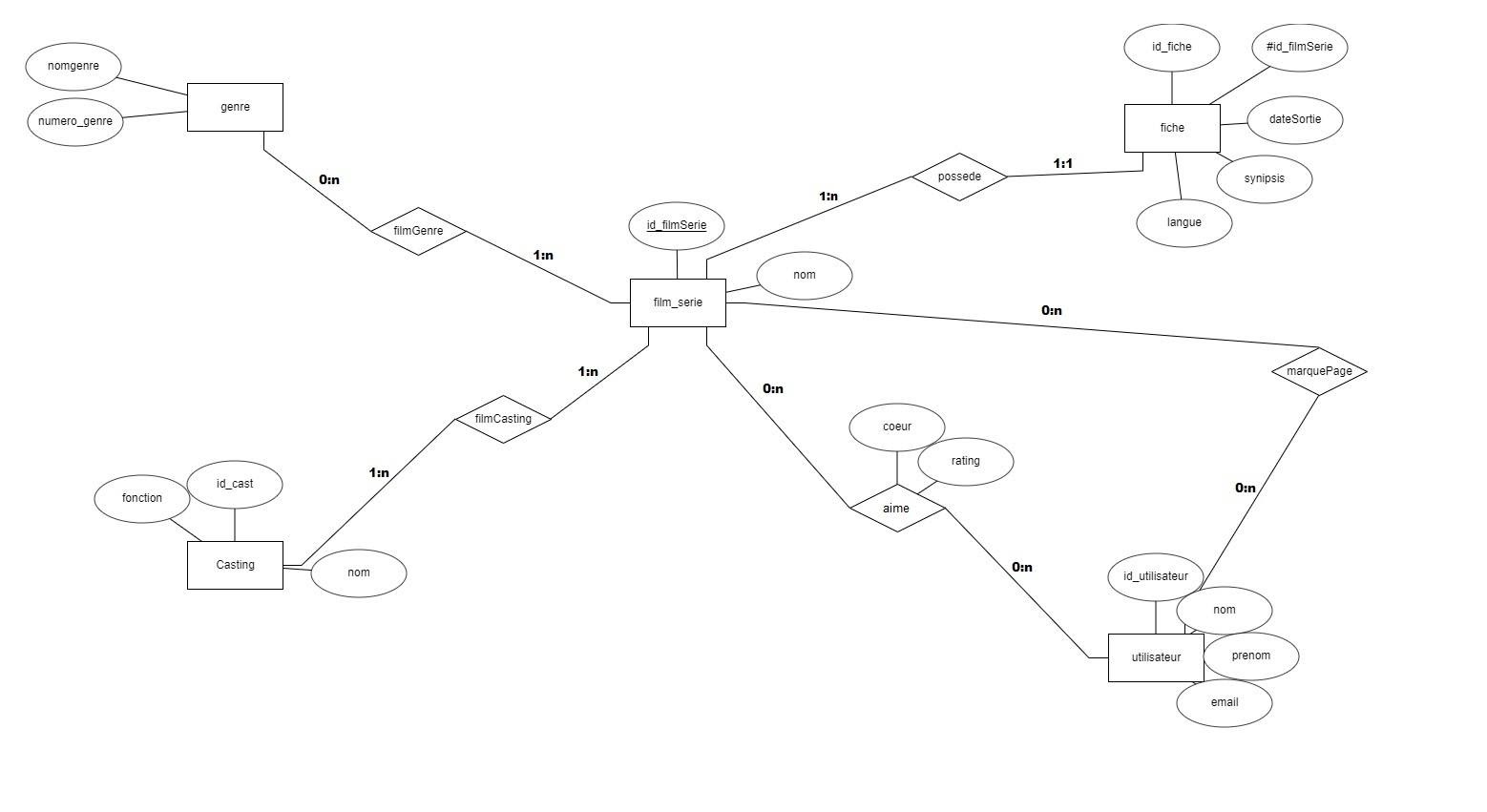
**RAPORT DE PROJET**

1. **Introduction**

Notre projet a pour but de cataloguer et d’organiser des films et des séries télévisées. Il a aussi pour but de stocker des utilisateurs et leurs préférences. Pour finalement pouvoir utiliser cette base de données pour supporter un site web d’archives de films, comme IMDb.

1. **Modele Entite Relation**

2.1. Le schema de la Base



Pour les besoins de la base de données, nous avions identifié les entitées suivantes:

* Le film/la série, avec un identifiant unique par film et un nom
* Chaque film possèderait deux fiches (une en français et l’autre en anglais) avec un identifiant unique, l’identifiant du film en question (clé étrangère), le synopsis, la date de sortie, et la langue.
* Des genres de films, avec un identifiant unique et un nom; chaque film pouvant posséder plusieurs genres (mais au moins un).
* Casting, qui listerait toutes les personnes impliquées dans un film, avec un identifiant unique, le nom ainsi que la fonction principale de la personne; un film peut logiquement avoir plusieurs membres de casting.
* Les utilisateurs, avec un identifiant, leurs noms, prénoms et adresse courriel. Les utilisateurs peuvent aimer un film, laisser une note, ou sauvegarder un film dans une liste de souhait (Marque page).

1. **Modele Relationnel**

3.1. Le schema du modele relationnel

film\_serie(id\_filmSerie, nom)

casting(id\_cast, nom, fonction)

filmCasting(#id\_filmSerie, #id\_cast)

genre(numero\_genre, nomgenre)

filmGenre(#id\_filmSerie, #id\_genre)

utilisateur(id utilisateur, nom, prenom, email)

jaime(#id\_filmSerie, #id\_utilisateur, coeur, rating)

marquePage(#id\_film, #id\_utilisateur)

fiche(id\_fiche, #id\_filmSerie, dateSortie, synopsis, langue)

Afin de pouvoir répondre à certaines requêtes, et vu les limitations dues à l’incapacité de pouvoir référencer plusieurs valeurs dans un attributs (par exemple énoncer tous les identifiants ou nom du casting), nous devions créer la table filmCasting et filmGenre; au lieu que le Casting et le Genre soient des attributs multivalués de la relation fiche.

filmCasting aurait une entrée par participation à un film (par exemple Quentin Tarantino serait deux uplets de cette table pour Pulp Fiction; une fois comme réalisateur et une fois comme acteur), et filmGenre aurait une entrée par genre de film (Seigneur des Anneaux ayant deux uplets, une fois pour le genre “Guerre” et l’autre pour “Fantaisie”). Ces tables nous permettent de faire les liens pour les requêtes qui seront énoncées à la section 6.

Pour les mêmes raisons, nous avons dû avoir les associations jaime et marquePage comme relations dans notre schéma, afin de pouvoir faire le lien entre un utilisateur et la note laissée sur un film, ou son ajout du film à une liste de souhait.

Autre que cela, le modèle relationnel reprenait les entités du modèle entité-association.

3.2. La normalisation

Toutes les relations de notre modèle sont en première forme normale vu que le domaine de chaque attribut est simple.

Toutes les relations de notre modèle sont aussi en deuxième forme normale:

film\_serie(id\_filmSerie, nom)  
*DFs: S ={(id\_filmSerie) →  nom}* (il pourrait y avoir deux films différents mais du même nom)

casting(id\_cast, nom, fonction)

*DFs: S ={(id\_cast) →  nom, (id\_cast) → fonction} (il peut y avoir deux acteurs/techniciens du mee nom)*

filmCasting(#id\_filmSerie, #id\_cast)

*DFs: s.o* Car un film ne détermine pas automatiquement son cast.

genre(numero\_genre, nomgenre)

*DFs: S={(numero\_genre) → nomgenre)}* théoriquement la même idée que pour film\_serie

filmGenre(#id\_filmSerie, #id\_genre)

*DFs: s.o.* Car un film ne détermine pas nécessairement un genre

utilisateur(id utilisateur, nom, prenom, email)

*DFs: S={(id\_utilisateur) → nom, (id\_utilisateur) → prenom, (id\_utilisateur) → email}*

jaime(#id\_filmSerie, #id\_utilisateur, coeur, rating)

*DFs: s.o.* Un coeur ne décide pas l'appréciation mesurée de l'utilisateur

marquePage(#id\_film, #id\_utilisateur)

*DFs: s.o.* Chaque utilisateur peut avoir une liste différente de films.

fiche(id\_fiche, #id\_filmSerie, dateSortie, synopsis, langue)

*DFs: S={(id\_fiche) → #id\_filmSerie, (id\_fiche) → synopsis, (id\_fiche) → langue, #id\_filmSerie → dateSortie)}*

On peut voir que tout attribut n'appartenant pas à la clé, ne dépend pas d'une partie de la clé.

Toutes les relations qui possèdent des dépendances fonctionnelles (film\_serie, casting, genre, utilisateur) sauf fiche sont en 3eme forme normale, avec #id\_filmSérie qui détermine la date de sortie d’un film sans pour autant être une clé de la relation (le même #id\_filmSerie apparaissant deux fois pour chaque film).

Les relations film\_serie, casting, genre et utilisateur sont aussi en FNBC vu que toutes les dépendances proviennent de la supercle de la relation. La relation fiche n’est pas en FNBC car #id\_filmSerie détermine la date de sortie sans être super clé.

1. **Langage de Definition de Donnee d’Oracle**

*4.1. Langage de Définition de Données d'Oracle*

A l’aide du langage SQL, on creer les tables suivantes:

* **film\_serie**( id\_filmSerie, nom)  le ID du film/serie est entier non null (INTEGER NOT NULL)  et son nom est une chaîne de caractères de maximum 50 caractères (VARCHAR(50)). La clé primaire est le ID du film/serie
* **Genre**(numero\_genre, nomgenre) le numero du genre est un entier non null et son nom est son nom est une chaîne de caractères de maximum 20 caractères. La clé primaire est le numéro du genre
* **filmGenre**(id\_filmSerie, id\_genre) cette table relie film\_serie et Genre par deux clés étrangères: id\_filmSerie et id\_genre (qui est le même que numero\_genre dans Genre). Il associe a chaque film ses genres.
* **casting**(id\_cast, nom, fonction) le ID de la personne est un entier non null, son nom est une chaîne de caractères de maximum 30 caractères et sa fonction est une chaîne de caractères de maximum 20 caractères. La clé primaire est le ID de la personne.
* **filmCasting**(id\_filmSerie, id\_cast, fonction) cette table connect film\_serie et casting donc elle contient deux clés étrangères (FOREIGN KEY): id\_filmSerie et id\_cast. Sa troisième colonne donne la fonction de la personne mentionnée dans le film spécifiée.
* **utilisateur**(id\_utilisateur, nom, prénom, email) le ID de l’utilisateur est un entier non null, son nom est une chaîne de maximum 15 caractères, son prénom est une chaîne de maximum 15 caractères et son email est une chaine de maximum 30 caractères. La cle primaire est le ID de l’utilisateur.
* **jaime**(id\_utilisateur, id\_filmSerie, coeur, rating) les deux id sont des cles etrangeres qui aident a créer un lien entre un utilisateur et un film. Cet utilisateur peut donc mettre un coeur, qui est un entier égal a 0 ou 1 (CHECK (coeur IN(0,1)) ). Ou même lui donner une note, qui est un entier entre 0 et 5

(CHECK (rating IN(0,1,2,3,4,5)) ).

* **marquePage**(id\_filmSerie, id\_utilisateur) cette table sert a créer une liste de films ou séries que chaque utilisateur aimerais regarder plus tard. On prend donc deux clés étrangères vus précédemment: id\_utilisateur et id\_filmSerie.
* **fiche**(id\_fiche, id\_filmSerie, dateSortie, synopsis, langue) le ID de chaque fiche est un entier non null, il prend ensuite la clé étrangere id\_filmSerie pour qu’on sache a quel filme appartient la fiche. Il a ensuite un synopsis, qui est une chaîne de maximum 350 caractères qui résume le film ou la série. Finalement, la colonne langue spécifie la langue de la fiche (‘English’ ou ‘Francais’). La langue du synopsis dépend donc de cette colonne. La clé primaire de cette table est le ID de la fiche.

*4.2. Chargement de la base*

A l’aide d’insertions dans les tables, on a put remplir toute nos tables avec au moins 50 lignes de valeurs chaque. Afin de pouvoir les utilisés pour executer des requetes SQL. Par exemple,  “INSERT INTO jaime (id\_utilisateur, id\_filmserie, coeur, rating) VALUES ( 001, 25, 1, 4)”. Cette insertion ajouté a la table jaime ’information que le premier utilisateur aime le 25e film (Parasite) et le note 4 / 5.

1. **Requete SQL**

5.1 Quel film est sorti en 1975 avec comme acteur Jack Nicholson?

Cette requête complexe utilise 4 relations (casting → filmCasting → film\_serie → fiche). En exécutant deux jointures, dont une en sous-requêtes, nous avons pu isoler le nom ‘Jack Nicholson’ et la date de sortie du film et faire une projection sur le nom du film cherché.

5.2 Quel sont tous les genres que le réalisateur Christopher Nolan a interprété?

Dans cette requête complexe de 5 relations (casting → filmCasting → film\_serie → filmGenre → genre), nous avons exécuté trois jointures et une sous-requête qui isole le nom ‘Chrisopher Nolan’ pour ensuite faire une projection distincte sur les genres associés à ce réalisateur.

5.3 Dans quel(s) film(s)/série(s) a participé Al Pacino et quelle est leurs dates de sorties?

À l’aide de cette requête complexe de 4 relations (casting → filmCasting → film\_serie → fiche), nous avons fait deux jointures et une sous-requête pour isoler le nom ‘Al Pacino’ de la table casting et faire une projection sur les noms des films et la date de sortie associées à Al Pacino.

5.4 Quels sont les films (avec leur synopsis en anglais) qui ont été aimé par l’utilisatrice Lindsay Logan?

Avec cette requête complexe de 4 relations (utilisateur → jaime → film\_serie → fiche), nous avons fait deux jointures et trois conditions à respecter de la table utilisateur. Par la suite nous avons fait une projection sur le nom de films aimé par Lindsay Logan et leur synopsis en anglais en incluant une condition sur la langue du synopsis.

5.5 Quel sont les genres des films en marque pages de l’utilisateur qui possède le email [Rfranklin@gmail.com](mailto:Rfranklin@gmail.com)?

Cette requête complexe de 5 relations (utilisateur → marquePage → film\_serie → filmGenre → genre) permet d’isoler une simple occurrence des genres associés aux films que l’utilisateur au email [Rfranklin@gmail.com](mailto:Rfranklin@gmail.com) a mit dans sa liste de souhaits. Nous avons utilisé trois jointures, une sous-requête et une projection distincte pour arriver au résultat voulu.

5.6 Quel est la note moyenne du film ‘Parasite’?

Dans cette requête simple de 2 relations (film\_serie → jaime, nous avons fait une projection sur la moyenne des notes attribuées au film du nom ‘Parasite’. Pour ce faire, une simple jointure entre la table film\_serie et jaime, en isolant le nom ‘Parasite’ a suffit.

5.7 Quels sont les noms des films qui font partie du genre Guerre?

À l’aide de cette requête simple de 3 relations (film\_serie → filmGenre → genre), nous avons fait une restriction sur la genre ‘Guerre’ pour ensuite faire deux jointures reliant les 3 tables. Finalement, la requête exécute une projection sur le nom des films se rattachant au genre désiré.

5.8 Quels sont les acteurs qui ont joué dans des films d’Action?

Dans cette requête complexe de 5 relations (casting → filmCasting → film\_serie → filmGenre → genre), nous la fait une restriction sur le genre ‘Action’ et 3 jointures reliant toutes les tables pour ensuite faire une projection distincte sur le nom des acteurs associés à ce genre.

5.9 Quels films sont sortie entre 1990 et 2000?

À l’intérieur de cette requête simple de 2 relations (film\_serie, fiche) nous avons fait une restriction sur les films sorties entre 1990/01/01 et 2000/12/31 pour ensuite joindre les deux tables et faire une projection sur le nom de films sortie dans cette intervalle.

* 1. fficher le synopsis français de la trilogie “Seigneur des Anneaux”?

Avec cette requête simple de 2 relations (film\_serie, fiche), nous avons fait une restriction sur les noms des films qui commence par les caractères ‘The Lord of the Rings’ pour isoler la trilogie pour ensuite faire une jointure entre les deux tables et une projections sur le synopsis où la langue est égale à ‘Français’.

**6. PL-SQL**

Pour notre section PL/SQL, nous cherchions à implémenter un bloc d’instructions, incluant un curseur, nous permettant d’entrer un nom de la table casting (soit un acteur, réalisateur ou écrivain) pour ressortir chaque film dans lesquelles il/elle est impliqué. Une variable ‘compteur’ est aussi déclarer pour compter le nombre d'occurrences de ce membre du casting.

Nous avons également créé un deuxième bloc d’instructions incluant un trigger pour relevé un message dans le output nous informant du nombre de tuple de la table qui a été modifier. Ce trigger apparaît lors de  l’insertion ou l’effacement d’un tuple.

* 1. **Application**

Notre groupe n’a pas pu compléter une application mais voici ce qu’on comptait faire. L’enregistrement d’un utilisateur serait possible en prenant son nom, prénom et email. Cet utilisateur pourrait ensuite ajouter des films, et les modifier s’il a les permissions.

 Sinon, il pourrait créer des listes de souhaits et marquer les films qu’il souhaiterait voir. Il pourrait également noter les films de 0 a 5 et marquer ses favoris par un coeur.

Les films pourront être organisées par genre, date de sortie, directeur ou même par acteur. On pourrait même  simplement montrer la filmographie d’un directeur ou d’un acteur.

1. **Conclusion**

Ce projet nous a permit de créer une base de données de toutes pièces et d’observer toute la matière du cours dans son application. Une base de donnée sur les films et séries de télévision a plusieurs fonctions intéressantes lorsqu’on y intègre une communauté d’utilisateur. Avec leurs interaction, la base peut compiler des données pertinentes sur chaque utilisateur pour ensuite produire des suggestions de films. Sommes toute, lors de la création de la base, nous avons pris conscience du potentiel immense qu’une telle base peut fournir en terme de donnée et de personnalisation des recherches pour chaque utilisateur différent.

1. **Annexe**

9.1 Annexe 1 : LDD pour la création des tables **(films\_series.sql)**

9.2 Annexe 2 : Script complet des requêtes SQL **(LMD.sql)**

9.3 Annexe 3 : Script complet du PL/SQL **(PLSQL.sql)**