Location Vidéo

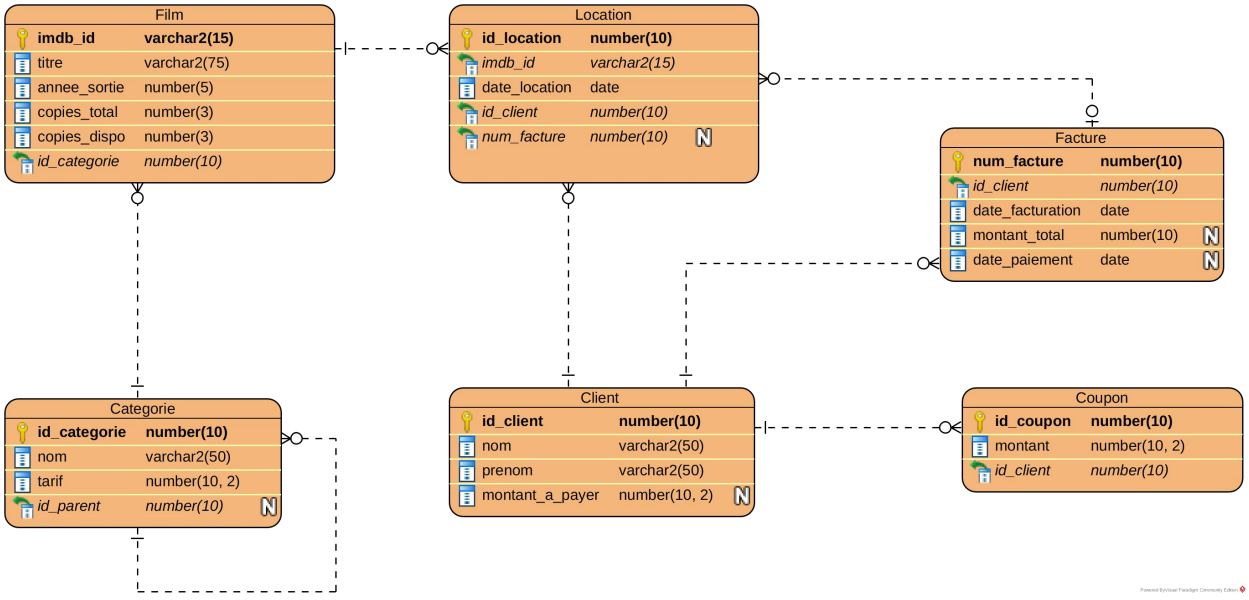
1. Présentation générale du laboratoire

Ce troisième laboratoire, d'une durée de deux semaines, vise à mettre en pratique vos apprentissages des extensions procédurales de Oracle (PL/SQL).

Vous disposez d'une base de données de locations vidéo et dont le LDD vous est fourni. Vous devez étendre cette base de données en programmant des procédures et des fonctions qui automatisent des traitements. La section suivante présente le modèle relationnel de la base de données. La section 3 énumère l'ensemble des fonctions et des procédures à programmer (et à tester!!).

Ce laboratoire doit être réalisé en groupes, en préservant les même groupes que le premier laboratoire (à moins qu'un changement ait été autorisé par votre enseignant-e).

2. Modèle relationnel



3. Procédures et fonctions à programmer:

(*) Vous pouvez répondre à cette question dès la semaine 1 du laboratoire.

Question 1 *:
Écrire un déclencheur *TRG_verifier_impayes_avant_location* qui empêche la création d'une nouvelle location si un client un montant impayé de plus de 50$. Si ce montant excède 25$, il doit être permis au client de louer mais un message d'alerte doit s'afficher dans la console (ex. : Attention, le client a des factures impayées). Les factures impayées sont celles dont la date de paiement est nulle (NULL).

Important: N'utilisez pas la colonne montant_a_payer de la table Client dans ce Trigger.

Question 2 *:
2.1 *: Écrire la procédure *p_Calculer_Montant_*Du qui met à jour la colonne dérivée *montant_total_a_payer* de la table Client, pour le client dont l'identifiant est passé en paramètre. Le montant total à payer est calculé en additionnant les montants de toutes les factures d'un client qui n'ont pas été payées (date_paiement est NULL).
2.2 : Écrire la procédure p_Calculer_Tous_Montants_Dus qui met à jour le *montant_total_a_payer* pour tous les clients.

Question 3 *:
Écrire la fonction f_Montant_Facture qui calcule et retourne le montant total d'une facture dont l'identifiant est reçu en paramètre.

Le montant d'une facture est calculé en se basant sur le prix de location. Le prix de location dépend de la catégorie du film loué (colonne *tarif* de la table *Categorie*).

Question 4:
Écrire la procédure *p_Emettre_Factures* qui génère les factures pour les locations qui n'ont pas été facturées (i.e. celles où les *numFacture* sont NULL). Vous devez générer une seule facture par client.

Vous remarquerez que, dans le script de définition des tables, les numéros de factures sont créés automatiquement par un déclencheur. Ce déclencheur utilise une séquence qui vous servira à récupérer le numéro de la dernière facture insérée (voir *currval*).

Voici l'algorithme que vous devez implémenter:
Pour tous les clients ayant des Locations non facturées **Faire:**
    Pour toutes les locations non facturées du client **Faire:**
        Si on n'a pas encore créé la facture Alors:
            Créer une nouvelle facture
            Récupérer le numéro de la facture créée
        Fin Si
        Mettre à jour la location avec le nouveau numéro de facture
    Fin Pour
Fin Pour

Question 5:
Écrire la procédure *p_Generer_Coupons* qui ajoute des coupons dans la table Coupon pour les ***n***-clients n'ayant pas de coupons, choisis aléatoirement. Le nombre ***n*** de clients ainsi que le montant des coupons sont passés en paramètre.

Note 1: Pour obtenir un client aléatoirement, il suffit de trier les clients aléatoirement avec la fonction *DBMS_RANDOM.value* : ORDER BY *DBMS_RANDOM.value*

Question 6 *:
6.1. Écrire la fonction *f_Nb_Films_Dans_Cat* qui retourne le nombre de films classés dans une catégorie donnée.
6.2. En utilisant cette fonction, écrire la requête SQL permettant d'afficher les noms des catégories, classées selon leur nombre de films.

Question 7 *:
Écrire la procédure *p_Afficher_Cats_Parentes* permettant d'afficher le nom de toutes les catégories parentes (une par ligne) de la catégorie dont l'identifiant est passé en paramètre.

Question 8:
Écrire la fonction *f_Est_Sous_Categorie* qui reçoit en argument deux identifiants de catégories *id_cat_enfant* et *id_cat_parent*. Votre fonction doit retourner la valeur 'O' si la catégorie ayant pour identifiant *id_cat_enfant* est un descendant de la catégorie *id_cat_parent, ou 'N' sinon*.

Question 9:
Écrire la fonction *f_Categorie_Plus_Populaire* qui reçoit en argument l'identifiant d'un client. Votre fonction doit retourner l'identifiant d'une catégorie principale qui est celle qui regroupe le plus grand nombre de films qui ont été loués par le client, en incluant ses sous-catégories. Nous entendons par "catégorie principale" une catégorie qui n'a pas de catégorie parente.

Note: Il serait judicieux de créer une autre fonction pour implémenter cette fonction.

Question 10:
En utilisant la fonction *f_Categorie_Plus_Populaire*, écrivez une requête SQL permettant d'afficher les catégories principales par ordre de popularité. La catégorie la plus populaire est celle est qui la plus populaire chez le plus grand nombre de clients.

4. Schéma et jeu de données:

Le schéma de la base de données vous est fourni sur Moodle - Il vous suffit de l'exécuter dans SQL Developer en vous connectant dans votre compte.

De plus, vous disposez d'un jeu de données fictif que vous pouvez importer également afin de tester vos procédures et fonctions. Ce jeu de données n'a pas la prétention d'être complet pour réaliser tous les tests possibles, vous pouvez le compléter au besoin.

5. Livrable et remise finale:

Vous devez remettre un fichier à l'extension **.sql** comprenant la définition de l'ensemble de vos procédures, fonctions et requêtes. Vos procédures et fonctions doivent respecter les normes de programmation qui vous ont été présentées.

La remise doit être effectuée sur Moodle durant la semaine du 25 mars 2019 (voir date précise selon votre groupe sur Moodle). En respect des règles du Service des Enseignements Généraux, toute remise tardive se verra attribuer la note de 0 (à moins d'entente justifiée et préalable avec l'enseignant-e).

Bon travail ☺