreciation <= 1),
        CONSTRAINT check_Trackso CHECK (tempo >= 0),
        CONSTRAINT check_tracks_durationMs CHECK (durationMs >= 0)
);

-- BONUS: Mise à jour dynamique du nombre total de tracks
CREATE OR REPLACE PROCEDURE proc_tracks_decrement_totalTk (p_idAlbum
CHAR) AS
BEGIN
    UPDATE Albums
    SET totalTracks = totalTracks - 1 WHERE idAlbum = p_idAlbum;
END;
/

CREATE OR REPLACE PROCEDURE proc_tracks_increment_totalTk (p_idAlbum
CHAR) AS
BEGIN
    UPDATE Albums
    SET totalTracks = totalTracks + 1 WHERE idAlbum = p_idAlbum;
END;
/

CREATE OR REPLACE TRIGGER trg_tracks_insert
AFTER INSERT ON Tracks

```

```

        FOR EACH ROW
BEGIN
    proc_tracks_increment_totalTk(:NEW.idAlbum);
END;
/

CREATE OR REPLACE TRIGGER trg_tracks_delete
    AFTER DELETE ON Tracks
    FOR EACH ROW
BEGIN
    proc_tracks_decrement_totalTk(:OLD.idAlbum);
END;
/

CREATE OR REPLACE TRIGGER trg_tracks_update
    AFTER UPDATE ON Tracks
    FOR EACH ROW
BEGIN
    proc_tracks_decrement_totalTk(:OLD.idAlbum);
    proc_tracks_increment_totalTk(:NEW.idAlbum);
END;
/

-- Association interpreter
CREATE TABLE TracksInterpretes (
    idTrack CHAR(22),
    idArtiste CHAR(22),
    CONSTRAINT pk_tracksInterpretes PRIMARY KEY (idTrack, idArtiste),
    CONSTRAINT fk_tracksInts_idT_vers_tracks FOREIGN KEY (idTrack)
REFERENCES Tracks (idTrack),
    CONSTRAINT fk_tracksInts_idA_vers_ast FOREIGN KEY (idArtiste)
REFERENCES Artistes(idArtiste)
);

-- Association participer
CREATE TABLE AlbumsParticipants (
    idAlbum CHAR(22),
    idArtiste CHAR(22),
    CONSTRAINT pk_albumsParticipants PRIMARY KEY (idAlbum, idArtiste),
    CONSTRAINT fk_albumsPcps_idAl_vers_albums FOREIGN KEY (idAlbum)
REFERENCES Albums(idAlbum),
    CONSTRAINT fk_albumsPcps_idAr_vers_ast FOREIGN KEY (idArtiste)
REFERENCES Artistes(idArtiste)

```

```

);

-- Inclusion participer vers interpreter en fonction de idArtiste
CREATE OR REPLACE TRIGGER trg_albumsPcps_artiste_tkInts
BEFORE INSERT OR UPDATE OR DELETE ON AlbumsParticipants
FOR EACH ROW
DECLARE
    v_count INTEGER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM Tracks t
    JOIN TracksInterpretes ti ON t.idTrack = ti.idTrack
    WHERE t.idAlbum = :new.idAlbum AND ti.idArtiste = :new.idArtiste;

    IF v_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'L''artiste doit être
interprète d''au moins une piste de cet album.');
```

interprète d''au moins une piste de cet album.');

```

    END IF;
END;
/

/*
Relations Utilisateurs
*/

CREATE TABLE Utilisateurs (
    idUtilisateur INTEGER,
    pseudo VARCHAR(50) NOT NULL,
    dateNaissance DATE NOT NULL,
    sexe CHAR(1),
    tempsMsTrackLast INTEGER,
    idTrackLast CHAR(22),
    CONSTRAINT pk_utilisateurs PRIMARY KEY (idUtilisateur),
    CONSTRAINT fk_utilisateurs_idTL_vers_idT FOREIGN KEY (idTrackLast)
REFERENCES Tracks(idTrack)
);

CREATE TABLE Classiques (
    idUtilisateur INTEGER,
    CONSTRAINT pk_classiques PRIMARY KEY (idUtilisateur),
    CONSTRAINT fk_classiques_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur)
);

```

```

CREATE TABLE Premiums (
    idUtilisateur INTEGER,
    nomPremium VARCHAR(50) NOT NULL,
    surnomPremium VARCHAR(50),
    CONSTRAINT pk_premiums PRIMARY KEY (idUtilisateur),
    CONSTRAINT fk_premiums_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur)
);

-- Association suivre
CREATE TABLE UtilisateursSuivre (
    idUtilisateur INTEGER,
    idArtiste CHAR(22),
    CONSTRAINT pk_utSuivre PRIMARY KEY (idUtilisateur, idArtiste),
    CONSTRAINT fk_utSuivre_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur),
    CONSTRAINT fk_utSuivre_idA_vers_Artistes FOREIGN KEY (idArtiste)
REFERENCES Artistes(idArtiste)
);

-- BONUS: Mise à jour dynamique du nombre de followers d'un artiste
CREATE OR REPLACE PROCEDURE proc_ast_decrement_follower (p_idArtiste
CHAR) AS
BEGIN
    UPDATE Artistes
    SET nbFollowers = nbFollowers - 1 WHERE idArtiste = p_idArtiste;
END;
/

CREATE OR REPLACE PROCEDURE proc_ast_increment_follower (p_idArtiste
CHAR) AS
BEGIN
    UPDATE Artistes
    SET nbFollowers = nbFollowers + 1 WHERE idArtiste = p_idArtiste;
END;
/

CREATE OR REPLACE TRIGGER tgr_utilisateursSuivre_insert
AFTER INSERT ON UtilisateursSuivre
FOR EACH ROW
BEGIN
    proc_ast_increment_follower(:NEW.idArtiste);

```

```

END;
/

CREATE OR REPLACE TRIGGER tgr_utilisateursSuivre_delete
  AFTER DELETE ON UtilisateursSuivre
  FOR EACH ROW
BEGIN
  proc_ast_decrement_follower(:OLD.idArtiste);
END;
/

CREATE OR REPLACE TRIGGER tgr_utilisateursSuivre_update
  AFTER UPDATE ON UtilisateursSuivre
  FOR EACH ROW
BEGIN
  proc_ast_decrement_follower(:OLD.idArtiste);
  proc_ast_increment_follower(:NEW.idArtiste);
END;
/

-- Association etreAmi
CREATE TABLE PremiumsAmis (
  idUtilisateur INTEGER,
  idUtilisateurAmi INTEGER,
  CONSTRAINT pk_premAmis PRIMARY KEY (idUtilisateur,
idUtilisateurAmi),
  CONSTRAINT fk_premAmis_idUt_vers_prem FOREIGN KEY (idUtilisateur)
REFERENCES Premiums(idUtilisateur),
  CONSTRAINT fk_premAmis_idUtAmi_vers_prem FOREIGN KEY
(idUtilisateurAmi) REFERENCES Utilisateurs(idUtilisateur),
  CONSTRAINT check_premAmis_idUt CHECK (idUtilisateurAmi !=
idUtilisateur)
);

-- Association etreBloque
CREATE TABLE PremiumsBloques (
  idUtilisateur INTEGER,
  idUtilisateurBloque INTEGER,
  CONSTRAINT pk_premBlq PRIMARY KEY (idUtilisateur,
idUtilisateurBloque),
  CONSTRAINT fk_premBlq_idUt_vers_prem FOREIGN KEY (idUtilisateur)
REFERENCES Premiums(idUtilisateur),

```

```

        CONSTRAINT fk_premBlq_idUtBlq_vers_prem FOREIGN KEY
(idUtilisateurBloque) REFERENCES Utilisateurs(idUtilisateur),
        CONSTRAINT check_premBlq_idUt CHECK (idUtilisateurBloque !=
idUtilisateur)
);

-- Exclusion idUtilisateurBloque NOT IN etreAmi
CREATE OR REPLACE TRIGGER trg_premiumsBloques
BEFORE INSERT OR UPDATE ON PremiumsBloques
FOR EACH ROW
DECLARE
    nbCheck INTEGER;
BEGIN
    SELECT COUNT(*) INTO nbCheck FROM PremiumsAmis
    WHERE PremiumsAmis.idUtilisateur = :new.idUtilisateur AND
PremiumsAmis.idUtilisateurAmi = :new.idUtilisateurBloque;

    IF (nbCheck > 0) THEN
        RAISE_APPLICATION_ERROR(-20000, 'Un utilisateur ne peut pas
bloquer un utilisateur ami.');
```

```

    END IF;
END;
/

-- Exclusion idUtilisateurAmi NOT IN etreBloque
CREATE OR REPLACE TRIGGER trg_premiumsAmis
BEFORE INSERT OR UPDATE ON PremiumsAmis
FOR EACH ROW
DECLARE
    nbCheck INTEGER;
BEGIN
    SELECT COUNT(*) INTO nbCheck FROM PremiumsBloques
    WHERE PremiumsBloques.idUtilisateur = :new.idUtilisateur AND
PremiumsBloques.idUtilisateurBloque = :new.idUtilisateurAmi;

    IF (nbCheck > 0) THEN
        RAISE_APPLICATION_ERROR(-20000, 'Un utilisateur ne peut pas
ajouter en ami un utilisateur bloqué.');
```

```

    END IF;
END;
/

-- Association partager

```

```

CREATE TABLE PremiumsPartage (
    idUtilisateur INTEGER,
    idUtilisateurPartage INTEGER,
    idTrack CHAR(22),
    CONSTRAINT pk_premPtg PRIMARY KEY (idUtilisateur,
idUtilisateurPartage, idTrack),
    CONSTRAINT fk_premPtg_idUt_vers_prem FOREIGN KEY (idUtilisateur)
REFERENCES Premiums(idUtilisateur),
    CONSTRAINT fk_premPtg_idUtPrtg_vers_prem FOREIGN KEY
(idUtilisateurPartage) REFERENCES Premiums(idUtilisateur),
    CONSTRAINT fk_premPtg_idTrack_vers_tracks FOREIGN KEY (idTrack)
REFERENCES Tracks(idTrack),
    CONSTRAINT check_premPtg_idUt CHECK (idUtilisateurPartage !=
idUtilisateur)
);

-- Association écouter
CREATE TABLE UtilisateursHistorique (
    idUtilisateur INTEGER,
    idTrack CHAR(22),
    dateEcoute TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_utHist PRIMARY KEY (idUtilisateur, idTrack,
dateEcoute),
    CONSTRAINT fk_utHist_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur),
    CONSTRAINT fk_utHist_idTrack_vers_tracks FOREIGN KEY (idTrack)
REFERENCES Tracks(idTrack)
);

-- Association aimer
CREATE TABLE UtilisateursLike (
    idUtilisateur INTEGER,
    idTrack CHAR(22),
    dateLike DATE DEFAULT CURRENT_DATE,
    CONSTRAINT pk_utLike PRIMARY KEY (idUtilisateur, idTrack),
    CONSTRAINT fk_utLike_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur),
    CONSTRAINT fk_utLike_idTrack_vers_tracks FOREIGN KEY (idTrack)
REFERENCES Tracks(idTrack)
);

-- Association évaluer
CREATE TABLE UtilisateursEvaluations (

```

```

    idEvaluation INTEGER,
    idUtilisateur INTEGER NOT NULL,
    idTrack CHAR(22) NOT NULL,
    note INTEGER NOT NULL,
    CONSTRAINT pk_utEval PRIMARY KEY (idEvaluation),
    CONSTRAINT fk_utEval_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur),
    CONSTRAINT fk_utEval_idTrack_vers_tracks FOREIGN KEY (idTrack)
REFERENCES Tracks(idTrack),
    CONSTRAINT check_utEval_note CHECK (note >= 0 AND note <= 20)
);

-- Auto-incrémentation de UtilisateursEvaluations
CREATE SEQUENCE Seq_UtEvaluations START WITH 1;

CREATE OR REPLACE TRIGGER tgr_utEvaluations
    BEFORE INSERT ON UtilisateursEvaluations
    FOR EACH ROW
BEGIN
    SELECT Seq_UtEvaluations.NEXTVAL INTO :new.idEvaluation FROM dual;
END;
/

/*
Relations Playlists
*/

CREATE TABLE ConteneursPlaylists (
    idConteneur INTEGER,
    nomConteneur VARCHAR(50) NOT NULL,
    typeConteneur VARCHAR(8) NOT NULL,
    idDossierParent INTEGER,
    idUtilisateur INTEGER NOT NULL,
    CONSTRAINT pk_ctnPlay PRIMARY KEY (idConteneur),
    CONSTRAINT fk_ctnPlay_idUt_vers_Ut FOREIGN KEY (idUtilisateur)
REFERENCES Utilisateurs(idUtilisateur),
    CONSTRAINT check_conteneursPlay_type CHECK (typeConteneur IN
('Playlist', 'Folder')),
    CONSTRAINT check_conteneursPlay_idD CHECK (idDossierParent !=
idConteneur)
);

```



```

CREATE TABLE Playlists (
    idPlaylist INTEGER,
    CONSTRAINT pk_playlists PRIMARY KEY (idPlaylist),
    CONSTRAINT fk_playlists_idP_vers_ctnPlay FOREIGN KEY (idPlaylist)
REFERENCES ConteneursPlaylists(idConteneur)
);

CREATE TABLE DossiersPlaylists (
    idDossier INTEGER,
    CONSTRAINT pk_dossiersPlaylists PRIMARY KEY (idDossier)
);

-- References circulaires Playlists
ALTER TABLE ConteneursPlaylists ADD CONSTRAINT
fk_ctnPlay_idD_vers_dossPlay FOREIGN KEY (idDossierParent) REFERENCES
DossiersPlaylists(idDossier);
ALTER TABLE DossiersPlaylists ADD CONSTRAINT
fk_dossPlay_idD_vers_ctnPlay FOREIGN KEY (idDossier) REFERENCES
ConteneursPlaylists(idConteneur);

-- Association ajouter
CREATE TABLE PlaylistsTracks (
    idPlaylist INTEGER,
    ordre INTEGER,
    idTrack CHAR(22) NOT NULL,
    CONSTRAINT pk_playTracks PRIMARY KEY (idPlaylist, ordre),
    CONSTRAINT fk_playTracks_idP_vers_play FOREIGN KEY (idPlaylist)
REFERENCES Playlists(idPlaylist),
    CONSTRAINT fk_playTracks_idT_vers_tracks FOREIGN KEY (idTrack)
REFERENCES Tracks(idTrack),
    CONSTRAINT check_playTracks_ordre CHECK (ordre > 0)
);

-- Sequence & Trigger Genres
CREATE SEQUENCE Seq_Genres START WITH 1;

CREATE OR REPLACE TRIGGER trg_genres_auto_increment
BEFORE INSERT ON Genres
FOR EACH ROW
BEGIN
    SELECT Seq_Genres.NEXTVAL INTO :new.idGenre FROM dual;
END;
/

```

```

-- Sequence & Trigger Prix
CREATE SEQUENCE Seq_Prix START WITH 1;

CREATE OR REPLACE TRIGGER trg_prix_auto_increment
  BEFORE INSERT ON Prix
  FOR EACH ROW
BEGIN
  SELECT Seq_Prix.NEXTVAL INTO :new.idPrix FROM dual;
END;
/

-- Sequence & Trigger ConteneursPlaylists
CREATE SEQUENCE Seq_ConteneursPlaylists START WITH 1;

CREATE OR REPLACE TRIGGER trg_ctnPlay_auto_increment
  BEFORE INSERT ON ConteneursPlaylists
  FOR EACH ROW
BEGIN
  SELECT Seq_ConteneursPlaylists.NEXTVAL INTO :new.idConteneur FROM
dual;
END;
/

-- Trigger ordre association ajouter
CREATE OR REPLACE TRIGGER trg_playlistsTracks_ordre
  BEFORE INSERT ON PlaylistsTracks
  FOR EACH ROW
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  maxCurrentOrdre INTEGER;
  nbCurrentTrack INTEGER;
BEGIN
  SELECT MAX(ordre) INTO maxCurrentOrdre FROM PlaylistsTracks WHERE
PlaylistsTracks.idPlaylist = :new.idPlaylist;
  SELECT COUNT(*) INTO nbCurrentTrack FROM PlaylistsTracks WHERE
PlaylistsTracks.idPlaylist = :new.idPlaylist AND
PlaylistsTracks.idTrack = :new.idTrack;

  IF (:new.ordre != maxCurrentOrdre + 1) THEN
    RAISE_APPLICATION_ERROR(-20000, 'L'ordre naturel des tracks
dans une même playlist doit être préservé. ');
  END IF;

```

```
    IF (nbCurrentTrack > 0) THEN
        RAISE_APPLICATION_ERROR(-20000, 'Ne peut pas avoir la meme
track plusieurs fois dans la meme playlist');
    END IF;
END;
/
```

Script d'insertion des données

```
/*
    Relations Artistes
*/

INSERT INTO Artistes (idArtiste, nomArtiste, populariteArtiste)
SELECT DISTINCT idArtist, name, popularity FROM Artists_Temp;

INSERT INTO Genres (nomGenre)
SELECT DISTINCT genre FROM Artists_Temp
WHERE genre IS NOT NULL;

INSERT INTO GenresArtistes
SELECT DISTINCT idArtist, idGenre FROM Artists_Temp at
JOIN Genres g ON at.genre = g.nomGenre;

INSERT INTO CategoriesPrix
SELECT DISTINCT category FROM Artists_Temp
WHERE category IS NOT NULL;

INSERT INTO Prix (nomPrix, nomCategorie)
SELECT DISTINCT award, category FROM Artists_Temp
WHERE award IS NOT NULL AND category IS NOT NULL;

INSERT INTO PrixArtistes
SELECT DISTINCT idPrix, year, idArtist FROM Artists_Temp at
JOIN Prix p ON at.award = p.nomPrix
WHERE year IS NOT NULL AND idArtist IS NOT NULL;

/*
    Relations Albums
*/

INSERT INTO Albums (idAlbum, nomAlbum, labelAlbum, typeAlbum,
anneeAlbum)
SELECT DISTINCT idAlbum, name, label, type, year FROM Albums_Temp;

INSERT INTO Copyrights
SELECT DISTINCT idCopyright, copyright FROM Albums_Temp;
```

```

INSERT INTO AlbumsCopyrights
SELECT DISTINCT idAlbum, idCopyright FROM Albums_Temp;

/*
Relations Tracks
*/

INSERT INTO Tracks
SELECT DISTINCT idTrack, name, idAlbum, popularity, danceAbility,
energy, key_signature, loudness, "mode",
speechiness, acousticness, instrumentalness, liveness,
valence, tempo, duration_ms FROM Tracks_Temp;

INSERT INTO TracksInterpretes
SELECT DISTINCT idTrack, idArtist FROM Tracks_Temp;

INSERT INTO AlbumsParticipants
SELECT DISTINCT idAlbum, idArtist FROM Albums_Temp;

/*
Relations Utilisateurs
*/

INSERT INTO Utilisateurs
SELECT DISTINCT idUser, pseudo, dateOfBirth, gender, timePlaying,
idCurrentTrack FROM Users_Temp;

INSERT INTO Classiques
SELECT DISTINCT idUser FROM Users_Temp
WHERE name IS NULL;

INSERT INTO Premiums
SELECT DISTINCT idUser, name, surname FROM Users_Temp
WHERE name IS NOT NULL;

INSERT INTO UtilisateursSuivre
SELECT DISTINCT idUser, idArtistFollow FROM Users_Temp
WHERE idArtistFollow IS NOT NULL;

INSERT INTO PremiumsAmis
SELECT DISTINCT idUserPremium, idUserFriend FROM Friends_Temp;

```

```

INSERT INTO PremiumsBloques
SELECT DISTINCT idUserPremium, idUserBlocked FROM Users_Blocked_Temp;

INSERT INTO PremiumsPartage
SELECT DISTINCT idUser, idUserShare, idTrack FROM Share_Temp;

INSERT INTO UtilisateursHistorique
SELECT DISTINCT idUser, idTrack, dateListen FROM Tracks_Temp
WHERE dateListen IS NOT NULL;

INSERT INTO UtilisateursLike
SELECT DISTINCT idUser, idTrack, dateLike FROM Likes_Temp;

INSERT INTO UtilisateursEvaluations (idUser, idTrack, note)
SELECT DISTINCT idUser, idTrack, note FROM Evaluations_Temp;

/*
  Relations Playlists
*/

-- Insérer les conteneurs sans parent
INSERT INTO ConteneursPlaylists (nomConteneur, typeConteneur,
idUser)
SELECT DISTINCT nameElement, type, idUser FROM Playlists_Temp;

-- Insérer le conteneur type & affecter les parents aux conteneurs
DECLARE
  id INTEGER;
BEGIN
  FOR x IN (SELECT DISTINCT idUser, nameElement, nameElementParent,
type FROM Playlists_Temp) LOOP
    SELECT idConteneur INTO id FROM ConteneursPlaylists WHERE
nomConteneur = x.nameElement AND idUtilisateur = x.idUser;

    -- Insertion du conteneur type
    IF x.type = 'Playlist' THEN
      INSERT INTO Playlists VALUES (id);
    ELSIF x.type = 'Folder' THEN
      INSERT INTO DossiersPlaylists VALUES (id);
    END IF;
  END LOOP;
END LOOP;

```

```

    FOR x IN (SELECT DISTINCT idUser, nameElement, nameElementParent,
type FROM Playlists_Temp) LOOP
        SELECT idConteneur INTO id FROM ConteneursPlaylists WHERE
nomConteneur = x.nameElement AND idUtilisateur = x.idUser;

        -- Affectation du parent
        UPDATE ConteneursPlaylists SET idDossierParent = (SELECT
idConteneur FROM ConteneursPlaylists WHERE nomConteneur =
x.nameElementParent AND idUtilisateur = x.idUser)
        WHERE x.nameElementParent IS NOT NULL AND idConteneur = (id);
    END LOOP;
END;

INSERT INTO PlaylistsTracks
SELECT DISTINCT idConteneur, ordre, idTrack FROM Playlists_Temp pt
JOIN ConteneursPlaylists cp ON pt.nameElement = cp.nomConteneur
WHERE type = 'Playlist' AND idUtilisateur = idUser
ORDER BY ordre;
-- L'insertion de cette requête prend environ 2 minutes

COMMIT;

```

Script d'altération et suppression

```
ALTER TABLE DossiersPlaylists DROP CONSTRAINT
fk_dossPlay_idD_vers_ctnPlay;
ALTER TABLE ConteneursPlaylists DROP CONSTRAINT
fk_ctnPlay_idD_vers_dossPlay;
DROP SEQUENCE Seq_Prix;
DROP SEQUENCE Seq_Genres;
DROP SEQUENCE Seq_UtEvaluations;
DROP SEQUENCE Seq_ConteneursPlaylists;
DROP PROCEDURE proc_ast_increment_follower;
DROP PROCEDURE proc_ast_decrement_follower;
DROP PROCEDURE proc_tracks_increment_totalTk;
DROP PROCEDURE proc_tracks_decrement_totalTk;

DROP TABLE PlaylistsTracks;
DROP TABLE DossiersPlaylists;
DROP TABLE Playlists;
DROP TABLE ConteneursPlaylists;

DROP TABLE UtilisateursEvaluations;
DROP TABLE UtilisateursLike;
DROP TABLE UtilisateursHistorique;
DROP TABLE PremiumsPartage;
DROP TABLE PremiumsBloques;
DROP TABLE PremiumsAmis;
DROP TABLE UtilisateursSuivre;
DROP TABLE Premiums;
DROP TABLE Classiques;
DROP TABLE Utilisateurs;

DROP TABLE AlbumsParticipants;
DROP TABLE TracksInterpretes;
DROP TABLE Tracks;
DROP TABLE AlbumsCopyrights;
DROP TABLE Copyrights;
DROP TABLE Albums;

DROP TABLE PrixArtistes;
DROP TABLE Prix;
DROP TABLE CategoriesPrix;
DROP TABLE GenresArtistes;
```



```
DROP TABLE Genres;  
DROP TABLE Artistes;
```