



PROJET ORACLE

Application de streaming musical (Spotify-like)

JULIEN Loïs, LEFEVRE Clément, SAUVÉE Étienne – IATC 4 2018/2019

Table des matières

| | |
|--|-----------|
| TABLE DES MATIERES | 1 |
| INTRODUCTION..... | 2 |
| MODELISATION DE LA BASE DE DONNEES | 3 |
| A. MODELE ENTITE-ASSOCIATION | 3 |
| B. MODELE RELATIONNEL | 4 |
| MISE EN PLACE DE LA BASE DE DONNEES | 5 |
| A. CREATIONS DES TABLES | 5 |
| B. CREATIONS DES INDEXES | 9 |
| C. TRIGGERS – REGLES DE GESTION | 10 |
| 1. TRIGGERS D’AUTO-INCREMENT | 10 |
| 2. TRIGGERS D’INTEGRITE..... | 11 |
| 3. SUPPRESSION DES DONNEES | 15 |
| D. JEU DE DONNEES..... | 18 |
| 1. INSERT INTO | 19 |
| 2. SQL*LOADER | 20 |
| 3. SCRIPT PL/SQL | 21 |
| 4. AUTRES..... | 21 |
| MANIPULATION DES DONNEES | 22 |
| A. VUES..... | 22 |
| B. REQUETES | 26 |
| SQL AVANCE | 32 |
| A. METADONNEES..... | 32 |
| B. ANALYSE ET ADMINISTRATION PHYSIQUE DES DONNEES | 33 |
| CONCLUSION | 36 |
| WEBOGRAPHIE | 36 |

Introduction

Ce rapport traite du projet de Base de Données - Approfondissement, module enseigné par Mme Finance, dans le cadre de notre seconde année de cycle ingénieur informatique (IATIC 4) à l'ISTY (Institut des Sciences et Techniques des Yvelines).

Le sujet était le suivant :

Le but de ce projet est de construire une base de données sous Oracle afin de mettre en pratique les différentes fonctionnalités standard d'un SGBD. Le projet se divise en trois parties décrites ci-après et sera réalisé par groupe de 3 étudiants au plus.

Comme demandé, nous avons formé un groupe de trois personnes : Lois JULLIEN, Clément LEFEVRE et Etienne SAUVEE.

Rapidement, nous avons décidé de nous intéresser aux applications de streaming musical (telles que Spotify, Deezer, Napster, etc.).

Ayant déjà préalablement utilisé ce type d'application par le passé, nous connaissions globalement leur fonctionnement, ce qui nous a facilité la modélisation de la base de données relationnelle.

Ce projet était à réaliser sur le SGBD Oracle, via le programme SQL*Plus, que nous avons découvert et pratiqué cette année, durant les travaux pratiques de ce module.

Au sein de ce rapport, nous utiliserons le cheminement suivant :

Dans un premier temps nous détaillerons la modélisation de la base de données, avant de traiter de sa création, puis de son remplissage et enfin de quelques opérations de « gestion avancée ».

Modélisation de la base de données

A. Modèle entité-association

Pour des raisons de lisibilité, nous avons choisi de ne pas représenter les attributs des tables sur ce modèle. Ils sont néanmoins mentionnés sur le modèle relationnel ci-après.

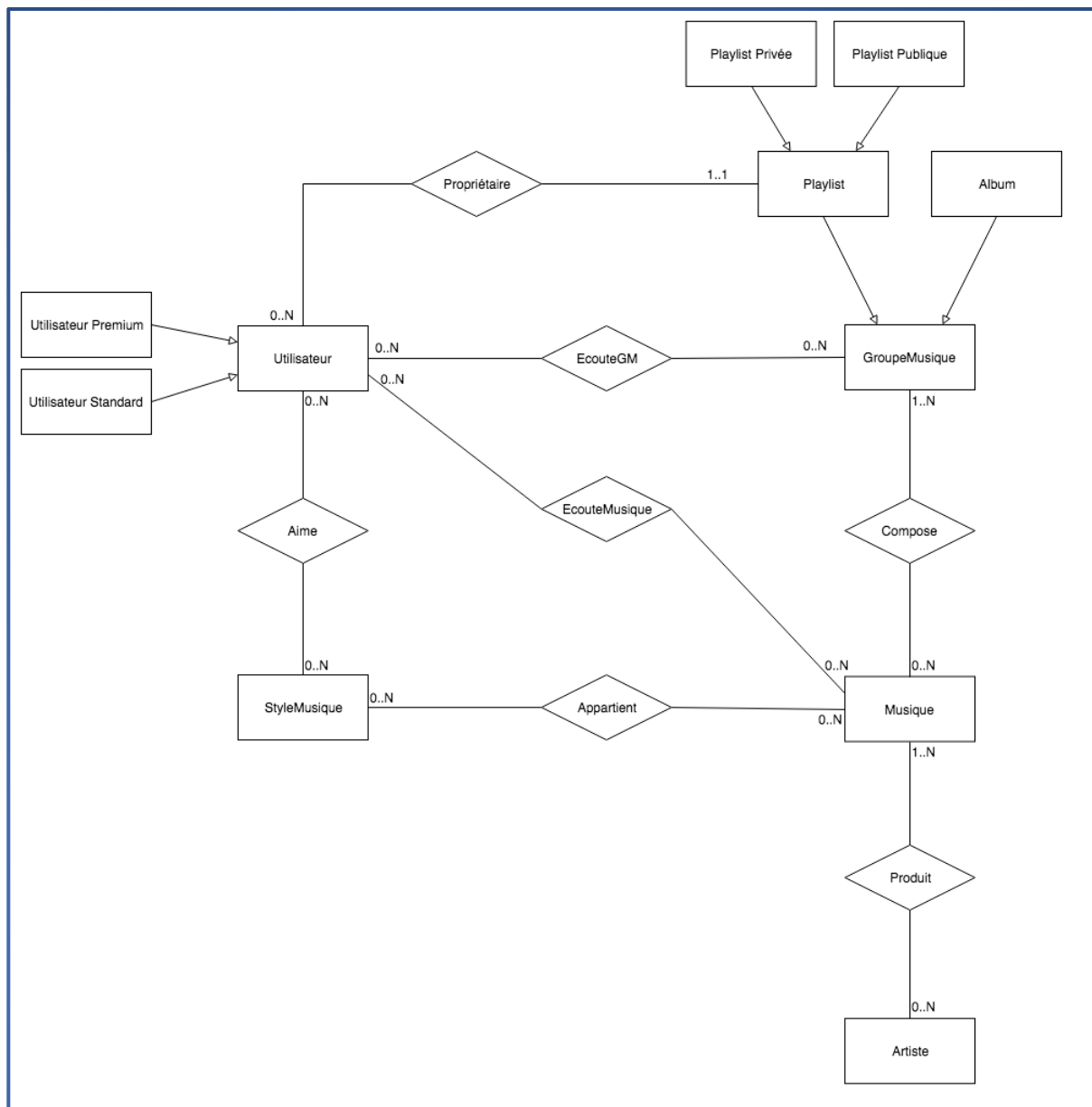


Figure 1 : Modèle entité-association de notre base de données (Application de streaming musical)

B. Modèle relationnel

A partir du modèle entité-association précédent, nous avons déterminé le modèle relationnel suivant :

```
UTILISATEUR(IdUtilisateur, Pseudo, Nom, Prénom, E-Mail, Nationalité, Sexe, EstPremium)
GROUPEMUSIQUE(IdGroupeMusique, Titre, EstPlaylist, EstPrivé, DateCreation)
MUSIQUE(IdMusique, Titre, Durée, DatePublication)
ARTISTE(IdArtiste, NomScene, Nationalité)
STYLEMUSIQUE(IdStyle, Nom)
PROPRIETAIRE(#IdUtilisateur, #IdGroupeMusique)
ECOUTEGM(#IdUtilisateur, #IdGroupeMusique, HeureEcoule, OrdreEcoule)
ECOUTEMUSIQUE(#IdUtilisateur, #IdMusique, HeureEcoule)
COMPOSE(#IdGroupeMusique, #IdMusique)
PRODUIT(#IdMusique, #IdArtiste)
APPARTIENT(#IdMusique, #IdStyleMusique)
AIME(#IdStyleMusique, #IdUtilisateur)
```

Figure 2 : Modèle relationnel de notre base de données (Application de streaming musical)

Mise en place de la base de données

A. Créations des tables

UTILISATEUR :

```
CREATE TABLE Utilisateur (
  IdUtilisateur int NOT NULL PRIMARY KEY,
  Pseudo char(20) NOT NULL,
  Nom char(15) NOT NULL,
  Prenom char(15) NOT NULL,
  Email char(40) CHECK (Email like '%_@_%._%') NOT NULL,
  Nationalite char(2) NOT NULL,
  DateNaissance date NOT NULL,
  Sexe char(1) CHECK (Sexe = 'M' OR Sexe = 'F') NOT NULL,
  EstPremium char(1) CHECK (EstPremium = 'Y' OR EstPremium = 'N') NOT NULL,
  CONSTRAINT CHK_NatUser CHECK (Nationalite IN ('DZ', 'DE', 'AR', 'AM', 'AU', 'AT', 'BD', 'BE',
    'BO', 'BR', 'KH', 'CM', 'CA', 'CL', 'CN', 'CY', 'CO', 'CG', 'CD', 'KR', 'KP', 'CR', 'CI',
    'HR', 'CU', 'DK', 'DO', 'EG', 'AE', 'EC', 'ES', 'EE', 'US', 'ET', 'FJ', 'FI', 'FR', 'GA',
    'GM', 'GE', 'GH', 'GI', 'GR', 'HT', 'HN', 'HK', 'IN', 'ID', 'IR', 'IQ', 'IE', 'IS', 'IL',
    'IT', 'JM', 'JP', 'JO', 'KE', 'LV', 'LB', 'LY', 'LI', 'LT', 'LU', 'MG', 'ML', 'MT', 'MA',
    'MX', 'MN', 'NA', 'NP', 'NE', 'NG', 'NO', 'NZ', 'OM', 'NL', 'PE', 'PL', 'PT', 'QA', 'RO',
    'RU', 'GB', 'SN', 'RS', 'SK', 'SI', 'SO', 'SD', 'SE', 'CH', 'TW', 'TD', 'TH', 'TO', 'TN',
    'TR', 'UA', 'UY', 'VU', 'VN', 'VE', 'ZM', 'ZW')));
```

Figure 3 : Requête de création de la table Utilisateur

Cette table possède quelques particularités :

- L'attribut Email possède une contrainte de domaine, correspondant à l'expression régulière suivante : '%_@_%._%'.
Le caractère _ correspond à un caractère quelconque simple, et le caractère % correspond à 0 ou plusieurs caractères quelconques.
Ceci correspond donc à la forme générale d'une adresse Email, qui doit contenir au moins un caractère avant l'@, puis au moins deux caractères, puis un . et enfin au moins deux caractères.
- L'attribut Nationalité possède également une contrainte de domaine particulière. Nous avons en effet souhaité utiliser la norme ISO-ALPHA-2, qui permet d'identifier un pays par une chaîne de deux lettres.
De fait, il nous est apparu judicieux de vérifier la validité de la nationalité saisie, en vérifiant la présence de la chaîne dans le tuple.
(NB : Nous n'avons saisi que les codes des principaux pays)
- Toutes les colonnes doivent être non-nulles.

GROUPEMUSIQUE :

```
CREATE TABLE GroupeMusique (
  IdGroupeMusique int PRIMARY KEY,
  Titre char(20) NOT NULL,
  NombreMusiques int NOT NULL,
  EstPlaylist char(1) CHECK (EstPlaylist = 'Y' OR EstPlaylist = 'N') NOT NULL,
  EstPrive char(1) CHECK (EstPrive = 'Y' OR EstPrive = 'N') NOT NULL,
  DateCreation date NOT NULL,
  CONSTRAINT CHK_AlbumPublic CHECK ((EstPlaylist = 'N' AND EstPrive = 'N') OR EstPlaylist = 'Y'));
```

Figure 4 : Requête de création de la table GroupeMusique

Oracle ne prenant pas en charge le type booléen, nous avons choisi de le remplacer par un char(1), en y appliquant des contraintes de domaines ('Y' pour Yes et 'N' pour No).

MUSIQUE :

```
CREATE TABLE Musique (
  IdMusique int PRIMARY KEY,
  Titre char(35) NOT NULL,
  Duree decimal(5,2) NOT NULL,
  DatePublication Date NOT NULL);
```

Figure 5 : Requête de création de la table Musique

Nous avons choisi le type *decimal(5,2)* pour la durée d'une musique, ce qui correspond à un nombre de la forme MMM.SS.

ARTISTE :

```
CREATE TABLE Artiste (
  IdArtiste int PRIMARY KEY,
  NomScene char(20) NOT NULL,
  Nationalite char(2) NOT NULL,
  CONSTRAINT CHK_NatArt CHECK (Nationalite IN ('DZ', 'DE', 'AR', 'AM', 'AU', 'AT', 'BD', 'BE',
  'BO', 'BR', 'KH', 'CM', 'CA', 'CL', 'CN', 'CY', 'CO', 'CG', 'CD', 'KR', 'KP', 'CR', 'CI',
  'HR', 'CU', 'DK', 'DO', 'EG', 'AE', 'EC', 'ES', 'EE', 'US', 'ET', 'FJ', 'FI', 'FR', 'GA',
  'GM', 'GE', 'GH', 'GI', 'GR', 'HT', 'HN', 'HK', 'IN', 'ID', 'IR', 'IQ', 'IE', 'IS', 'IL',
  'IT', 'JM', 'JP', 'JO', 'KE', 'LV', 'LB', 'LY', 'LI', 'LT', 'LU', 'MG', 'ML', 'MT', 'MA',
  'MX', 'MN', 'NA', 'NP', 'NE', 'NG', 'NO', 'NZ', 'OM', 'NL', 'PE', 'PL', 'PT', 'QA', 'RO',
  'RU', 'GB', 'SN', 'RS', 'SK', 'SI', 'SO', 'SD', 'SE', 'CH', 'TW', 'TD', 'TH', 'TO', 'TN',
  'TR', 'UA', 'UY', 'VU', 'VN', 'VE', 'ZM', 'ZW')));
```

Figure 6 : Requête de création de la table Artiste

Nous avons utilisé ici la même contrainte que pour la nationalité de l'utilisateur.

STYLEMUSIQUE :

```
CREATE TABLE StyleMusique (
  IdStyleMusique int PRIMARY KEY,
  NomStyle char(15) NOT NULL);
```

Figure 7 : Requête de création de la table StyleMusique

ECOUTEGM :

```
CREATE TABLE EcouleGM (
  IdUtilisateur int NOT NULL,
  IdGroupeMusique int NOT NULL,
  HeureEcoule timestamp NOT NULL,
  OrdreEcoule char(4) DEFAULT 'ASC' CHECK (OrdreEcoule = 'DESC' OR OrdreEcoule = 'ASC' OR OrdreEcoule = 'RAND') NOT NULL,
  FOREIGN KEY (IdUtilisateur) REFERENCES Utilisateur(IdUtilisateur),
  FOREIGN KEY (IdGroupeMusique) REFERENCES GroupeMusique(IdGroupeMusique),
  PRIMARY KEY (IdUtilisateur, IdGroupeMusique, HeureEcoule));
```

Figure 8 : Requête de création de la table EcouleGM

Nous avons défini plusieurs ordres d'écoute pour un groupe de musique : 'ASC' pour ascendant, 'DESC' pour descendant et 'RAND' pour une écoute aléatoire. Le mot clé DEFAULT nous permet de définir la valeur par défaut de cet ordre (ici nous avons choisi 'ASC').

De plus, la clé primaire de *EcouleGM* est composée des deux clés étrangères *IdUtilisateur* et *IdGroupeMusique*, mais également d'une heure d'écoute (de type *Timestamp*, de la forme 'JJ-MMM-AAAA hh.mm.ss').

ECOUTEMUSIQUE :

```
CREATE TABLE EcouleMusique (
  IdUtilisateur int NOT NULL,
  IdMusique int NOT NULL,
  HeureEcoule timestamp NULL,
  FOREIGN KEY (IdUtilisateur) REFERENCES Utilisateur(IdUtilisateur),
  FOREIGN KEY (IdMusique) REFERENCES Musique(IdMusique),
  PRIMARY KEY (IdUtilisateur, IdMusique, HeureEcoule));
```

Figure 9 : Requête de création de la table EcouleMusique

Le principe est le même que pour *EcouleGM*, la clé primaire est donc également une composition des deux clés étrangères identifiant l'*Utilisateur* et le *GroupeMusique* et de l'heure d'écoute.

COMPOSE :

```
CREATE TABLE Compose (
  IdGroupeMusique int NOT NULL,
  IdMusique int NOT NULL,
  FOREIGN KEY (IdGroupeMusique) REFERENCES GroupeMusique(IdGroupeMusique),
  FOREIGN KEY (IdMusique) REFERENCES Musique(IdMusique),
  PRIMARY KEY (IdGroupeMusique, IdMusique));
```

Figure 10 : Requête de création de la table Compose

PRODUIT :

```
CREATE TABLE Produit (  
    IdArtiste int NOT NULL,  
    IdMusique int NOT NULL,  
    FOREIGN KEY (IdArtiste) REFERENCES Artiste(IdArtiste),  
    FOREIGN KEY (IdMusique) REFERENCES Musique(IdMusique),  
    PRIMARY KEY (IdArtiste,IdMusique));
```

Figure 11 : Requête de création de la table Produit

APPARTIENT :

```
CREATE TABLE Appartient (  
    IdMusique int NOT NULL,  
    IdStyleMusique int NOT NULL,  
    FOREIGN KEY (IdMusique) REFERENCES Musique(IdMusique),  
    FOREIGN KEY (IdStyleMusique) REFERENCES StyleMusique(IdStyleMusique),  
    PRIMARY KEY (IdMusique,IdStyleMusique));
```

Figure 12 : Requête de création de la table Appartient

AIME :

```
CREATE TABLE Aime (  
    IdUtilisateur int NOT NULL,  
    IdStyleMusique int NOT NULL,  
    FOREIGN KEY (IdUtilisateur) REFERENCES Utilisateur(IdUtilisateur),  
    FOREIGN KEY (IdStyleMusique) REFERENCES StyleMusique(IdStyleMusique),  
    PRIMARY KEY (IdUtilisateur,IdStyleMusique));
```

Figure 13 : Requête de création de la table Aime

PROPRIÉTAIRE :

```
CREATE TABLE Proprietaire (  
    IdUtilisateur int NOT NULL,  
    IdGroupeMusique int NOT NULL,  
    FOREIGN KEY (IdUtilisateur) REFERENCES Utilisateur(IdUtilisateur),  
    FOREIGN KEY (IdGroupeMusique) REFERENCES GroupeMusique(IdGroupeMusique),  
    PRIMARY KEY (IdUtilisateur,IdGroupeMusique));
```

Figure 14 : Requête de création de la table Propriétaire

B. Créations des indexes

Les indexes permettent d'accéder plus rapidement aux données, et donc d'améliorer les performances de notre application.

Cependant, les indexes ont un coût particulier pour les opérations INSERT INTO, UPDATE et DELETE, c'est pourquoi il est important de les choisir judicieusement.

Voici donc une liste non-exhaustive des indexes mis en place pour notre base de données.

```
CREATE INDEX idx_pseudos ON Utilisateur(idUtilisateur,Pseudo);
CREATE INDEX idx_nomscene ON Artiste(idArtiste,NomScene);
CREATE INDEX idx_id_titre_musique ON Musique(idMusique,Titre);
CREATE INDEX idx_titre_grpmusique ON GroupeMusique(idGroupeMusique,Titre);
CREATE INDEX idx_email_pseudos ON Utilisateur(idUtilisateur,Email,Pseudo);
CREATE INDEX idx_premium_users ON Utilisateur(IdUtilisateur,Pseudo,EstPremium);
CREATE INDEX idx_ecoutes_m ON EcouteMusique(IdUtilisateur, IdMusique);
CREATE INDEX idx_ecoute_gm ON EcouteGroupeMusique(IdUtilisateur,IdGroupeMusique);
CREATE INDEX idx_compose ON Compose(IdMusique,IdGroupeMusique);
CREATE INDEX idx_appartient ON Appartient(IdMusique, IdStyleMusique);
CREATE INDEX idx_style ON StyleMusique(IdStyleMusique, NomStyle);
CREATE INDEX idx_aime ON Aime(IdUtilisateur,IdStyleMusique);
```

Figure 15 : Indexes de notre base de données

C. Triggers – Règles de gestion

Avant même d'insérer nos données dans les tables précédemment créées, nous avons défini des triggers, garantissant leur intégrité.

1. Triggers d'auto-incrément

Dans un premier temps, nous avons mis en place des triggers sur certaines de nos tables afin d'imiter le concept d'auto-incrément, absent sur Oracle.

Pour se faire, on procède de la manière suivante, pour chacune des tables concernées :

- On crée une séquence, qui générera les identifiants de la table
- On définit un trigger sur cette table, qui avant chaque insertion, insèrera la valeur suivante de la séquence, en tant qu'identifiant de la ligne.

```
CREATE SEQUENCE SEQ_USERS START WITH 1;

CREATE OR REPLACE TRIGGER AI_USERS
BEFORE INSERT ON Utilisateur
FOR EACH ROW
BEGIN
    SELECT SEQ_USERS.NEXTVAL
    INTO :NEW.IdUtilisateur
    FROM DUAL;
END;
/
```

Figure 18 : Auto-incrément sur la table Utilisateur

```
CREATE SEQUENCE SEQ_ARTISTS START WITH 1;

CREATE OR REPLACE TRIGGER AI_ARTISTS
BEFORE INSERT ON Artiste
FOR EACH ROW
BEGIN
    SELECT SEQ_ARTISTS.NEXTVAL
    INTO :NEW.IdArtiste
    FROM DUAL;
END;
/
```

Figure 17 : Auto-incrément sur la table Artiste

```
CREATE SEQUENCE SEQ_MUSIC START WITH 1;

CREATE OR REPLACE TRIGGER AI_MUSIC
BEFORE INSERT ON Musique
FOR EACH ROW
BEGIN
    SELECT SEQ_MUSIC.NEXTVAL
    INTO :NEW.IdMusique
    FROM DUAL;
END;
/
```

Figure 16 : Auto-incrément sur la table Musique

```
CREATE SEQUENCE SEQ_GM START WITH 1;

CREATE OR REPLACE TRIGGER AI_GM
BEFORE INSERT ON GroupeMusique
FOR EACH ROW
BEGIN
    SELECT SEQ_GM.NEXTVAL
    INTO :NEW.IdGroupeMusique
    FROM DUAL;
END;
/
```

Figure 20 : Auto-incrément sur la table GroupeMusique

```
CREATE SEQUENCE SEQ_SM START WITH 1;

CREATE OR REPLACE TRIGGER AI_SM
BEFORE INSERT ON StyleMusique
FOR EACH ROW
BEGIN
    SELECT SEQ_SM.NEXTVAL
    INTO :NEW.IdStyleMusique
    FROM DUAL;
END;
/
```

Figure 19 : Auto-incrément sur la table StyleMusique

2. Triggers d'intégrité

CHK PROPRIETAIRE VALIDE :

Règles de gestions :

- Un utilisateur ne peut être propriétaire que d'une playlist (pas d'un album)
- Une playlist ne possède qu'un et un seul propriétaire

```
CREATE OR REPLACE TRIGGER CHK_PROPRIETAIRE_VALIDE
BEFORE INSERT OR UPDATE ON Proprietaire
FOR EACH ROW
DECLARE
  Prop int;
  EstPL char(1);
BEGIN
  SELECT COUNT(*) INTO Prop
  FROM Proprietaire P
  WHERE P.IdGroupeMusique = :New.IdGroupeMusique;

  SELECT GM.EstPlaylist INTO EstPL
  FROM GroupeMusique GM
  WHERE GM.IdGroupeMusique = :New.IdGroupeMusique;

  IF EstPL = 'N' THEN
    RAISE_APPLICATION_ERROR(-20001, 'Le groupe de musique numero ' || :New.IdGroupeMusique || ' est un album.');

```

Figure 21 : Trigger sur la table Propriétaire

Ce premier trigger permet de vérifier les deux règles de gestions préalablement citées, et donc de vérifier la valeur qui s'apprête à être insérée dans la table Propriétaire.

- Dans un premier temps, vérifie que le groupe de musique identifié est bien une playlist, et non un album (un album ne pouvant pas posséder de propriétaires). S'il s'agit d'un album, alors on lève une erreur, associant un code et un message correspondant, l'insertion est donc interrompue.
- S'il s'agit d'une playlist, alors il est nécessaire de vérifier que celle-ci n'est pas déjà associée à un autre utilisateur. Si c'est le cas, alors, de la même manière, on interrompt l'insertion, une playlist ne pouvant posséder qu'un seul propriétaire.

CHK_ECOUTEGM_VALIDE :

Règles de gestion :

- Une playlist privée ne peut être écoutée que par son propriétaire.
- Une playlist vide ne peut être écoutée
- L'heure d'écoute n'est valide que si elle est comprise entre la date de création du groupe de musique et l'heure actuelle.

```
CREATE OR REPLACE TRIGGER CHK_ECOUTEGM_VALIDE
BEFORE INSERT OR UPDATE ON EcouteGM
FOR EACH ROW
DECLARE
    Prop int;
    EstPr char(1);
    DateCr timestamp;
    NbMusiques int;
BEGIN
    SELECT GM.EstPrive, TO_TIMESTAMP(TO_CHAR(GM.DateCreation, 'DD-Mon-YYYY HH24:MI:SS'), 'DD-Mon-YYYY HH24:MI:SS') INTO EstPr, DateCr
    FROM GroupeMusique GM
    WHERE GM.IdGroupeMusique = :New.IdGroupeMusique;

    SELECT P.IdUtilisateur INTO Prop
    FROM Proprietaire P
    WHERE P.IdGroupeMusique = :New.IdGroupeMusique;

    SELECT COUNT(*) INTO NbMusiques
    FROM Compose C
    WHERE C.IdGroupeMusique = :New.IdGroupeMusique;

    IF EstPr = 'Y' AND :New.IdUtilisateur <> Prop THEN
        RAISE_APPLICATION_ERROR(-20003, 'La playlist numero ' || :New.IdGroupeMusique || ' est privee, seul son proprietaire peut l''ecouter.');

```

Figure 22 : Trigger sur la table EcouteGM

Pour vérifier la validité d'un tuple de EcouteGM, on effectue donc les vérifications suivantes :

- S'il s'agit d'une playlist privée, on vérifie que l'utilisateur l'ayant écouté est son propriétaire, sinon on lève une erreur.
- Si le groupe de musiques est vide, alors l'utilisateur n'a pas pu l'écouter, on n'insérera donc aucune ligne dans EcouteGM, on lève donc une erreur.
- Enfin, si la date d'écoute est invalide (non comprise entre la date de création de la playlist et la date actuelle), ici encore on lèvera une erreur.

CHK_ECOUTEMUSIQUE VALIDE :

Règles de gestion :

- L'heure d'écoute n'est valide que si elle est comprise entre la date de création du groupe de musique et l'heure actuelle.

```
CREATE OR REPLACE TRIGGER CHK_ECOUTEMUSIQUE_VALIDE
BEFORE INSERT OR UPDATE ON EcouteMusique
FOR EACH ROW
DECLARE
    DateCr timestamp;
BEGIN
    SELECT TO_TIMESTAMP(TO_CHAR(M.DatePublication, 'DD-Mon-YYYY HH24:MI:SS'), 'DD-Mon-YYYY HH24:MI:SS') INTO DateCr
    FROM Musique M
    WHERE M.IdMusique = :New.IdMusique;

    IF :New.HeureEcouté > SYSTIMESTAMP OR :New.HeureEcouté < DateCr THEN
        RAISE_APPLICATION_ERROR(-20004, 'Heure d'écoute invalide');
    END IF;

    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL;
END;
/
```

Figure 23 : Trigger sur EcouteMusique

CHK_USER VALIDE :

```
CREATE OR REPLACE TRIGGER CHK_USER_VALIDE
BEFORE INSERT OR UPDATE ON Utilisateur
FOR EACH ROW
BEGIN
    IF :New.DateNaissance > SYSDATE OR :New.DateNaissance < TO_DATE('01-jan-1900') THEN
        RAISE_APPLICATION_ERROR(-2005, 'Date de naissance incorrecte');
    END IF;

    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL;
END;
/
```

Figure 24 : Trigger sur Utilisateur

On doit également procéder de la même manière pour la table Utilisateur, en vérifiant sa date de naissance.

CHK_AIME_VALIDE (clés étrangères) :

```

CREATE OR REPLACE TRIGGER CHK_AIME_VALIDE
BEFORE INSERT OR UPDATE ON Aime
FOR EACH ROW
DECLARE
fk1 int := 0;
fk2 int := 0;
pk int := 0;
BEGIN
    -- Vérif existence de la 1ère clé étrangère
    SELECT COUNT(SM.IdStyleMusique) INTO fk1
    FROM StyleMusique SM
    WHERE SM.IdStyleMusique = :New.IdStyleMusique;

    -- Vérif existence de la 2ème clé étrangère
    SELECT COUNT(U.IdUtilisateur) INTO fk2
    FROM Utilisateur U
    WHERE U.IdUtilisateur = :New.IdUtilisateur;

    -- Vérif unicité de la clé primaire
    SELECT COUNT(*) INTO pk
    FROM Aime A
    WHERE A.IdUtilisateur = :New.IdUtilisateur AND A.IdStyleMusique = :New.IdStyleMusique;

    IF pk > 0 THEN
        RAISE_APPLICATION_ERROR(-2007, 'Contrainte d''unicite de cle primaire violee');
    ELSIF fk1 = 0 THEN
        RAISE_APPLICATION_ERROR(-2006, 'Contrainte d''integrite violee, cle parente introuvable');
    ELSIF fk2 = 0 THEN
        RAISE_APPLICATION_ERROR(-2006, 'Contrainte d''integrite violee, cle parente introuvable');
    END IF;

    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL;
END;
/

```

Figure 25 : Trigger sur Aime

Comme demandé dans l'énoncé, nous avons tenté de reproduire le travail effectué par Oracle lors de la validation d'une clé étrangère.

Notre association Aime étant composée de deux clés étrangères (formant sa clé primaire), il y a trois vérifications à effectuer :

- La présence de la première clé étrangère dans la première table maître
- La présence de la seconde clé étrangère dans la deuxième table maître
- L'unicité de la clé primaire, c'est à dire de la composition des deux clés étrangères.

Si l'une de ces trois contraintes n'est pas vérifiée, alors l'insertion dans la table association n'est pas possible, on lève donc une erreur l'annulant.

3. Suppression des données

Nous avons également mis en place des triggers permettant de supprimer nos données de manière cohérente dans notre base de données.

En effet, si l'on supprime un artiste de la base, il est nécessaire de supprimer des autres tables toutes ses références directes et indirectes, et ce sans violer de contraintes de clé étrangères.

Nous avons donc défini les triggers suivants :

DL MUSIQUE :

Règles de gestion :

- Lorsqu'une musique est supprimée de la table, les données d'autres tables lui faisant références sont, elles aussi supprimées.
- Un album vide est supprimé.

Ce trigger s'exécute avant la suppression d'un tuple de la table Musique.

La musique étant référencée dans les tables associations Produit, Compose, Appartient et EcouteMusique, on commence donc par supprimer les lignes correspondantes.

Dans le cas où la Musique s'apprêtant à être supprimée faisait partie d'un album, et que cet album est devenu vide, alors on le supprime.

```
CREATE OR REPLACE TRIGGER DL_MUSIQUE
BEFORE DELETE ON Musique
FOR EACH ROW
BEGIN
    DELETE FROM Produit WHERE IdMusique = :Old.IdMusique;
    DELETE FROM Compose WHERE IdMusique = :Old.IdMusique;
    DELETE FROM Appartient WHERE IdMusique = :Old.IdMusique;
    DELETE FROM EcouteMusique WHERE IdMusique = :Old.IdMusique;

    DELETE FROM GroupeMusique GM
    WHERE GM.IdGroupeMusique NOT IN (
        SELECT IdGroupeMusique
        FROM Compose C
        WHERE C.IdGroupeMusique = GM.IdGroupeMusique AND C.IdMusique = :Old.IdMusique)
    AND GM.EstPlaylist = 'N';
END;
/
```

Figure 26 : Trigger s'exécutant avant la suppression d'un tuple de Musique

DL GM :

Afin de supprimer correctement un groupe de musique, il faut simplement supprimer les lignes correspondantes des tables associations. Cela revient à définir la clause 'ON DELETE CASCADE' sur les contraintes de clés étrangères faisant référence à GroupeMusique.

```
CREATE OR REPLACE TRIGGER DL_GM
BEFORE DELETE ON GroupeMusique
FOR EACH ROW
BEGIN
    DELETE FROM Proprietaire WHERE IdGroupeMusique = :Old.IdGroupeMusique;
    DELETE FROM EcouleGM WHERE IdGroupeMusique = :Old.IdGroupeMusique;
    DELETE FROM Compose WHERE IdGroupeMusique = :Old.IdGroupeMusique;
END;
/
```

Figure 27 : Trigger s'exécutant avant la suppression d'un tuple de GroupeMusique

DL SM :

```
CREATE OR REPLACE TRIGGER DL_SM
BEFORE DELETE ON StyleMusique
FOR EACH ROW
BEGIN
    DELETE FROM Aime WHERE IdStyleMusique = :Old.IdStyleMusique;
    DELETE FROM Appartient WHERE IdStyleMusique = :Old.IdStyleMusique;
END;
/
```

Figure 28 : Trigger s'exécutant avant la suppression d'un tuple de StyleMusique

DL ARTISTE :

Règles de gestion :

- Lorsqu'un artiste est supprimé, s'il en est le seul producteur, ses musiques sont supprimées
- Lorsqu'un artiste est supprimé, s'il n'en est pas l'unique producteurs, ses musiques sont conservées, mais il n'est plus mentionné en tant que producteur de cette musique.

Pour effectuer ce traitement, nous avons choisi de mettre en place un curseur, parcourant les différents identifiants des musiques produites par l'artiste

```
CREATE OR REPLACE TRIGGER DL_ARTISTE
BEFORE DELETE ON Artiste
FOR EACH ROW
DECLARE
    nbArtistes int;
    CURSOR MusiquesASupprimer IS SELECT IdMusique FROM Produit WHERE IdArtiste = :Old.IdArtiste;
    m_IdMusique int;
BEGIN
    OPEN MusiquesASupprimer;
    LOOP
        FETCH MusiquesASupprimer INTO m_IdMusique;
        EXIT WHEN MusiquesASupprimer%NOTFOUND;

        SELECT COUNT(IdArtiste) INTO nbArtistes
        FROM Produit P
        WHERE P.IdMusique = m_IdMusique;

        IF nbArtistes = 1 THEN
            DELETE FROM Musique WHERE IdMusique = m_IdMusique;
        ELSE
            DELETE FROM Produit WHERE IdMusique = m_IdMusique;
        END IF;
    END LOOP;
    CLOSE MusiquesASupprimer;
END;
/
```

Figure 29 : Trigger s'exécutant avant la suppression d'un tuple de la table Artiste

DL_USER :

Règles de gestion :

- Lorsqu'un utilisateur est supprimé, toutes les playlists dont il était propriétaire sont également supprimées
- Lorsqu'un utilisateur est supprimé, toutes les informations qui lui sont relatives sont elles-aussi supprimées.

```
CREATE OR REPLACE TRIGGER DL_USER
BEFORE DELETE ON Utilisateur
FOR EACH ROW
BEGIN
    DELETE FROM GroupeMusique
    WHERE IdGroupeMusique IN (
        SELECT IdGroupeMusique
        FROM Proprietaire P
        WHERE P.IdUtilisateur = :Old.IdUtilisateur);

    DELETE FROM EcouteMusique WHERE IdUtilisateur = :Old.IdUtilisateur;
    DELETE FROM EcouteGM WHERE IdUtilisateur = :Old.IdUtilisateur;
    DELETE FROM Aime WHERE IdUtilisateur = :Old.IdUtilisateur;
    DELETE FROM Proprietaire WHERE IdUtilisateur = :Old.IdUtilisateur;
END;
/
```

Figure 30 : Trigger s'exécutant avant la suppression d'un tuple de la table Utilisateur

D. Jeu de données

Pour la mise en place du jeu de données, nous avons souhaité traiter le maximum de cas de figures, en le diversifiant du mieux possible.

Le jeu de données étant de forte taille, nous n'allons pas détailler le contenu de nos tables, mais expliquer comment nous l'avons généré, pour chaque table :

- **Utilisateur** : Utilisateurs provenant de différents pays, de sexes féminin et masculin, et ayant souscrit à l'abonnement premium ou non.
- **Artiste** : Artistes de nationalités différentes.
- **Musique** : Différentes musiques, de différents styles et artistes, ayant été réalisées à des dates différentes et par des artistes différents.
- **GroupeMusique** : Des albums (sans utilisateur propriétaire), ainsi que des playlists publiques et privées.
- **StyleMusique** : Styles musicaux divers et variés .
- **EcouteMusique** : Des utilisateurs (premium ou non) écoutant des musiques à des instants donnés, et d'autres utilisateur n'ayant pas écouté de musique.
- **EcouteGM** : Des utilisateurs écoutant la playlist (privée ou publique) dont ils sont propriétaires, des utilisateurs écoutant les playlists publiques (ou albums), une ou plusieurs fois, à des instants donnés, et d'autres utilisateurs n'ayant pas écouté de groupe de musique.
- **Propriétaire** : Certains utilisateurs sont propriétaires d'une ou plusieurs playlists, d'autres n'en possèdent aucune.
- **Aime** : Certains utilisateurs aiment un ou plusieurs style de musique, d'autres n'en aiment pas.
- **Appartient** : Une musique appartient à un ou plusieurs styles.
- **Compose** : Un groupe de musique peut être composé de 0 ou plusieurs musiques, nous avons donc généré des groupes de musiques vides (composé d'aucune musique), d'autres étant composés d'une ou plusieurs musiques.
- **Produit** : Un artiste produit 0 ou plusieurs musiques, et plusieurs artistes peuvent être producteurs d'une seule chanson.
Nous avons donc mis en place des musiques associées à un seul artiste, d'autres étant associées à plusieurs artistes.
Certains artistes ne sont pas associés à une musique.

Comme demandé dans le sujet, nous avons mis en place trois méthodes différentes pour remplir nos tables, que nous allons détailler ci-après :

1. Insert Into

La première méthode consistait à insérer les tuples dans nos tables via la commande de SQL INSERT INTO.

On notera que les identifiants ne sont pas saisis, puisque générés automatiquement par le trigger (cf. B.)

```
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Pink Floyd','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Led Zeppelin','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Telephone','FR');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Rolling Stones','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('AC/DC','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('The Beatles','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Scorpions','DE');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('U2','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Bob Dylan','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Johnny Halliday','FR');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Bob Marley','JM');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Orelsan','FR');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Drake','CA');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('JUL','FR');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Jain','CA');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Beyonce','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Rihanna','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Eminem','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Sia','AU');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Ariana Grande','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Drake','CA');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Miles Davis','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Ray Charles','GE');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Damso','BE');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Hans Zimmer','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Gorillaz','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Iron Maiden','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('ZZ Top','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Black Sabbath','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('2Pac','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Amy Winehouse','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Avicii','SE');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Psy','KR');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Queen','GB');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Michael Jackson','US');
INSERT INTO Artiste (NomScene,Nationalite) VALUES('Celine Dion','CA');
```

Figure 31 : Insertion de tuples dans la table Artiste par SQL

L'inconvénient de cette méthode est qu'elle est assez redondante : on doit en effet recopier pour chaque tuple la ligne INSERT INTO table VALUES (...).

Nous allons donc traiter une autre méthode, l'insertion par SQL*Loader.

2. SQL*Loader

SQL*Loader est un programme permettant d'insérer des tuples dans une table depuis le terminal.

Le fichier doit avoir une forme semblable à celui ci-dessous :

```
LOAD DATA
INFILE *
APPEND
INTO TABLE Utilisateur
FIELDS TERMINATED BY ','
(Pseudo,Nom,Prenom,Email,Nationalite,DateNaissance,Sexe,EstPremium)
BEGINDATA
Clem27,Lefevre,Clement,lefevreclement27@gmail.com,FR,21-apr-1998,M,Y
BetiGrad,Gradasso,Bertrand,gradassoBertrand@jourrapide.com,FR,29-sep-1980,M,Y
DanaVan,Vanmatre,Dana,DanaSVanmatre@dayrep.com,US,24-jan-1986,F,N
Xx_Duvi_xX,Duval,Vick,vickduval@yahoo.fr,FR,21-sep-1973,F,N
LeiLian,Lian,Lei,LianLei@teleworm.us,CN,17-aug-1971,M,Y
StuijRiaz123,Riaz,Stuij,RiazStuij@jourrapide.com,NL,28-mar-2000,F,N
Claudiaaa_Nus,Nussbaum,Claudia,ClaudiaNussbaum@rhyta.com,US,21-oct-1961,F,Y
Dark0sb0rne,Osborne,ElIiot,ElIiotOsborne@teleworm.us,US,9-jan-2001,M,N
TheLewisBenson,Benson,Lewis,LewisBenson@dayrep.com,AR,13-aug-1989,M,Y
BadGal58,Storey,Emma,EmmaStorey@armyspy.com,BR,20-sep-1958,F,Y
GorshkovGeorg,Gorshkov,George,GeorgeGorshkov@armyspy.com,RU,02-jun-1984,M,N
POliveira,Oliveira,Patricia,olivp@wanadoo.fr,PT,23-apr-1999,F,Y
ItalianoPizzaPasta,Udinese,Abelino,AbelinoUdinese@jourrapide.com,IT,6-feb-1973,M,Y
SnowyWhite,Henry,White,HenrySWhite@teleworm.us,US,31-aug-1962,M,N
WinterIsComing,Tomlinson,Julie,JulieDTomlinson@armyspy.com,GB,11-dec-1964,F,Y
Thanaa,Malouf,Usaymah,UsaymahThanaaMalouf@teleworm.us,ML,9-dec-1992,F,N
ShingTseng,Tseng,Shing,ShingTseng@rhyta.com,CN,5-dec-1972,M,Y
Jak0b,Andersen,Jakob,JakobAndersen@armyspy.com,NO,6-jan-1963,M,Y
Forld1997,Petersen,Gustav,GustavNPetersen@armyspy.com,DK,22-dec-1997,M,N
Thionus,Lamontagne,Alain,AlainLamontagne@dayrep.com,FR,27-feb-1953,M,N
Wastione,Bordeleau,Yvette,YvetteBordeleau@armyspy.com,FR,19-nov-1994,F,Y
Gazinsibelf45,Santacruz,Marina,MarinaSantacruzGuajardo@dayrep.com,ES,01-feb-2000,F,N
Ancralows,Vigil Hinojosa,Laurentino,LaurentinoVigilHinojosa@dayrep.com,MX,03-mar-1996,M,N
Thistried,Crawford,Alexander,AlexanderCrawford@armyspy.com,GB,14-dec-1998,M,Y
Thintich,Kaiser,Franziska,FranziskaKaiser@dayrep.com,DE,24-mar-1970,F,Y
Qual1996,Silva Martins,Murilo,MuriloSilvaMartins@armyspy.com,BR,28-jun-1996,M,N
Apere1960,Jakobsen,Jonathan,JonathanJakobsen@armyspy.com,SE,15-may-1960,M,Y
Dession1951,Kucharska,Kasia,KasiaKucharska@rhyta.com,PL,10-mar-1961,F,Y
Tores1982,Zito,Artemia,ArtemiaZito@teleworm.us,IT,8-sep-1972,F,Y
Whimints1954,Hixson,Harry,HarryHixson@teleworm.us,AU,22-apr-1954,M,N
```

Figure 32 : Insertion de tuples dans la table Utilisateur par SQL Loader

Cette méthode est plus avantageuse que la première pour l'insertion de grandes quantités de données.

De plus, même si nous ne l'avons pas effectué dans ce projet, il est possible d'insérer les données depuis un fichier au format csv.

3. Script PL/SQL

```

DECLARE
  TYPE TYP_SM IS TABLE OF STYLEMUSIQUE.NOMSTYLE%TYPE;
  NOM_STYLE_MUS TYP_SM;
  i int;
BEGIN
  NOM_STYLE_MUS := TYP_SM ('Rock','Pop','Inde','Reggae','Rap','RnB','Soul','Instrumental','Blues','Metal','Hard Rock','Folk','Funk',
    'Disco','Country','Techno','EDM','Electro','Classique','House','Gospel','Ambient','New Wave','Psychedelique','World','Hip-Hop',
    'Salsa','Jazz','Opera','Zouk','Dance','Variete','Film');

  i := NOM_STYLE_MUS.FIRST;

  WHILE i IS NOT NULL LOOP
    INSERT INTO StyleMusique(NomStyle) VALUES (NOM_STYLE_MUS(i));
    i := NOM_STYLE_MUS.NEXT(i);
  END LOOP;

END;
/

```

Figure 33 : Script PL/SQL d'insertion de données dans la table StyleMusique

Pour insérer nos données depuis un script PL/SQL, on saisit dans un premier temps nos données sous la forme d'un tableau.

Suite à cela, on boucle jusqu'au dernier élément de ce tableau, et à chaque itération on insère dans la table StyleMusique la valeur correspondante.

Ici encore, notre trigger d'auto-incrément nous a facilité la tâche, puisque nous n'avons pas à nous soucier de l'identifiant à chaque insertion.

4. Autres

Pour certaines tables nécessitant un nombre important de tuples pour être suffisamment intéressante, telles que EcouteMusique et EcouteGM, nous avons implémenté un programme en C qui nous a permis de générer des données.

Les données ont donc été générées (pseudo) aléatoirement, pour ces tables.

Manipulation des données

A. Vues

Une vue peut être définie comme une table virtuelle, définie par une requête, qui offre une manière alternative d'accéder aux données.

Nous avons décidé de détailler les vues avant les requêtes effectuées, puisque certaines de nos requêtes les utilisent.

Nous allons détailler ci-après les différentes vues implémentées dans le cadre du projet, ainsi que les droits d'accès associés.

Vue sur toutes les musiques écoutées :

Cette vue nous permet d'obtenir la liste complètes des musiques écoutées, en passant par la table Musique, ou la table GroupeMusique.

Pour ce faire, on effectue l'union (en conservant les doublons, d'où le ALL) de deux requêtes :

- L'ensemble des musiques écoutées
- L'ensemble des musiques composants les groupes de musiques écoutés.

Cette vue est donc un historique des écoutes de musiques ayant eu lieu, mais il s'agit d'une vue au sens pratique du terme, puisque la requête suivante est utilisée pour de nombreuses requêtes, en faire une vue simplifie donc l'écriture des autres requêtes.

```
CREATE VIEW MusiqueEcoutees
(IdMusique,Titre,IdUtilisateur,HeureEcoule) AS
SELECT M.IdMusique, M.Titre, EM.IdUtilisateur, EM.HeureEcoule
FROM EcouteMusique EM, Musique M
WHERE EM.IdMusique = M.IdMusique
UNION ALL
SELECT M.IdMusique, M.Titre, EGM.IdUtilisateur, EGM.HeureEcoule
FROM EcouteGM EGM, Compose C, Musique M
WHERE EGM.IdGroupeMusique = C.IdGroupeMusique AND C.IdMusique = M.IdMusique;
```

Figure 34 : Vue MusiqueEcoutees

Droits d'accès :

- Les administrateurs, gérants de l'application ont accès à l'historique en totalité, pour tous les utilisateurs.
- Les utilisateurs ont accès à leur historique (il suffit de rajouter une condition sur l'identifiant)

Vue sur tous les utilisateurs :

Cette vue nous permet d'afficher les informations non-sensibles (privées) de nos utilisateurs, tels que leur adresse E-Mail par exemple.

On pourrait imaginer un système (restreint) de profil, où chaque utilisateur a accès à ces données.

```
CREATE VIEW V_Utilisateurs
(IdUtilisateur, Pseudo, Age, Nationalite) AS
SELECT IdUtilisateur, Pseudo, TRUNC((SYSDATE - DateNaissance) / 365.25) AS AGE, Nationalite
FROM Utilisateur, DUAL;
```

Figure 35 : Vue V_Utilisateurs

Droits d'accès : Tous les utilisateurs ont accès à cette vue

Vue sur les musiques, leurs artistes et leurs styles :

Cette vue permet de recenser toutes les musiques, ainsi que les artistes associés, et les éventuels styles de musiques.

On remarquera l'utilisation d'une jointure externe, une musique ne pouvant appartenir à aucun style.

```
CREATE VIEW ListeMusiques
(IdArtiste, NomArtiste, IdMusique, TitreMusique, NomStyle) AS
SELECT A.IdArtiste, A.NomScene, M.IdMusique, M.Titre, SM.NomStyle
FROM Musique M
INNER JOIN Produit P ON M.IdMusique = P.IdMusique
INNER JOIN Artiste A ON P.IdArtiste = A.IdArtiste
LEFT OUTER JOIN Appartient AP ON AP.IdMusique = M.IdMusique
LEFT OUTER JOIN StyleMusique SM ON AP.IdStyleMusique = SM.IdStyleMusique
ORDER BY IdArtiste, IdMusique;
```

Figure 36 : Vue ListeMusiques

Droits d'accès : Tous les utilisateurs ont accès à cette vue.

Vue sur les groupes de musiques visibles :

Cette vue permet de recenser les groupes de musiques accessibles aux utilisateurs. Un utilisateur a accès aux groupes de musiques publics, et aux groupes de musiques privés dont il est le propriétaire, on fait donc l'union de ces deux requêtes.

```
CREATE VIEW GMVisibles
(IdUtilisateur, IdGroupeMusique, NomGroupeMusique) AS
SELECT U.IdUtilisateur, GM.Idgroupe musique, GM.Titre
FROM Utilisateur U, GroupeMusique GM, Proprietaire P
WHERE GM.EstPrive = 'Y' AND U.IdUtilisateur = P.IdUtilisateur
AND P.IdGroupeMusique = GM.IdGroupeMusique
UNION
SELECT U.IdUtilisateur, GM.IdGroupeMusique, GM.Titre
FROM Utilisateur U, GroupeMusique GM
WHERE GM.EstPrive = 'N';
```

Figure 37 : Vue GMVisibles

Droits d'accès :

- Chaque utilisateur a accès à sa propre vue (les groupes de musiques publics et ses éventuelles playlists privées).
- L'administrateur a accès à l'ensemble de la vue

Vue sur le nombre d'écoutes de chaque artiste:

Cette vue permet de compter le nombre d'écoutes de chaque artiste. On utilise la jointure externe afin d'afficher les artistes n'ayant aucune écoute.

```
CREATE VIEW EcouleArtiste
(IdArtiste, NomArtiste, Nombre d'écoutes) AS
SELECT A.IdArtiste, A.NomScene, COUNT(ME.IdMusique)
FROM Artiste A
LEFT OUTER JOIN Produit P ON A.IdArtiste = P.IdArtiste
LEFT OUTER JOIN MusiqueEcoutees ME ON P.IdMusique = ME.IdMusique
GROUP BY (A.IdArtiste, A.NomScene);
```

Figure 38 : Vue EcouleArtiste

Droits d'accès :

- Chaque artiste a accès à sa propre vue personnalisée
- Les utilisateurs n'ont pas accès à cette vue
- L'administrateur a accès à cette vue dans sa totalité

Vue sur toutes les groupes de musiques et leurs composition :

Cette vue permet d'afficher la composition de chaque groupe de musique public y-compris ceux étant vides (grâce à la jointure externe).

```
CREATE VIEW ListeGM
(IdGroupeMusique, TitreGroupeMusique, IdMusique, Titre, Duree) AS
SELECT GM.IdGroupeMusique, GM.Titre, M.IdMusique, M.Titre, M.Duree
FROM GroupeMusique GM
LEFT OUTER JOIN Compose C ON GM.IdGroupeMusique = C.IdGroupeMusique
LEFT OUTER JOIN Musique M ON C.IdMusique = M.IdMusique
WHERE GM.EstPrive = 'N'
ORDER BY IdGroupeMusique;
```

Figure 39 : Vue ListeGM

Droits d'accès : Tous les utilisateurs ont accès à cette vue, qui recense uniquement les informations relatives aux groupes de musiques publics.

Vue sur les pseudos des utilisateurs et les styles qu'ils aiment :

Cette vue recense les différents utilisateurs ainsi que leurs styles favoris. S'ils n'ont mentionné aucun styles de musique dans leurs préférences, on affiche uniquement le pseudo de l'utilisateur (grâce à la jointure externe).

```
CREATE VIEW UtilisateursStyles
(IdUtilisateur, Pseudo, IdStyleMusique, Styles_Aimes) AS
SELECT U.IdUtilisateur, U.Pseudo, SM.IdStyleMusique, SM.NomStyle
FROM Utilisateur U
LEFT OUTER JOIN Aime A ON U.IdUtilisateur = A.IdUtilisateur
LEFT OUTER JOIN StyleMusique SM ON A.IdStyleMusique = SM.IdStyleMusique
ORDER BY IdUtilisateur;
```

Figure 40 : Vue UtilisateursStyles

Droits d'accès : Tous les utilisateurs ont accès à cette vue, et donc aux préférences de tous les utilisateurs.

B. Requêtes

Lors de la création de la base de données, nous nous sommes posé les questions suivantes, au langage naturel, que nous allons traduire en requête SQL.

1. Liste de tous les utilisateurs triée par ordre alphabétique (de nom de famille) :

```
SELECT *  
FROM Utilisateur U  
ORDER BY U.Nom
```

2. Rendre l'utilisateur 3 Premium :

```
UPDATE Utilisateur  
SET EstPremium = 'Y'  
WHERE IdUtilisateur = 3;
```

3. Suppression de la chanson 3 de la Playlist 4

```
DELETE FROM Compose  
WHERE IdGroupeMusique = 3  
AND IdMusique = 4;
```

4. Liste des utilisatrices (F) propriétaires d'au moins une playlist :

```
SELECT U.*  
FROM Utilisateur U, Proprietaire P, GroupeMusique GM  
WHERE U.Sexe = 'F' AND U.IdUtilisateur = P.IdUtilisateur  
AND P.IdGroupeMusique = GM.IdGroupeMusique;
```

5. Pseudos et dates de naissances des utilisateurs aimant le rock :

```
SELECT U.IdUtilisateur, U.Pseudo, U.DateNaissance  
FROM Utilisateur U, Aime A, StyleMusique SM  
WHERE U.IdUtilisateur = A.IdUtilisateur  
AND A.IdStyleMusique = SM.IdStyleMusique  
AND SM.NomStyle = 'Rock';
```

6. Suppression des playlists de plus de 10 ans :

```
DELETE  
FROM GroupeMusique GM  
WHERE GM.EstPlaylist = 'Y' AND (SYSDATE - GM.DateCreation) > 365 * 10
```

7. Calcul de l'âge moyen des utilisateurs :

```
SELECT AVG(Age) AS Age_Moyen
FROM (
  SELECT MONTHS_BETWEEN (TRUNC(SYSDATE),U.DateNaissance) / 12 AS Age
  FROM Utilisateur U, Dual);
```

8. Liste des artistes ayant déjà effectué une collaboration pour une musique :

```
SELECT A1.IdArtiste, A1.NomScene
FROM Artiste A1, Produit P1
WHERE A1.IdArtiste = P1.IdArtiste AND EXISTS (
  SELECT P2.IdArtiste
  FROM Produit P2
  WHERE P1.IdMusique = P2.IdMusique AND P1.IdArtiste <> P2.IdArtiste);
```

9. Liste des utilisateurs n'aimant pas le rock mais en ayant écouté au moins une fois :

On prend l'intersection des utilisateurs n'aimant pas le rock les utilisateurs en ayant déjà écouté au moins une fois.

```
SELECT U.*
FROM Utilisateur U
WHERE U.IdUtilisateur NOT IN (
  SELECT IdUtilisateur
  FROM Aime A, StyleMusique SM
  WHERE A.IdStyleMusique = SM.IdStyleMusique AND SM.NomStyle = 'Rock')
INTERSECT
(
  SELECT U.*
  FROM Utilisateur U, EcouteMusique EM, Appartient A, StyleMusique SM
  WHERE U.IdUtilisateur = EM.IdUtilisateur
  AND EM.IdMusique = A.IdMusique
  AND A.IdStyleMusique = SM.IdStyleMusique
  AND SM.NomStyle = 'Rock'
  UNION ALL
  SELECT U.*
  FROM Utilisateur U, EcouteGM EGM, Compose C, Appartient A, StyleMusique SM
  WHERE U.IdUtilisateur = EGM.IdUtilisateur
  AND EGM.IdGroupeMusique = C.IdGroupeMusique
  AND C.IdMusique = A.IdMusique
  AND A.IdStyleMusique = SM.IdStyleMusique
  AND SM.NomStyle = 'Rock');
```

On peut néanmoins simplifier l'écriture de cette requête grâce à la vue *MusiqueEcoutees* précédemment définie :

```

SELECT U.*
FROM Utilisateur U
WHERE U.IdUtilisateur NOT IN (
    SELECT IdUtilisateur
    FROM Aime A, StyleMusique SM
    WHERE A.IdStyleMusique = SM.IdStyleMusique AND SM.NomStyle = 'Rock')
INTERSECT
(
    SELECT U.*
    FROM Utilisateur U, MusiqueEcoutees EM, Appartient A, StyleMusique SM
    WHERE U.IdUtilisateur = EM.IdUtilisateur
    AND EM.IdMusique = A.IdMusique
    AND A.IdStyleMusique = SM.IdStyleMusique
    AND SM.NomStyle = 'Rock');

```

10. La chanson la plus écoutée par nationalité (et son nombre d'écoutes associé)

```

WITH T AS (
    SELECT U.Nationalite, E.IdMusique, E.Titre, COUNT(E.IdMusique) AS Compte, ROW_NUMBER() OVER (
        PARTITION BY U.Nationalite
        ORDER BY U.Nationalite ASC) AS RN
    FROM MusiqueEcoutees E
    INNER JOIN Utilisateur U ON E.IdUtilisateur = U.IdUtilisateur
    GROUP BY E.IdMusique, E.Titre, U.Nationalite
    ORDER BY U.Nationalite, Compte DESC
)
SELECT Nationalite, Idmusique, Titre, Compte
FROM T
WHERE RN = 1;

```

Grâce à cette syntaxe, on peut récupérer la plus grande valeur de chaque groupe issu du 'GROUP BY'.

Dans notre cas, on regroupe les chansons selon la nationalité, en les triant, au sein de chaque groupe, par ordre décroissant du nombre d'écoutes.

La chanson la plus écoutée par nationalité est donc la première ligne de chaque groupe.

Il est possible de s'affranchir de la notation WITH..AS, en utilisant la syntaxe suivante :

```

SELECT Nationalite, Idmusique, Titre, Compte
FROM (
    SELECT U.Nationalite, E.IdMusique, E.Titre, COUNT(E.IdMusique) AS Compte,
    ROW_NUMBER() OVER (PARTITION BY U.Nationalite
        ORDER BY U.Nationalite ASC) AS RN
    FROM MusiqueEcoutees E
    INNER JOIN Utilisateur U ON E.IdUtilisateur = U.IdUtilisateur
    GROUP BY E.IdMusique, E.Titre, U.Nationalite
    ORDER BY U.Nationalite, Compte DESC)
WHERE RN = 1;

```

11. Les trois artistes favoris de chaque utilisateurs (et le temps d'écoute associé)

```
WITH TotalUtilisateurParArtiste AS (
  SELECT U.IdUtilisateur, U.Pseudo, A.IdArtiste, A.NomScene, SUM(M.Duree) AS TempsEcoute,
  ROW_NUMBER() OVER (PARTITION BY U.IdUtilisateur ORDER BY SUM(M.Duree) DESC) AS RN
  FROM Musique M
  INNER JOIN MusiqueEcoutees E ON M.IdMusique = E.IdMusique
  INNER JOIN Utilisateur U ON E.IdUtilisateur = U.IdUtilisateur
  INNER JOIN Produit P ON M.IdMusique = P.IdMusique
  INNER JOIN Artiste A ON A.IdArtiste = P.IdArtiste
  GROUP BY A.IdArtiste, A.NomScene, U.IdUtilisateur, U.Pseudo)
SELECT IdUtilisateur, Pseudo, IdArtiste, NomScene, TempsEcoute
FROM TotalUtilisateurParArtiste
WHERE RN <= 3;
```

12. Titres recommandés aux utilisateurs selon leurs goûts (avec facteur de recommandation)

Le facteur de recommandation mentionné ici n'est autre que le nombre de similitudes entre les styles aimés par l'utilisateur, et les styles associés à la musique. Par exemple, si l'utilisateur aime le Rock et la Pop, et qu'une musique appartient au style Pop et au style Rock, alors le facteur de recommandation vaudra 2, puisque la musique sera plus susceptible de plaire à l'utilisateur.

```
SELECT U.IdUtilisateur, U.Pseudo, M.IdMusique, M.Titre, COUNT(M.IdMusique) AS FacteurRecommandation
FROM Musique M
INNER JOIN Appartient AP ON M.IdMusique = AP.IdMusique
INNER JOIN Aime AI ON AP.IdStyleMusique = AI.IdStyleMusique
INNER JOIN Utilisateur U ON AI.IdUtilisateur = U.IdUtilisateur
GROUP BY U.IdUtilisateur, U.Pseudo, M.IdMusique, M.Titre
ORDER BY U.IdUtilisateur, FacteurRecommandation DESC;
```

13. Nombre d'écoutes par style de musique :

On effectue ici la jointure entre deux vues, la première permettant de récupérer toutes les écoutes ayant eu lieu, et la seconde recensant toutes les musiques et les styles associés.

En effectuant le GROUP BY IdMusique, NomStyle, on supprime les doublons (puisque une même chanson peut être produite par plusieurs artistes).

```
SELECT NomStyle, COUNT(ME.IdMusique) AS NombreEcoutes
FROM MusiqueEcoutees ME
LEFT OUTER JOIN (
  SELECT IdMusique, NomStyle
  FROM ListeMusiques
  GROUP BY IdMusique, NomStyle) LM
ON LM.IdMusique = ME.IdMusique
GROUP BY NomStyle;
```

14. Liste des chansons qui apparaissent sur plus de deux playlists différentes :

```
SELECT M.Titre, COUNT(DISTINCT GM.IdGroupeMusique)
FROM GroupeMusique GM, Compose C, Musique M
WHERE GM.EstPlaylist = 'Y'
AND GM.IdGroupeMusique = C.IdGroupeMusique
AND C.IdMusique = M.IdMusique
GROUP BY M.Titre
HAVING COUNT(DISTINCT GM.IdGroupeMusique) > 2;
```

15. Liste des dix chansons les plus écoutées dans le monde :

```
SELECT M.IdMusique, M.Titre, NB.NbEcoutés
FROM Musique M, (
    SELECT ME.IdMusique, COUNT(ME.IdMusique) AS NbEcoutés
    FROM MusiqueEcoutées ME
    GROUP BY ME.IdMusique
    ORDER BY NbEcoutés DESC; ) NB
WHERE ROWNUM <= 10 AND M.IdMusique = NB.IdMusique;
```

16. Les cinq musiques rock les plus écoutées en 2017 :

Les colonnes HeureEcouté des tables EcoutéMusique et EcoutéGroupeMusique étant des types TIMESTAMP, il est important de convertir notre entrée dans ce type, à l'aide de la fonction TO_TIMESTAMP, en précisant le masque désiré.

```
SELECT IdMusique, Titre, NbEcoutés
FROM (
    SELECT ME.IdMusique, ME.Titre, COUNT(ME.IdMusique) AS NbEcoutés
    FROM MusiqueEcoutées ME
    WHERE ME.HeureEcouté >= TO_TIMESTAMP('01-jan-2017 00:00:00', 'DD-Mon-YYYY HH24:Mi:SS')
    AND ME.HeureEcouté <= TO_TIMESTAMP('31-dec-2017 23:59:59', 'DD-Mon-YYYY HH24:Mi:SS')
    GROUP BY ME.IdMusique, ME.Titre
    ORDER BY NbEcoutés DESC)
WHERE ROWNUM <= 5;
```

17. Suppressions de toutes les playlists vides

```
DELETE FROM GroupeMusique
WHERE IdGroupeMusique NOT IN (
    SELECT IdGroupeMusique
    FROM Compose );
```


18. Publication des playlists de l'utilisateur 1 :

```
UPDATE GroupeMusique
SET EstPrive = 'N'
WHERE IdGroupeMusique IN
(
    SELECT GroupeMusique.IdGroupeMusique
    FROM Utilisateur
    INNER JOIN Proprietaire ON Utilisateur.IdUtilisateur=Proprietaire.IdUtilisateur
    INNER JOIN GroupeMusique ON Proprietaire.IdGroupeMusique = GroupeMusique.IdGroupeMusique
    WHERE Utilisateur.IdUtilisateur = 1
);
```

19. Liste de tous les utilisateurs français ayant une adresse mail se terminant par 'dayrep.com' et étant propriétaires d'une playlist :

```
SELECT *
FROM Utilisateur U
INNER JOIN Proprietaire P ON U.IdUtilisateur = P.IdUtilisateur
WHERE U.Email LIKE '%@dayrep.com%' AND U.Nationalite = 'FR' ;
```

20. Pourcentage d'écoute par artiste :

```
SELECT Artiste.IdArtiste, Artiste.NomScene, SUM(NbEcoutes.TMP1 / NbEcoutes.TMP2 * 100) AS Pourcentage
FROM Artiste INNER JOIN Produit ON Artiste.IdArtiste = Produit.IdArtiste INNER JOIN (
    SELECT IdMusique, COUNT(IdMusique) AS TMP1, (
        SELECT COUNT(*)
        FROM MusiqueEcoutees) AS TMP2
    FROM MusiqueEcoutees
    GROUP BY IdMusique) NbEcoutes
ON Produit.IdMusique = NbEcoutes.IdMusique
GROUP BY Artiste.IdArtiste, Artiste.NomScene;
```


SQL avancé

A. Métadonnées

Liste des contraintes :

A l'aide de la requête SQL ci-dessous, on peut afficher les différentes contraintes d'intégrité de la base de données, classées par table et par type (C = contrainte Check, P = clé primaire, R = intégrité référentielle, U = clé unique)

```
SELECT Table_name, Constraint_name, Constraint_type, Column_name, Search_condition
FROM User_constraints NATURAL JOIN User_cons_columns
ORDER BY Table_name, Constraint_type;
```

Figure 41 : liste_orc_constraints

Liste des triggers :

De la même manière, on peut afficher les différents triggers mis en place sur nos tables.

```
SELECT Table_name, Trigger_name, Trigger_type, Trigger_body
FROM User_triggers
ORDER BY Table_name;
```

Figure 42 : liste_orc_triggers

Liste des indexes :

```
SELECT Table_name, Index_name, Index_type, Column_name, Uniqueness
FROM User_indexes NATURAL JOIN User_ind_columns
ORDER BY Table_name;
```

Figure 43 : liste_orc_indexes

Liste des tables (et vues) :

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, NULLABLE
FROM user_tab_columns
ORDER BY TABLE_NAME;
```

Figure 44 : liste_orc_tab

Liste des séquences :

```
SELECT SEQUENCE_NAME, MIN_VALUE, MAX_VALUE, INCREMENT_BY, LAST_NUMBER
FROM USER_SEQUENCES;
```

Figure 45 : liste_orc_seqs

B. Analyse et administration physique des données

Liste des utilisateurs n'aimant pas le Rock mais en ayant déjà écouté au moins une fois :

```
SELECT U.*
FROM Utilisateur U
WHERE U.IdUtilisateur NOT IN (
    SELECT IdUtilisateur
    FROM Aime A, StyleMusique SM
    WHERE A.IdStyleMusique = SM.IdStyleMusique AND SM.NomStyle = 'Rock')
INTERSECT
(
    SELECT U.*
    FROM Utilisateur U, MusiqueEcoutees EM, Appartient A, StyleMusique SM
    WHERE U.IdUtilisateur = EM.IdUtilisateur
    AND EM.IdMusique = A.IdMusique
    AND A.IdStyleMusique = SM.IdStyleMusique
    AND SM.NomStyle = 'Rock');
```

Plan d'exécution :

Ce plan d'exécution a été généré grâce à la commande EXPLAIN PLAN FOR [Requête] de SQL*Plus.

Il a ensuite été affiché grâce à la requête :

```
select * from table(dbms_xplan.display);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|------|-----------------------------|-----------------|------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | 1 | 3982 | 22 (73) | 00:00:01 |
| 1 | INTERSECTION | | | | | |
| 2 | SORT UNIQUE | | 29 | 3770 | 7 (15) | 00:00:01 |
| * 3 | FILTER | | | | | |
| 4 | TABLE ACCESS FULL | UTILISATEUR | 29 | 3770 | 3 (0) | 00:00:01 |
| 5 | NESTED LOOPS | | | | | |
| 6 | NESTED LOOPS | | 1 | 56 | 3 (0) | 00:00:01 |
| * 7 | INDEX RANGE SCAN | SYS_C0025852 | 1 | 26 | 2 (0) | 00:00:01 |
| * 8 | INDEX UNIQUE SCAN | SYS_C0025815 | 1 | | 0 (0) | 00:00:01 |
| * 9 | TABLE ACCESS BY INDEX ROWID | STYLEMUSIQUE | 1 | 30 | 1 (0) | 00:00:01 |
| 10 | SORT UNIQUE | | 1 | 212 | 15 (14) | 00:00:01 |
| 11 | NESTED LOOPS | | 1 | 212 | 14 (8) | 00:00:01 |
| * 12 | HASH JOIN | | 1 | 186 | 14 (8) | 00:00:01 |
| 13 | MERGE JOIN CARTESIAN | | 1 | 160 | 6 (0) | 00:00:01 |
| * 14 | TABLE ACCESS FULL | STYLEMUSIQUE | 1 | 30 | 3 (0) | 00:00:01 |
| 15 | BUFFER SORT | | 29 | 3770 | 3 (0) | 00:00:01 |
| 16 | TABLE ACCESS FULL | UTILISATEUR | 29 | 3770 | 3 (0) | 00:00:01 |
| 17 | VIEW | MUSIQUEECOUTEES | 507 | 13182 | 8 (13) | 00:00:01 |
| 18 | UNION-ALL | | | | | |
| 19 | TABLE ACCESS FULL | ECOUTEMUSIQUE | 157 | 4082 | 3 (0) | 00:00:01 |
| * 20 | HASH JOIN | | 350 | 18200 | 5 (20) | 00:00:01 |
| 21 | INDEX FAST FULL SCAN | SYS_C0025837 | 46 | 1196 | 2 (0) | 00:00:01 |
| 22 | INDEX FAST FULL SCAN | SYS_C0025827 | 99 | 2574 | 2 (0) | 00:00:01 |
| * 23 | INDEX UNIQUE SCAN | SYS_C0025847 | 1 | 26 | 0 (0) | 00:00:01 |

Figure 46 : Plan d'exécution

```
Predicate Information (identified by operation id):
-----

 3 - filter( NOT EXISTS (SELECT 0 FROM "STYLEMUSIQUE" "SM", "AIME" "A" WHERE
                        "IDUTILISATEUR"=:B1 AND "A"."IDSTYLEMUSIQUE"="SM"."IDSTYLEMUSIQUE" AND
                        "SM"."NOMSTYLE"='Rock'))
 7 - access("IDUTILISATEUR"=:B1)
 8 - access("A"."IDSTYLEMUSIQUE"="SM"."IDSTYLEMUSIQUE")
 9 - filter("SM"."NOMSTYLE"='Rock')
12 - access("U"."IDUTILISATEUR"="EM"."IDUTILISATEUR")
14 - filter("SM"."NOMSTYLE"='Rock')
20 - access("EGM"."IDGROUPEMUSIQUE"="C"."IDGROUPEMUSIQUE")
23 - access("EM"."IDMUSIQUE"="A"."IDMUSIQUE" AND
            "A"."IDSTYLEMUSIQUE"="SM"."IDSTYLEMUSIQUE")

Note
-----
- dynamic sampling used for this statement (level=2)
```

Figure 47 : Informations relatives au plan d'exécution

Ce plan correspond aux différentes opérations effectuées par Oracle pour exécuter la requête.

Il est calculé par l'optimiseur d'Oracle, le CBO (Cost-Based Optimizer), selon des critères de coûts.

Grâce à ce plan, on peut donc visualiser le coût de chaque opération effectuée par le SGBD en terme de temps, de pourcentage de CPU utilisé, et de lignes retournées.

Le plan d'exécution nous décrit également les méthodes d'accès aux tables, par exemple :

- TABLE ACCESS FULL = Accès via un scan complet de la table
- TABLE ACCESS BY INDEX = Accès via un index

Mais aussi l'ordre et la méthode de jointure adoptée, par exemple :

- NESTED LOOPS JOIN = Les deux tables sont jointes en récupérant d'abord le résultat d'une table puis en recherchant la condition de jointure dans l'autre table, en la parcourant ligne par ligne
- HASH JOIN = Jointure par utilisation d'une table de hachage
- MERGE JOIN = Jointure par combinaison des deux tables (qui doivent être triées)

On remarque également les dépendances entre les opérations, indiquées par les tabulations. Ainsi, l'opération INTERSECTION (1) ne pourra avoir lieu avant que les opérations 3 et 10 (et toutes leurs dépendances) soient traitées.

Grâce aux prédicats fournis sous le plan, il est possible d'identifier la jointure et les filtres correspondants à l'opération sur le plan.

Il faut donc lire ce plan de bas en haut, de l'intérieur vers l'extérieur.

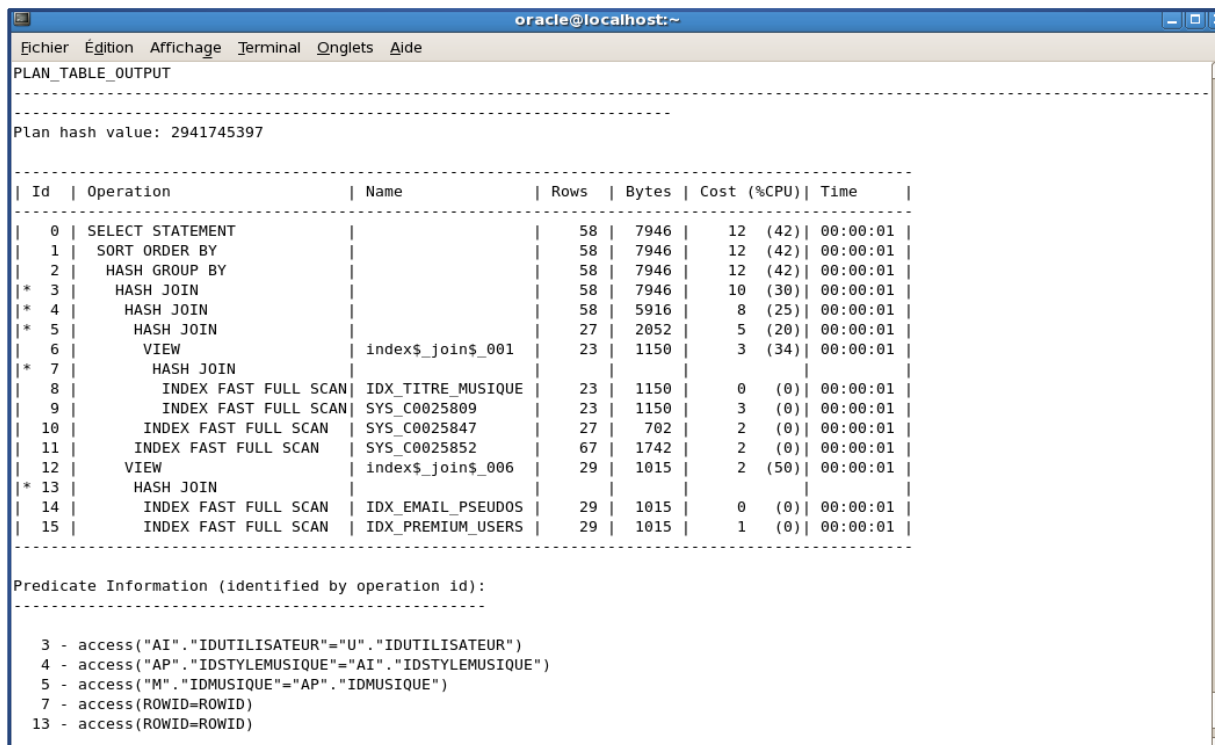
On remarque que les opérations de test de valeurs (d'égalité, ou opérateur IN/NOT IN) sont présents sous le nom 'FILTER'.

Concernant le déroulement de cette requête, les opérations concernant la vue MusiqueEcoutees et ses jointures sont exécutées en premiers, puis la sous-requête supérieure est exécutée, et enfin l'opération INTERSECT effectue l'intersection des deux sous-requêtes.

Musiques recommandées pour les utilisateurs selon leurs goûts (avec facteur de recommandation) :

```
SELECT U.IdUtilisateur, U.Pseudo, M.IdMusique, M.Titre, COUNT(M.IdMusique) AS FacteurRecommandation
FROM Musique M
INNER JOIN Appartient AP ON M.IdMusique = AP.IdMusique
INNER JOIN Aime AI ON AP.IdStyleMusique = AI.IdStyleMusique
INNER JOIN Utilisateur U ON AI.IdUtilisateur = U.IdUtilisateur
GROUP BY U.IdUtilisateur, U.Pseudo, M.IdMusique, M.Titre
ORDER BY U.IdUtilisateur, FacteurRecommandation DESC;
```

Plan d'exécution :



PLAN_TABLE_OUTPUT

Plan hash value: 2941745397

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|------|----------------------|--------------------|------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | 58 | 7946 | 12 (42) | 00:00:01 |
| 1 | SORT ORDER BY | | 58 | 7946 | 12 (42) | 00:00:01 |
| 2 | HASH GROUP BY | | 58 | 7946 | 12 (42) | 00:00:01 |
| * 3 | HASH JOIN | | 58 | 7946 | 10 (30) | 00:00:01 |
| * 4 | HASH JOIN | | 58 | 5916 | 8 (25) | 00:00:01 |
| * 5 | HASH JOIN | | 27 | 2052 | 5 (20) | 00:00:01 |
| 6 | VIEW | index\$_join\$_001 | 23 | 1150 | 3 (34) | 00:00:01 |
| * 7 | HASH JOIN | | | | | |
| 8 | INDEX FAST FULL SCAN | IDX_TITRE_MUSIQUE | 23 | 1150 | 0 (0) | 00:00:01 |
| 9 | INDEX FAST FULL SCAN | SYS_C0025809 | 23 | 1150 | 3 (0) | 00:00:01 |
| 10 | INDEX FAST FULL SCAN | SYS_C0025847 | 27 | 702 | 2 (0) | 00:00:01 |
| 11 | INDEX FAST FULL SCAN | SYS_C0025852 | 67 | 1742 | 2 (0) | 00:00:01 |
| 12 | VIEW | index\$_join\$_006 | 29 | 1015 | 2 (50) | 00:00:01 |
| * 13 | HASH JOIN | | | | | |
| 14 | INDEX FAST FULL SCAN | IDX_EMAIL_PSEUDOS | 29 | 1015 | 0 (0) | 00:00:01 |
| 15 | INDEX FAST FULL SCAN | IDX_PREMIUM_USERS | 29 | 1015 | 1 (0) | 00:00:01 |

Predicate Information (identified by operation id):

```

3 - access("AI"."IDUTILISATEUR"="U"."IDUTILISATEUR")
4 - access("AP"."IDSTYLEMUSIQUE"="AI"."IDSTYLEMUSIQUE")
5 - access("M"."IDMUSIQUE"="AP"."IDMUSIQUE")
7 - access(ROWID=ROWID)
13 - access(ROWID=ROWID)
```

Figure 48 : Plan d'exécution

Pour cette requête, on remarque que nos indexes précédemment définis (après la création des tables) ont été utilisés pour parcourir de manière plus performante la table Utilisateur. On notera également que l'ordre des jointures diffère de celui que nous avons mis en place, puisque la jointure entre Aime et Utilisateur (3) est effectuée après celle entre Appartient et Aime (4), et cette dernière est elle-même effectuée après la jointure entre Musique et Appartient (5). (Nous l'avons écrit dans l'ordre (5) – (4) – (3)).

Le GROUP BY et le ORDER BY apparaissent eux également sous forme de jointures, en bas de tableau.

Conclusion

Ce projet de Bases de Données Avancées nous a permis de mettre en pratique les notions étudiées en cours, et ce de manière plus concrète et plus poussée que durant les Travaux Dirigés de ce module.

De fait, de la modélisation de notre base, en passant par sa création et son remplissage, jusqu'à son interrogation et son étude avancée, ce projet nous a permis d'apprendre la mise en place d'une base de donnée et la gestion d'une base de donnée, bien que la taille de celle-ci soit relativement réduite.

En effet, du fait de certaines limites fixées dans l'énoncé (nombre de tables), et d'autres que nous nous sommes fixées, cette base de donnée n'est pas complète, et ne serait adaptée que pour une version (très) minimaliste de ce type d'application.

Parmi les concessions effectuées, on notera les suivantes :

- Lors d'une écoute de musique, on conçoit que l'utilisateur écoute la musique dans sa totalité.
- Lors d'une écoute d'un groupe de musique (playlist ou album), on conçoit que l'utilisateur écoute la totalité des musiques composant ce groupe de musique.

Malgré cela, ce projet s'est révélé très instructif, et l'expérience qu'il nous a apporté nous sera forcément utile au cours de notre vie professionnelle, au cours de laquelle nous serons certainement amenés à travailler sur des bases de données (relationnelles ou objets) et y effectuer des opérations similaires.

Webographie

<https://sql.sh/>

<https://www.w3schools.com/sql/>

<https://stackoverflow.com/>

<https://www.developpez.net/>