HAYE Marc
BENHALIMA Mattéo
DAOUDI Oussama
FUCHS Théo
S5

SAE 2.04 - Livrable 2

Sommaire

Modèle Entité-Association

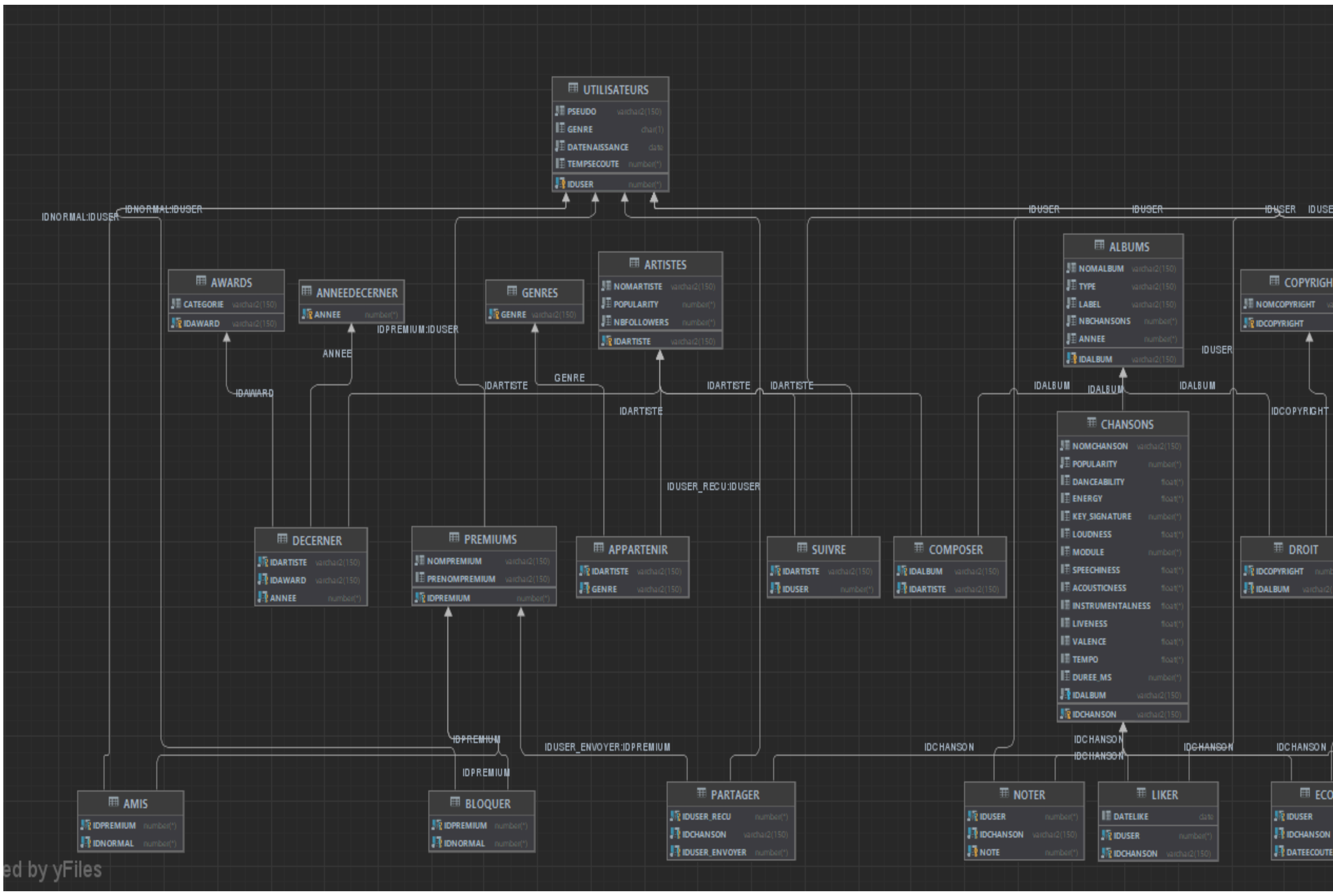

Schéma Relationnel

Artistes (IdArtiste, nomArtiste, Popularity, nbFollower);

Amis (#idPremium, #idNormal)

Utilisateurs (idUser, pseudo, genre ,dateNaissance, tempsEcoute);

Premiums (#idPremium, nomPremium, prenomPremium);

Awards (idAward, categorie);

Copyrights (idCopyright , nomCopyright);

Genres (genre);

Droit (#idCopyright, #idAlbum);

Dossiers (nomDossier, #idUser, #nomDossierParent);

DatesEcoute (dateEcoute);

AnneeDecerner (annee);

Albums (IdAlbum, nomAlbum, type, label, nbChansons, annee);

Chansons (idChanson, nomChanson, popularity, danceAbility, energy, key_signature, loudness, module, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duree_ms, #IdAlbum);

Playlists (nomPlaylist, #nomDossierParent, #idUser);

Noter (#idUser, #idChanson, note);

Liker (#idUser, #idChanson, dateLike);

Partager (#idUser_Recu, #idChanson, #idUser_Envoyer);

Decerner (#IdArtiste, #idAward, #annee);

Suivre (#IdArtiste, #idUser);

Composer (<u>#IdAlbum, #IdArtiste</u>);

Bloquer (<u>#idNormal,#idPremium</u>);

EtrePresent (<u>#idChanson, #nomPlaylist, #idUser</u>, ordre);

Appartenir (<u>#IdArtiste, #genre</u>);

Ecouter (<u>#idUser, #idChanson, #dateEcoute</u>);

Scripts Creations Tables et Insertions

```
DROP TRIGGER trg_Bloquer_Exclusion;
DROP TRIGGER trg_Amis_Exclusion;
DROP TABLE DROIT CASCADE CONSTRAINTS;
DROP TABLE Ecouter CASCADE CONSTRAINTS;
DROP TABLE Appartenir CASCADE CONSTRAINTS;
DROP TABLE EtrePresent CASCADE CONSTRAINTS;
DROP TABLE Bloquer CASCADE CONSTRAINTS;
DROP TABLE Amis CASCADE CONSTRAINTS;
DROP TABLE Composer CASCADE CONSTRAINTS;
DROP TABLE Suivre CASCADE CONSTRAINTS;
DROP TABLE Decerner CASCADE CONSTRAINTS;
DROP TABLE Partager CASCADE CONSTRAINTS;
DROP TABLE Liker CASCADE CONSTRAINTS;
DROP TABLE Noter CASCADE CONSTRAINTS;
DROP TABLE Playlists CASCADE CONSTRAINTS;
DROP TABLE Chansons CASCADE CONSTRAINTS;
DROP TABLE Albums CASCADE CONSTRAINTS;
DROP TABLE AnneeDecerner CASCADE CONSTRAINTS;
DROP TABLE DatesEcoute CASCADE CONSTRAINTS;
DROP TABLE Dossiers CASCADE CONSTRAINTS;
DROP TABLE Genres CASCADE CONSTRAINTS;
DROP TABLE Copyrights CASCADE CONSTRAINTS;
DROP TABLE Awards CASCADE CONSTRAINTS;
DROP TABLE Premiums CASCADE CONSTRAINTS;
DROP TABLE Utilisateurs CASCADE CONSTRAINTS;
DROP TABLE Artistes CASCADE CONSTRAINTS;
```

```
CREATE TABLE Artistes(
                    IdArtiste VARCHAR(150),
                    nomArtiste VARCHAR(150) NOT NULL,
                    popularity INT NOT NULL,
                    NBfollowers INT NOT NULL,
                    PRIMARY KEY(IdArtiste),
                    CONSTRAINT pop_art_entre0et100 CHECK ( popularity<=100 AND
popularity>=0 )
);
CREATE TABLE Utilisateurs(
                              idUser INT,
                              pseudo VARCHAR(150) NOT NULL,
                              genre CHAR ,
                              dateNaissance DATE NOT NULL,
                              tempsEcoute INT,
                              PRIMARY KEY(idUser),
                              CONSTRAINT format_genre CHECK (genre IN('H','F'))
);
CREATE TABLE Premiums(
                    idPremium INT,
                    nomPremium VARCHAR(150) NOT NULL,
                    prenomPremium VARCHAR(150),
                    PRIMARY KEY(idPremium),
                    FOREIGN KEY(idPremium) REFERENCES Utilisateurs(idUser)
);
CREATE TABLE Awards(
                    idAward VARCHAR(150),
                    categorie VARCHAR(150) NOT NULL,
                    PRIMARY KEY(idAward)
);
CREATE TABLE Copyrights(
                    idCopyright INT,
                    nomCopyright VARCHAR(250) NOT NULL,
                    PRIMARY KEY(idCopyright)
);
CREATE TABLE Genres(
                    genre VARCHAR(150),
                    PRIMARY KEY(genre)
);
CREATE TABLE Dossiers (
                    nomDossier VARCHAR(150),
                    idUser INT,
                    nomDossierParent VARCHAR(150),
                    PRIMARY KEY (nomDossier, idUser),
                    FOREIGN KEY (idUser) REFERENCES Utilisateurs(idUser),
                    FOREIGN KEY (nomDossierParent, idUser) REFERENCES
Dossiers(nomDossier, idUser)
);

CREATE TABLE DatesEcoute(
                              dateEcoute DATE,
                              PRIMARY KEY(dateEcoute)
);
CREATE TABLE AnneeDecerner(
                              annee INT,
                              PRIMARY KEY(annee)
);
CREATE TABLE Albums(
                    IdAlbum VARCHAR(150),
                    nomAlbum VARCHAR(150) NOT NULL,
                    type VARCHAR(150) NOT NULL,
                    label VARCHAR(150) NOT NULL,
                    nbChansons INT NOT NULL,
                    annee INT NOT NULL,
                    PRIMARY KEY(IdAlbum)
                    );
CREATE TABLE Chansons(
                    idChanson VARCHAR(150),
                    nomChanson VARCHAR(150) NOT NULL,
                    popularity INT NOT NULL,
                    danceAbility FLOAT,
                    energy FLOAT,
                    key_signature INT,
                    loudness FLOAT,
                    module INT,
                    speechiness FLOAT,
                    acousticness FLOAT,
                    instrumentalness FLOAT,
                    liveness FLOAT,
                    valence FLOAT,
                    tempo FLOAT,
                    duree_ms INT,
                    IdAlbum VARCHAR(150) NOT NULL,
                    PRIMARY KEY(idChanson),
                    FOREIGN KEY(IdAlbum) REFERENCES Albums(IdAlbum),
                    CONSTRAINT key_signature_positive check ( key_signature>=0 ),
                    CONSTRAINT module_positive check ( module>=0 ),
                    CONSTRAINT tempo_positive check ( tempo>=0 ),
                    CONSTRAINT duree_ms_positive check ( duree_ms>=0 ),
		CONSTRAINT pop_cha_entre0et100 CHECK ( popularity<=100 AND
popularity>=0 ),
		CONSTRAINT danceAbility_entre0et1 CHECK ( danceAbility<=1 AND
danceAbility>=0 ),
		CONSTRAINT energy_entre0et1 CHECK ( energy<=1 AND energy>=0 ),
		CONSTRAINT loudness_plus_petit0 CHECK ( loudness<=0),
		CONSTRAINT speechiness_entre0et1 CHECK ( speechiness<=1 AND
speechiness>=0 ),
		CONSTRAINT acousticness_entre0et1 CHECK ( acousticness<=1 AND
acousticness>=0 ),
		CONSTRAINT instrumentalness_entre0et1 CHECK ( instrumentalness<=1
AND
						instrumentalness>=0 ),
		CONSTRAINT liveness_entre0et1 CHECK ( liveness<=1 AND liveness>=0 ),
		CONSTRAINT valence_entre0et1 CHECK ( valence<=1 AND valence>=0 )
);
CREATE TABLE Playlists (
		nomPlaylist VARCHAR(150),
		nomDossierParent VARCHAR(150),
		idUser INT NOT NULL,
		PRIMARY KEY (nomPlaylist, idUser),
		FOREIGN KEY (nomDossierParent, idUser) REFERENCES
Dossiers(nomDossier, idUser),
		FOREIGN KEY (idUser) REFERENCES Utilisateurs(idUser)
);

CREATE TABLE Noter(
		idUser INT,
		idChanson VARCHAR(150),
		note INT,
		PRIMARY KEY(idUser, idChanson,note),
		FOREIGN KEY(idUser) REFERENCES Utilisateurs(idUser),
		FOREIGN KEY(idChanson) REFERENCES Chansons(idChanson),
		CONSTRAINT note_entre0et20 CHECK ( note<=20 AND note>=0 )
);
CREATE TABLE Liker(
		idUser INT,
		idChanson VARCHAR(150),
		dateLike DATE,
		PRIMARY KEY(idUser, idChanson),
		FOREIGN KEY(idUser) REFERENCES Utilisateurs(idUser),
		FOREIGN KEY(idChanson) REFERENCES Chansons(idChanson)
);
CREATE TABLE Partager(
		idUser_Recu INT,
		idChanson VARCHAR(150),
                    idUser_Envoyer INT,
                    PRIMARY KEY(idUser_Envoyer, idChanson, idUser_Recu),
                    FOREIGN KEY(idUser_Recu) REFERENCES Utilisateurs(idUser),
                    FOREIGN KEY(idChanson) REFERENCES Chansons(idChanson),
                    FOREIGN KEY(idUser_Envoyer) REFERENCES Premiums(idPremium)
);
CREATE TABLE Decerner(
                    IdArtiste VARCHAR(150),
                    idAward VARCHAR(150),
                    annee INT,
                    PRIMARY KEY(IdArtiste, idAward, annee),
                    FOREIGN KEY(IdArtiste) REFERENCES Artistes(IdArtiste),
                    FOREIGN KEY(idAward) REFERENCES Awards(idAward),
                    FOREIGN KEY(annee) REFERENCES AnneeDecerner(annee)
);
CREATE TABLE Suivre(
                    IdArtiste VARCHAR(150),
                    idUser INT,
                    PRIMARY KEY(IdArtiste, idUser),
                    FOREIGN KEY(IdArtiste) REFERENCES Artistes(IdArtiste),
                    FOREIGN KEY(idUser) REFERENCES Utilisateurs(idUser)
);
CREATE TABLE Composer(
                    IdAlbum VARCHAR(150),
                    IdArtiste VARCHAR(150),
                    PRIMARY KEY(IdAlbum, IdArtiste),
                    FOREIGN KEY(IdAlbum) REFERENCES Albums(IdAlbum),
                    FOREIGN KEY(IdArtiste) REFERENCES Artistes(IdArtiste)
);
CREATE TABLE Amis(
                    idPremium INT,
                    idNormal INT,
                    PRIMARY KEY(idNormal, idPremium),
                    FOREIGN KEY(idNormal) REFERENCES Utilisateurs(idUser),
                    FOREIGN KEY(idPremium) REFERENCES Premiums(idPremium)
);
CREATE TABLE Bloquer(
                    idPremium INT,
                    idNormal INT,
                    PRIMARY KEY(idNormal, idPremium),
                    FOREIGN KEY(idNormal) REFERENCES Utilisateurs(idUser),
                    FOREIGN KEY(idPremium) REFERENCES Premiums(idPremium)
);
CREATE TABLE EtrePresent (
                              idChanson VARCHAR(150),
                              nomPlaylist VARCHAR(150),
                              idUser INT,
                              ordre INT,
                              PRIMARY KEY (idChanson, nomPlaylist, idUser),
                              FOREIGN KEY (idChanson) REFERENCES Chansons(idChanson),
                              FOREIGN KEY (nomPlaylist, idUser) REFERENCES
Playlists(nomPlaylist, idUser),
                              CONSTRAINT ordre_positive CHECK (ordre >= 0)
);

CREATE TABLE Appartenir(
                    IdArtiste VARCHAR(150),
                    genre VARCHAR(150),
                    PRIMARY KEY(IdArtiste, genre),
                    FOREIGN KEY(IdArtiste) REFERENCES Artistes(IdArtiste),
                    FOREIGN KEY(genre) REFERENCES Genres(genre)
);
CREATE TABLE Ecouter(
                    idUser INT,
                    idChanson VARCHAR(150),
                    dateEcoute DATE,
                    PRIMARY KEY(idUser, idChanson, dateEcoute),
                    FOREIGN KEY(idUser) REFERENCES Utilisateurs(idUser),
                    FOREIGN KEY(idChanson) REFERENCES Chansons(idChanson),
                    FOREIGN KEY(dateEcoute) REFERENCES DatesEcoute(dateEcoute)
);

CREATE TABLE DROIT(
                    IDCOPYRIGHT INT,
                    IDALBUM VARCHAR(150),
                    PRIMARY KEY (IDALBUM,IDCOPYRIGHT),
                    FOREIGN KEY (IDALBUM) REFERENCES ALBUMS(IdAlbum),
                    FOREIGN KEY (IDCOPYRIGHT) REFERENCES COPYRIGHTS(idCopyright)
);


CREATE OR REPLACE TRIGGER trg_Amis_Exclusion
          BEFORE INSERT OR UPDATE ON Amis
          FOR EACH ROW
DECLARE
          v_count NUMBER;
          conflit_amis EXCEPTION;
BEGIN
          SELECT COUNT(*)
          INTO v_count
          FROM Bloquer
          WHERE (idPremium = :new.idPremium AND idNormal = :new.idNormal)
```

```
        OR (idPremium = :new.idNormal AND idNormal = :new.idPremium);
        IF v_count > 0 THEN
        RAISE conflit_amis;
        END IF;
EXCEPTION
        WHEN conflit_amis THEN
        RAISE_APPLICATION_ERROR(-20111, 'Les utilisateurs ne peuvent pas être à la
fois amis et
bloqués.');
END;
/
CREATE OR REPLACE TRIGGER trg_Bloquer_Exclusion
        BEFORE INSERT OR UPDATE ON Bloquer
        FOR EACH ROW
DECLARE
        v_count NUMBER;
        bloquer_conflit EXCEPTION;
BEGIN
        SELECT COUNT(*)
        INTO v_count
        FROM Amis
        WHERE (idPremium = :new.idPremium AND idNormal = :new.idNormal)
        OR (idPremium = :new.idNormal AND idNormal = :new.idPremium);
        IF v_count > 0 THEN
        RAISE bloquer_conflit;
        END IF;
EXCEPTION
        WHEN bloquer_conflit THEN
        RAISE_APPLICATION_ERROR(-20111, 'Les utilisateurs ne peuvent pas être amis et
bloqués en même
temps.');
        END;
/

INSERT INTO ARTISTES(idartiste, nomartiste, popularity, nbfollowers)
SELECT DISTINCT IDARTIST,NAME,POPULARITY,FOLLOWERS
FROM TEMPARTISTES;
INSERT INTO UTILISATEURS(iduser, pseudo, genre, datenaissance, tempsecoute)
SELECT DISTINCT IDUSER,PSEUDO,GENDER,DATEOFBIRTH,TIMEPLAYING
FROM TEMPUTILISATEURS;
INSERT INTO PREMIUMS(idpremium, nompremium, prenompremium)
SELECT DISTINCT IDUSERPREMIUM,NAMEPREMIUM,SURNAMEPREMIUM
FROM TEMPAMIS;
INSERT INTO AWARDS(idaward, categorie)
SELECT DISTINCT AWARD,CATEGORY
FROM TEMPARTISTES
```

```
WHERE AWARD IS NOT NULL ;
INSERT INTO COPYRIGHTS(idcopyright, nomcopyright)
SELECT DISTINCT TEMPALBUMS.IDCOPYRIGHT,TEMPALBUMS.COPYRIGHT
FROM TEMPALBUMS;
INSERT INTO GENRES(GENRE)
SELECT DISTINCT GENRE
FROM TEMPARTISTES
WHERE GENRE IS NOT NULL ;
INSERT INTO DOSSIERS (NOMDOSSIER, IDUSER, nomDossierParent)
select DISTINCT NAMEELEMENT,idUser,NAMEELEMENTPARENT FROM
TEMPPLAYLISTS
WHERE "type"='Folder';
INSERT INTO DATESECOUTE(DATEECOUTE)
SELECT DISTINCT DATELISTEN
FROM TEMPCHANSONS
WHERE DATELISTEN IS NOT NULL ;
INSERT INTO ANNEEDECERNER(ANNEE)
SELECT DISTINCT "year"
FROM TEMPARTISTES
WHERE "year" IS NOT NULL;

INSERT INTO ALBUMS(ALBUMS.IDALBUM, NOMALBUM, TYPE, LABEL, NBCHANSONS,
annee)
SELECT DISTINCT
TEMPALBUMS.IDALBUM,NAME,TEMPALBUMS."type",LABEL,TOTAL_TRACKS,"year"
FROM TEMPALBUMS;

INSERT INTO CHANSONS(idchanson, nomchanson, popularity, danceability, energy,
                    key_signature, loudness, module, speechiness, acousticness,
instrumentalness, liveness,
                    valence, tempo, duree_ms, idalbum)
SELECT DISTINCT
IDTRACK,NAME,POPULARITY,DANCEABILITY,ENERGY,KEY_SIGNATURE,LOUDNESS,"
mode",SPEECHINESS,ACOUSTICNESS,INSTRUMENTALNESS,LIVENESS,VALENCE,TE
MPO,DURATION_MS,IDALBUM
FROM TEMPCHANSONS;
INSERT INTO PLAYLISTS ( NOMPLAYLIST, IDUSER, nomDossierParent)
select DISTINCT NAMEELEMENT,idUser,NAMEELEMENTPARENT
          FROM TEMPPLAYLISTS WHERE "type" = 'Playlist';
INSERT INTO NOTER(iduser, idchanson, note)
SELECT DISTINCT IDUSER,IDTRACK,NOTE
FROM TEMPEVALUATIONS;
INSERT INTO LIKER(iduser, idchanson, datelike)
SELECT IDUSER,IDTRACK,DATELIKE
FROM TEMPLIKES;
INSERT INTO PARTAGER(iduser_recu, idchanson, iduser_envoyer)
```

```
SELECT DISTINCT IDUSERSHARE,IDTRACK,IDUSER
FROM TEMPPARTAGER
WHERE EXISTS (SELECT * FROM PREMIUMS u WHERE
u.IDPREMIUM=TEMPPARTAGER.IDUSER)
AND EXISTS (SELECT * FROM PREMIUMS u WHERE
u.IDPREMIUM=TEMPPARTAGER.IDUSERSHARE)
AND EXISTS(SELECT IDTRACK FROM CHANSONS c WHERE
c.IDCHANSON=TEMPPARTAGER.IDTRACK);
INSERT INTO DECERNER(idartiste, idaward, annee)
SELECT DISTINCT IDARTIST,AWARD,"year"
FROM TEMPARTISTES
WHERE "year" IS NOT NULL;
INSERT INTO SUIVRE(idartiste, iduser)
SELECT IDARTISTFOLLOW,IDUSER
FROM TEMPUTILISATEURS
WHERE IDARTISTFOLLOW IS NOT NULL ;
INSERT INTO COMPOSER(idalbum, idartiste)
SELECT DISTINCT IDALBUM,IDARTIST
FROM TEMPALBUMS;
INSERT INTO AMIS(idpremium, idnormal)
SELECT IDUSERPREMIUM,IDUSERFRIEND
FROM TEMPAMIS;
insert into BLOQUER (IDPREMIUM, IDNORMAL)
Select IDUSERPREMIUM , iduserblocked
from TEMPUSERS_BLOCKED
WHERE EXISTS (SELECT * FROM PREMIUMS WHERE
idPremium=TEMPUSERS_BLOCKED.IDUSERPREMIUM)
AND EXISTS (SELECT * FROM UTILISATEURS WHERE
idUser=TEMPUSERS_BLOCKED.IDUSERBLOCKED);
INSERT INTO ETREPRESENT(idchanson, nomPlaylist,idUser, ordre)
SELECT IDTRACK,NAMEELEMENT,IDUSER,ORDRE
FROM TEMPPLAYLISTS
WHERE "type"='Playlist';
INSERT INTO APPARTENIR(idartiste, genre)
SELECT DISTINCT IDARTIST,GENRE
FROM TEMPARTISTES
WHERE GENRE IS NOT NULL;
INSERT INTO ECOUTER(iduser, idchanson, dateecoute)
SELECT DISTINCT IDUSER,IDTRACK,DATELISTEN
FROM TEMPCHANSONS
WHERE IDUSER is not null;
INSERT INTO DROIT(idcopyright, idalbum)
SELECT DISTINCT IDCOPYRIGHT,IDALBUM
FROM TEMPALBUMS;
```

Vues

1. **Pour chaque morceau (ou piste) le nombre d'interprètes, le nombre d'écoutes, le nombre d'écoutes en cours, l'évaluation moyenne, le nombre de likes, le nombre de playlists qui contiennent le morceau, le nombre de playlists où le morceau est en première position, le nombre de partages.**

```
CREATE OR REPLACE VIEW VueInfoMorceau AS

    SELECT c.idChanson, c.nomChanson, (SELECT COUNT(DISTINCT co.IdArtiste)

            FROM Composer co

                JOIN Albums al ON co.IdAlbum = al.IdAlbum

            WHERE al.IdAlbum = c.IdAlbum) AS nbInterpretes, (

            SELECT COUNT(*)

            FROM Ecouter e

            WHERE e.idChanson = c.idChanson) AS nbEcoutes, (

            SELECT COUNT(*)

            FROM Ecouter e

            WHERE e.idChanson = c.idChanson

                AND e.dateEcoute = TRUNC(SYSDATE)) AS

                nbEcoutesEnCours, (SELECT AVG(n.note)

                    FROM Noter n

                    WHERE n.idChanson = c.idChanson) AS
evaluationMoyenne, (

                    SELECT COUNT(*)

                    FROM Liker l

                    WHERE l.idChanson = c.idChanson) AS nbLikes, (

                    SELECT COUNT(DISTINCT ep.nomPlaylist)
```

```
                                        FROM EtrePresent ep

                                        WHERE ep.idChanson = c.idChanson) AS
nbPlaylistsContenant, (

                                        SELECT COUNT(*)

                                        FROM EtrePresent ep

                                        WHERE ep.idChanson = c.idChanson

                                              AND ep.ordre = 1) AS
nbPlaylistsPremierePosition, (

                                              SELECT COUNT(*)

                                              FROM Partager p

                                              WHERE p.idChanson = c.idChanson) AS
nbPartages

      FROM Chansons c;
```

2. **<u>Pour chaque album, le nombre de morceaux, le morceau le moins écouté et le morceau le plus écouté.</u>**

```
create or replace view nbEcouteChanson(idChanson,nbEcoute) as

select IDCHANSON , count(IDUSER)

from ECOUTER

group by IDCHANSON;




create or replace view
InfosAlbums(idAlbums,nbMorceaux,morceauxLeMoinsEcouter,morceauxLePlusEcou
ter) as

select IDALBUM, count(IDCHANSON) ,

      (select idChanson

      from nbEcouteChanson
```

```
        where nbEcoute = (select min(nbEcoute)
                from nbEcouteChanson nb
                join CHANSONS c1 on nb.idChanson=c1.IDCHANSON
                where c1.IDALBUM = c.IDALBUM)),
        (select idChanson
        from nbEcouteChanson
        where nbEcoute = (select max(nbEcoute)
                from nbEcouteChanson nb
                join CHANSONS c1 on nb.idChanson=c1.IDCHANSON
                where c1.IDALBUM = c.IDALBUM))
    from CHANSONS c
    group by IDALBUM
```

3. **<u>Le morceau le plus écouté pour chacun des signes astrologiques des utilisateurs.</u>**

```
CREATE OR REPLACE FUNCTION get_signeAstrologique (
    current_date IN DATE
) RETURN VARCHAR2
    IS
    current_day NUMBER;
    current_month NUMBER;
BEGIN
    -- Tronquer la date pour obtenir le jour précis de l'année (en ignorant l'année)
```

```
current_day := EXTRACT(current_date, 'DAY');

current_month := EXTRACT(current_date, 'MONTH');


IF (current_month=12 AND current_day>=23) THEN

return 'Capricorne';

ELSIF current_month=12 OR current_month=11 AND current_day >=23 THEN

return 'Sagittaire';

ELSIF current_month=11 OR current_month=10 AND current_day >=24 THEN

return 'Scorpion';

ELSIF current_month=10 OR current_month=09 AND current_day >=23 THEN

return 'Balance';

ELSIF current_month=9 OR current_month=08 AND current_day >=23 THEN

return 'Vierge';

ELSIF current_month=8 OR current_month=07 AND current_day >=23 THEN

return 'Lion';

ELSIF current_month=7 OR current_month=06 AND current_day >=22 THEN

return 'Cancer';

ELSIF current_month=6 OR current_month=05 AND current_day >=21 THEN

return 'Gémeaux';

ELSIF current_month=5 OR current_month=04 AND current_day >=20 THEN

return 'Taureau';

ELSIF current_month=4 OR current_month=03 AND current_day >=21 THEN

return 'Bélier';

ELSIF current_month=3 OR current_month=02 AND current_day >=20 THEN
```

```
        return 'Poissons';

        ELSIF current_month=2 OR current_month=01 AND current_day >=21 THEN

        return 'Verseau';

        ELSE

        return 'Capricorne';

        end if;

END;




create or replace view signesAstro(idUser,signeAstro) as

select idUser , get_signeAstrologique(DATENAISSANCE)

from UTILISATEURS;






create or replace view nbecouteChansonSigneAstro(idChanson,signeAstro,nbEcoute) as

select IDCHANSON,signeAstro,count(DATEECOUTE)

from ECOUTER e

join signesAstro sA on e.IDUSER = sA.idUser

group by IDCHANSON,signeAstro;




create or replace view
morceauxPlusEcouterPourChaqueSigneAstro(signeAstro,nomChanson) as
```

```
select signeAstro, NOMCHANSON

from nbecouteChansonSigneAstro s1

join CHANSONS C2 on s1.idChanson = C2.IDCHANSON

where nbEcoute = (select max(nbEcoute)

                  from nbecouteChansonSigneAstro s2

                  where S2.signeAstro = s1.signeAstro)

group by signeAstro,NOMCHANSON,s1.idChanson;
```

4. **<u>Les 3 artistes les plus écoutés par genre de musique (attention il peut y avoir des exæquo).</u>**

```
CREATE OR REPLACE VIEW Top3Artistes AS

        SELECT g.genre, a.nomArtiste, COUNT(e.idChanson) AS totalEcoutes, rnk

        FROM Ecouter e

        JOIN Chansons c ON e.idChanson = c.idChanson

        JOIN Albums al ON c.IdAlbum = al.IdAlbum

        JOIN Composer co ON al.IdAlbum = co.IdAlbum

        JOIN Artistes a ON co.IdArtiste = a.IdArtiste

        JOIN Appartenir ap ON a.IdArtiste = ap.IdArtiste

        JOIN Genres g ON ap.genre = g.genre

        JOIN ( SELECT ap.genre, a.nomArtiste, COUNT(e.idChanson) AS totalEcoutes,
RANK() OVER (PARTITION BY ap.genre ORDER BY COUNT(e.idChanson) DESC) AS rnk

        FROM Ecouter e

                JOIN Chansons c ON e.idChanson = c.idChanson

                JOIN Albums al ON c.IdAlbum = al.IdAlbum

                JOIN Composer co ON al.IdAlbum = co.IdAlbum
        JOIN Artistes a ON co.IdArtiste = a.IdArtiste

        JOIN Appartenir ap ON a.IdArtiste = ap.IdArtiste

    GROUP BY ap.genre, a.nomArtiste ) ranked_artistes ON ranked_artistes.genre =
g.genre

                                AND ranked_artistes.nomArtiste = a.nomArtiste

                                AND ranked_artistes.totalEcoutes =
totalEcoutes

    WHERE ranked_artistes.rnk <= 3

    GROUP BY g.genre, a.nomArtiste, totalEcoutes, rnk

    ORDER BY g.genre, totalEcoutes DESC;
```

5. **Pour chaque utilisateur, le pseudo de l'utilisateur et le nom du morceau le plus écouté parmi ceux qui se trouvent dans ses playlists. Pour ce morceau, on veut indiquer le chemin dans la playlist (par exemple dossier1/dossier11/PlaylistNemard/La quête).**

```
    CREATE OR REPLACE VIEW UtilisateursTopMorceaux AS

        WITH EcoutesParUtilisateur AS (

        SELECT e.idUser, e.idChanson, COUNT(e.idChanson) AS nbEcoutes

                FROM Ecouter e

                GROUP BY e.idUser, e.idChanson ),

        MorceauPlusEcoute AS (

        SELECT ep.idUser, ep.idChanson, RANK() OVER (PARTITION BY ep.idUser

                    ORDER BY ep.nbEcoutes DESC) AS rnk

                    FROM EcoutesParUtilisateur ep ),

        CheminPlaylists AS (
        SELECT p.nomPlaylist, d1.nomDossier AS dossierNiveau1, d2.nomDossier
AS dossierNiveau2, d3.nomDossier AS dossierNiveau3, CASE

        WHEN d3.nomDossier IS NOT NULL THEN d1.nomDossier || '/' ||
d2.nomDossier || '/' || d3.nomDossier || '/' || p.nomPlaylist

        WHEN d2.nomDossier IS NOT NULL THEN d1.nomDossier || '/' ||
d2.nomDossier || '/' || p.nomPlaylist

        ELSE d1.nomDossier || '/' || p.nomPlaylist END AS chemin

        FROM Playlists p

        JOIN Dossiers d1 ON p.nomDossierParent = d1.nomDossier

        LEFT JOIN Dossiers d2 ON d1.nomDossierParent = d2.nomDossier

        LEFT JOIN Dossiers d3 ON d2.nomDossierParent = d3.nomDossier ),

        UtilisateursEtPlaylists AS (

        SELECT u.idUser, u.pseudo, mp.idChanson, c.nomChanson, cp.chemin

        FROM Utilisateurs u

        JOIN MorceauPlusEcoute mp ON u.idUser = mp.idUser

                                AND mp.rnk = 1

        JOIN EtrePresent ep ON mp.idChanson = ep.idChanson

                        AND mp.idUser = ep.idUser

        JOIN Chansons c ON mp.idChanson = c.idChanson

        JOIN Playlists p ON ep.nomPlaylist = p.nomPlaylist

                        AND ep.idUser = p.idUser

        JOIN CheminPlaylists cp ON p.nomPlaylist = cp.nomPlaylist )

        SELECT idUser, pseudo, nomChanson, chemin

        FROM UtilisateursEtPlaylists

        ORDER BY idUser;
```

6. **<u>Pour chaque utilisateur premium, les morceaux partagés avec les utilisateurs amis.</u>**

```
CREATE VIEW vuePremiumMorceauxPartages AS

(SELECT pseudo, IdChanson

FROM Premium Prem JOIN Partager Part ON Prem.IdUtilisateur =
Prem.IdUtilisateur_A_Envoyé

WHERE Prem.IdUtilisateur = Part.idUtilisateur_A_Reçu

GROUP BY Prem.IdUtilisateur);
```

7. **<u>Le moment de la journée où il y a le plus d'écoutes (matin 6h - 12h, après-midi 12h - 18h, soir 18h - 24h, nuit 0h - 6h).</u>**

```
CREATE OR REPLACE FUNCTION get_moment_in_the_day (

        -- paramètre de la fonction

        current_date IN DATE

) RETURN VARCHAR2 -- indique le type de retour

        IS

        -- déclaration des variables

        current_hour NUMBER;

BEGIN

        -- Extraire l'heure de la date donnée en paramètre

        current_hour := EXTRACT(HOUR FROM current_date);



        -- Déterminer le moment de la journée en fonction de l'heure

        IF current_hour >=18 THEN

        RETURN 'soir';
        ELSIF current_hour >=12 THEN

        return 'après-midi';

        ELSIF current_hour >=6 THEN

        return 'matin';

        ELSE

        return 'nuit';

        end if;

END;



create or replace view momentEcoute (idChanson,idUser,heureEcoute,momentEcoute) as

select * , get_moment_in_the_day(e.DATEECOUTE)

from ECOUTER e;



create or replace view nbEcouteParMomentJournee(momentJournee,nbEcoute) as

select momentEcoute,count(momentEcoute)

from momentEcoute

group by momentEcoute;


create or replace view momentLePlusEcouter(momentJournee) as

select momentJournee

from nbEcouteParMomentJournee

where nbEcoute = (select max(nbEcoute)
            from nbEcouteParMomentJournee);
```

8. **<u>Pour chaque utilisateur, la note moyenne des évaluations de chacun des morceaux évalués.</u>**

```
CREATE VIEW vueEvalMoyenneChanson (pseudo,IdChanson, noteMoyenne) AS

(SELECT pseudo, IdChanson, AVG(note) AS noteMoyenne

FROM Noter N JOIN Utilisateurs U ON U.idUtilisateur = N.IdUtilisateur

GROUP BY U.IdUtilisateur, pseudo, IdChanson);
```

9. **<u>Pour chaque morceau, la médiane figure parmi les notes moyennes attribuées par chaque utilisateur (sur le morceau).</u>**

```
CREATE OR REPLACE VIEW VUE9 AS

WITH Moyennes_Utilisateurs AS (

        SELECT idChanson,idUser,AVG(note) AS moyenne_note

        FROM NOTER

        GROUP BY idChanson, idUser),

        Notes_Classees AS (

        SELECT idChanson, moyenne_note, ROW_NUMBER() OVER (PARTITION BY
idChanson ORDER BY moyenne_note) AS rn,

        COUNT(*) OVER (PARTITION BY idChanson) AS cnt

        FROM Moyennes_Utilisateurs

        )

SELECT

        idChanson,

        AVG(moyenne_note) AS mediane_note

        FROM Notes_Classees
        WHERE rn IN (FLOOR((cnt+1)/2), CEIL((cnt+1)/2))

        GROUP BY idChanson;
```

10. Le morceau qui a le plus grand écart type sur ses notes.

```
create or replace view StatsNoteChanson (idChanson,ecartType,noteMoyenne) as
select IDCHANSON , sqrt( (sum(power( n.NOTE - avg_Chanson,2)))/nb_User)
        ,avg_Chanson
        from (select IDCHANSON , count(IDUSER) as nb_User , avg(NOTE) as
avg_Chanson
        from NOTER
        group by IDCHANSON
        ) a
        join NOTER n on a.IDCHANSON = n.IDCHANSON
group by IDCHANSON;

create or replace view ChansonMaxEcartType(idChanson,nomChanson,ecartType) as
select idChanson,C2.NOMCHANSON,ecartType
from StatsNoteChanson
join CHANSONS C2 on StatsNoteChanson.idChanson = C2.IDCHANSON
where ecartType = (select max(ecartType) from StatsNoteChanson);
```