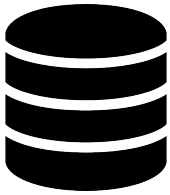
Rapport de Projet BDD2
Site de Replay



Marko MARTIC

Octobre 2020

Table des matières

Chapitre 1

Introduction

1.1 Présentation

Dans le cadre de l'unité d'enseignement Bases de Données 2 en L3 d'Informatique à l'Université de Strasbourg, nous devons réaliser une base de données d'un site de replay de vidéos, permettant aux utilisateurs de mettre en favoris des vidéos, de consulter leur historique, d'avoir des suggestions de vidéos selon la popularité de ces dernières, de l'intérêt que l'utilisateur est susceptible de leur porter etc...

Dans mon cas j'ai décidé de réaliser le projet sous ORACLE.

1.2 Get started

Ce projet se présente sous plusieurs fichier :

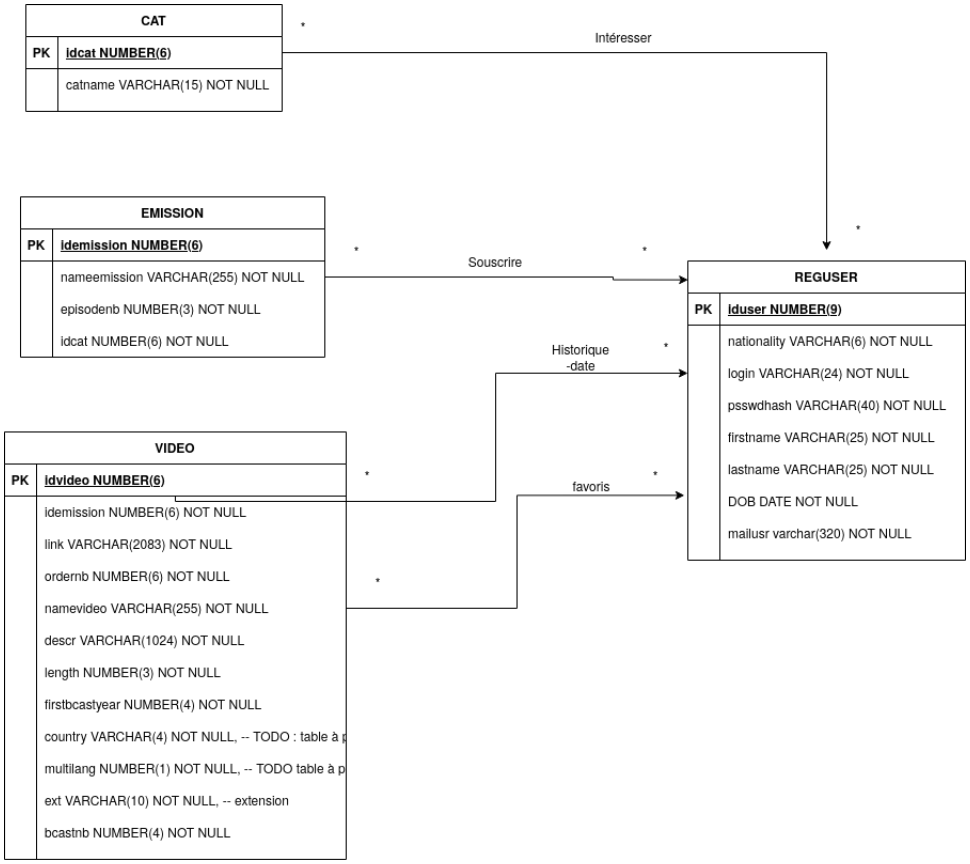
- *create.sql* : fichier faisant la création des tables, vues, séquences et triggers d'auto-incréments
- *fill.sql* : fichier permettant de remplir la base de données ORACLE avec des données permettant de tester le bon fonctionnement du système
- *destroy.sql* : fichier permettant de détruire la base de données (tables, vues etc.)
- *request.sql*: fichier regroupant toutes les requêtes demandées dans le sujet.
- *Rapport.pdf*: ce-même compte rendu

DISCLAIMERS :

- Les données introduites ne sont pas forcément cohérentes, par exemple (entre autres), ne sachant pas à quelle date le projet allait être corrigé, et voulant m'assurer que les requêtes faisant intervenir sysdate produisent toujours des résultats, j'ai enregistré des vues de vidéos pour les mois à venir, jusqu'en janvier 2021, pour être large.

- le nb maxi de vues/minute pour les utilisateurs enregistrés sera géré par le PL/SQL dans la deuxième partie, pour les utilisateurs non-enregistrés, ce sera à la couche applicative de s'en occuper.

1.3 Schémas



(le schéma UML est plus complet)

Figure 1.1: schéma entité-association non finit

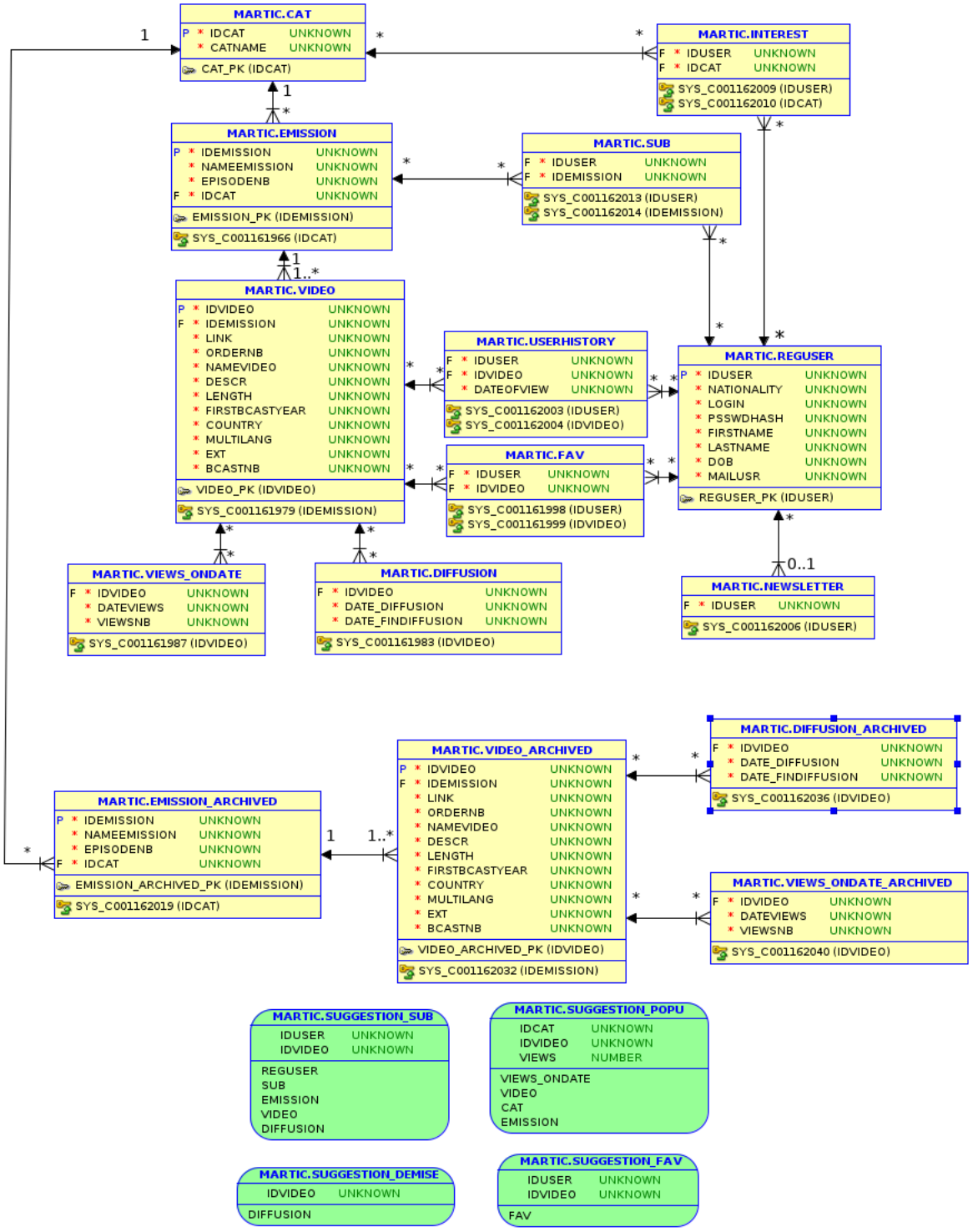


Figure 1.2: schéma UML (étoile rouge = NOT NULL)

Chapitre 2

Explications

Ici seront détaillés les différents points qui méritent que l'on s'y attarde !

2.1 Champs intéressants

Le champ episodenb de la table emission désigne le nombre d'épisodes de l'émission, il y en a au moins 1 sinon l'émission n'existerait pas.

- Le champ ordernb de la table video, désigne l'ordre de la vidéo dans l'émission. ordernb de la Video ne devra jamais dépasser episodenb de l'émission associée.
- Le champ multilang de la table video, est un booléen. S'il vaut 1, alors le lecteur de vidéo au niveau de l'applicatif fera le nécessaire pour trouver la piste audio correspondante à la langue que souhaite l'utilisateur, la BDD ne s'en préoccupe pas.
- Le champ bcastnb de la table video doit valoir count(date_diffusion) group by idvideo de la table diffusion.
- La table diffusion contient un champ date_findiffusion, afin de savoir quand proposer

TODO : mettre un déclencheur sur l'insertion d'une nouvelle diffusion, il ne faut pas que la date_findiffusion-date-diffusion<7 (pour respecter le sujet)

2.2 Tables intéressantes

Pour s'occuper des vues des vidéos, il y a un peu de redondance, et pour être plus efficace, on pourrait en créer plus, mais ce serait à voir en fonction des performances obtenues/voulues une fois la BDD réellement mise en place.

Pour gérer les vues, deux tables ont été mises en place :

- create table VIEWS_ONDATE
  (idvideo NUMBER(6) NOT NULL,
  dateviews DATE NOT NULL,
  viewsnb NUMBER(9) NOT NULL,
  FOREIGN KEY (idvideo) REFERENCES VIDEO);

Cette table enregistre le nombre de vues par jour, à chaque nouvelle vue, le views_nb de la date en cours s'incrémente de 1.

La somme des vues d'une vidéo s'obtient avec un sum(viewsnb) group by idvideo, sachant que c'est une opération qui sera fréquente, il peut être intéressant d'ajouter un champ total_views, même si cela créé de la redondance.

- create table USERHISTORY
  (iduser NUMBER(6) NOT NULL,
  idvideo NUMBER(6) NOT NULL,
  dateofview DATE NOT NULL,
  FOREIGN KEY (iduser) REFERENCES REGUSER,
  FOREIGN KEY (idvideo) REFERENCES VIDEO);

Ceci est la deuxième table gérant les vues des vidéos, il y a un peu de redondance mais cela favorise la cohérance et me semble-t-il la performance.

La newsletter n'est pas encore gérée.

2.3 Vues

Il y a 4 différentes vues, avec une de plus qui pourrait éventuellement avoir lieu :

- SUGGESTION_SUB : suggère les vidéos à l'utilisateur venant des émissions auxquelles il est abonné, qui datent de moins d'une semaine.
- SUGGESTION_DEMISE : suggère les vidéos qui disparaîtront prochainement.
- SUGGESTION_FAV : suggère vidéos mises en favoris par l'utilisateur.
- SUGGESTION_POPU : suggère les vidéos les plus vues dans les deux dernières semaines.

Une cinquième vue pourrait avoir lieu, car elle collerait au sujet, c'est celle qui sélectionne l'historique de l'utilisateur.

2.4 Archivage

J'ai décidé d'avoir plusieurs tables d'archivages correspondant aux différentes tables liées aux vidéos, comme emission ainsi que les 2 tables gérant les vues des vidéos.

Elles n'ont pas été remplies car aucune requête n'y faisait référence...

Quand toutes les vidéos d'une émissions sont archivées, l'émission est archivée également.

Une vidéo archivée n'est plus disponible par le site de replay, donc si un utilisateur l'avait mis en favoris, elle serait inaccessible.

Chapitre 3

Remplissage de la BDD

3.1 Départ lent

Pour commencer le remplissage, j'avais créé à la main les catégories, émissions ainsi qu'une partie des vidéos à l'aide notamment de plugin firefox webscraper, qui permettait de sélectionner des div de façon automatique et d'en faire l'export, avec python j'ai pu faire les requêtes avec quelques champs qui variaient et c'étaient ceux venant du plugin.

L'apparence du plugin :

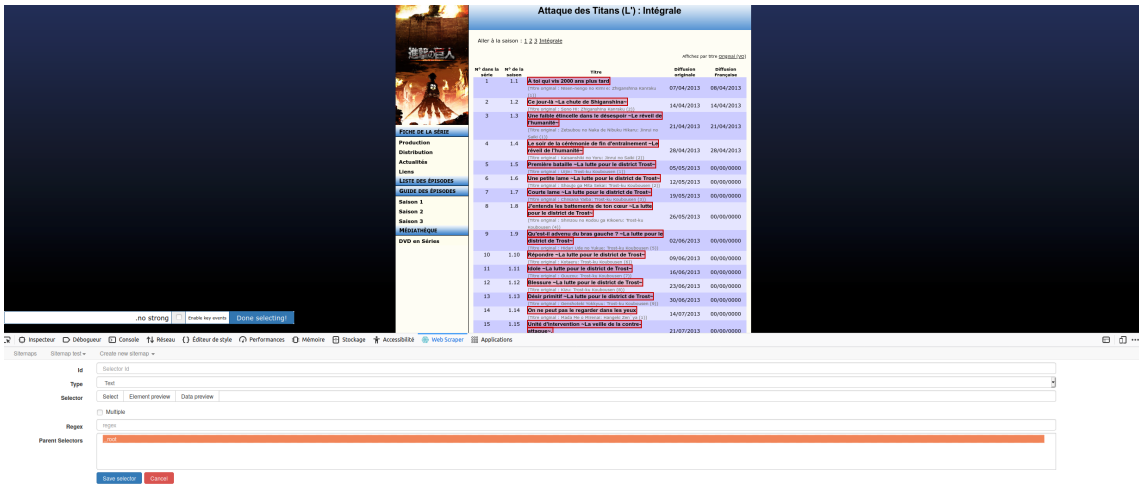


FIGURE 3.1 : plugin webscraper

3.2 Suite plus aisée

Le siteweb https ://www.generatedata.com/ a fait la suite du travail, plutôt bien. A l'exception près des dates, le site exportait les dates sous la forme '2020/09/01' alors que ce qu'il fallait c'est TO_DATE('2020/09/01', 'yyyy/mm/dd').

VSCode a une fonction de remplacement bien pratique, par regex :

Remplacement de : ('\d+/\d+/\d+') par TO_DATE($1, 'yyyy/mm/dd') et c'était réglé !

Chapitre 4

Requêtes

4.1 Premières requêtes

RAS

4.2 Requête 5

Pour cette requête je n'ai pas réussi à faire ce que je souhaitais, je voulais faire un produit cartésien entre les vidéos des différents utilisateurs, et faire une UNION de ces différentes sous-tables :

Explication (avec table simplifiée, la date a été enlevée pour l'exemple) :

Les 10 couples de vidéos apparaissant le plus souvent simultanément dans un historique de visionnage d'utilisateur.

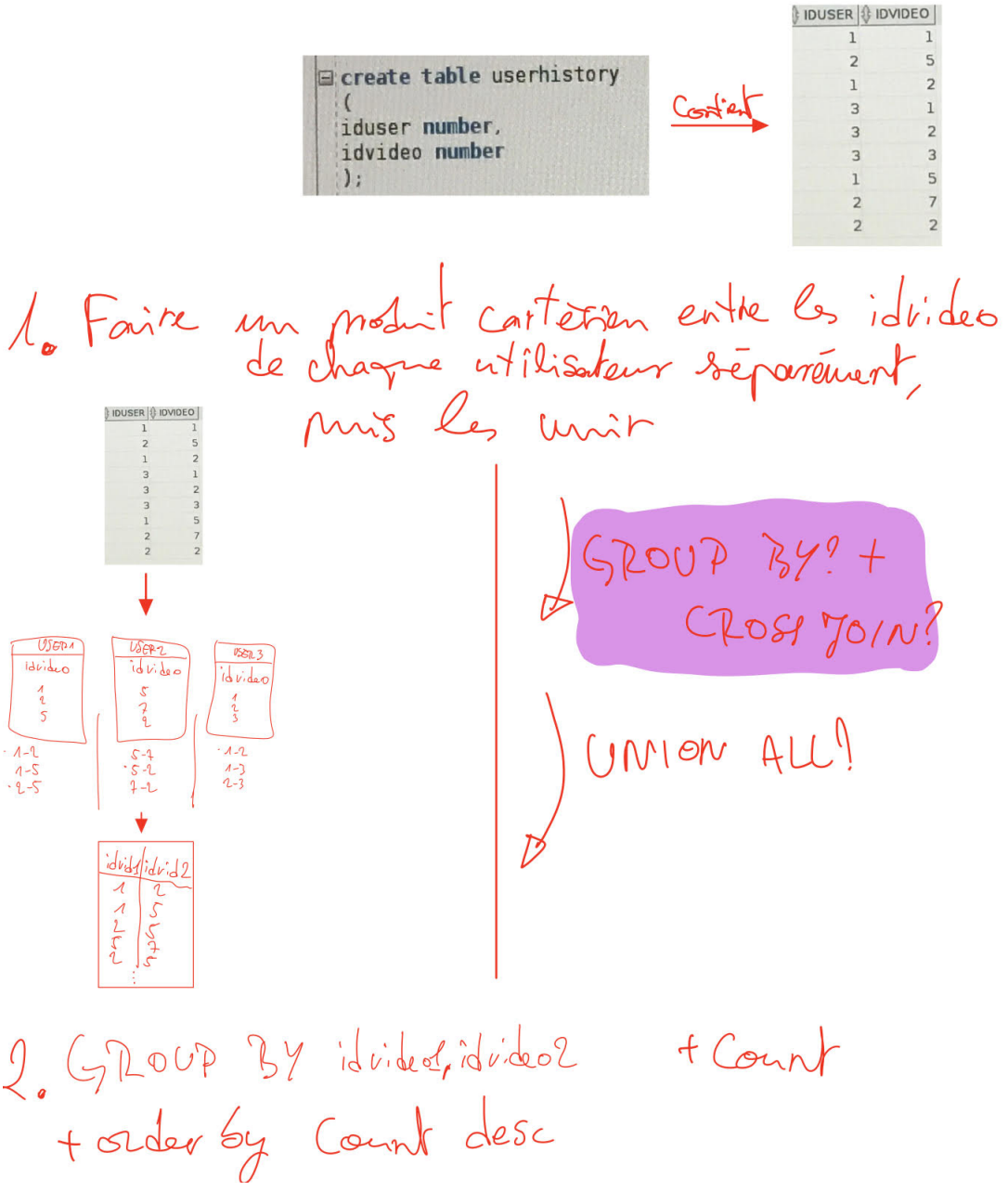


FIGURE 4.1 : requête 5 problème en violet non résolu

A la place de faire un produit cartésien comme dit précédemment, j'ai fait finalement un produit cartésien entre l'user-history sur lui-même, avec quelques conditions de filtrage, pour retomber sur le résultat voulu, ce qui est largement moins efficace que la première version que je voulais faire...mais ça a l'air de bien fonctionner malgré tout !

Chapitre 5

Conclusion

Le travail effectué a permis de me mettre à jour sur la réalisation de schémas (que j'ai normalement compris mais pas finalisé par manque de temps), la modélisation ainsi que le SQL en général, et notamment les jointures.