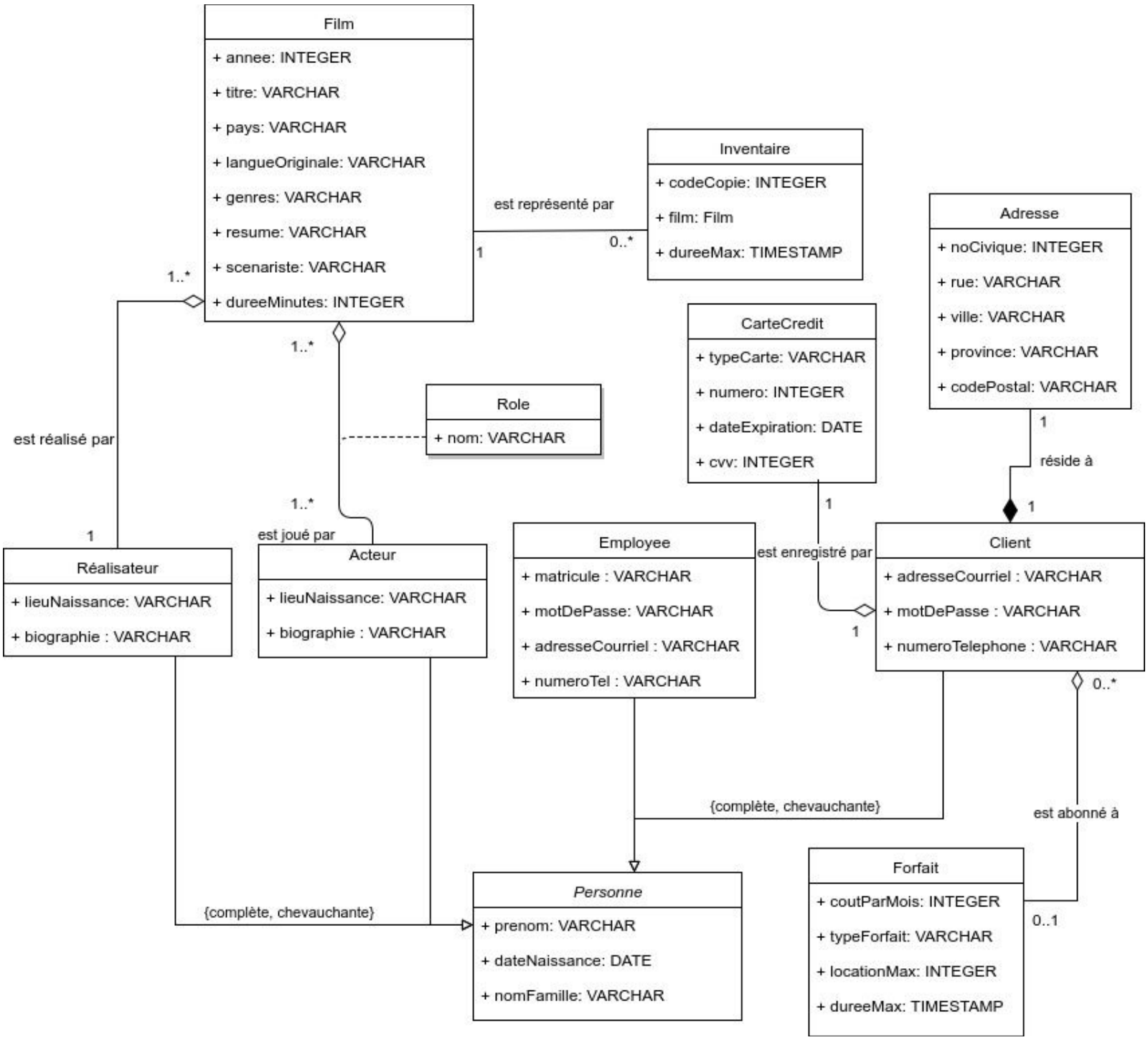
Laboratoire 1 : Conception du schéma relationnel

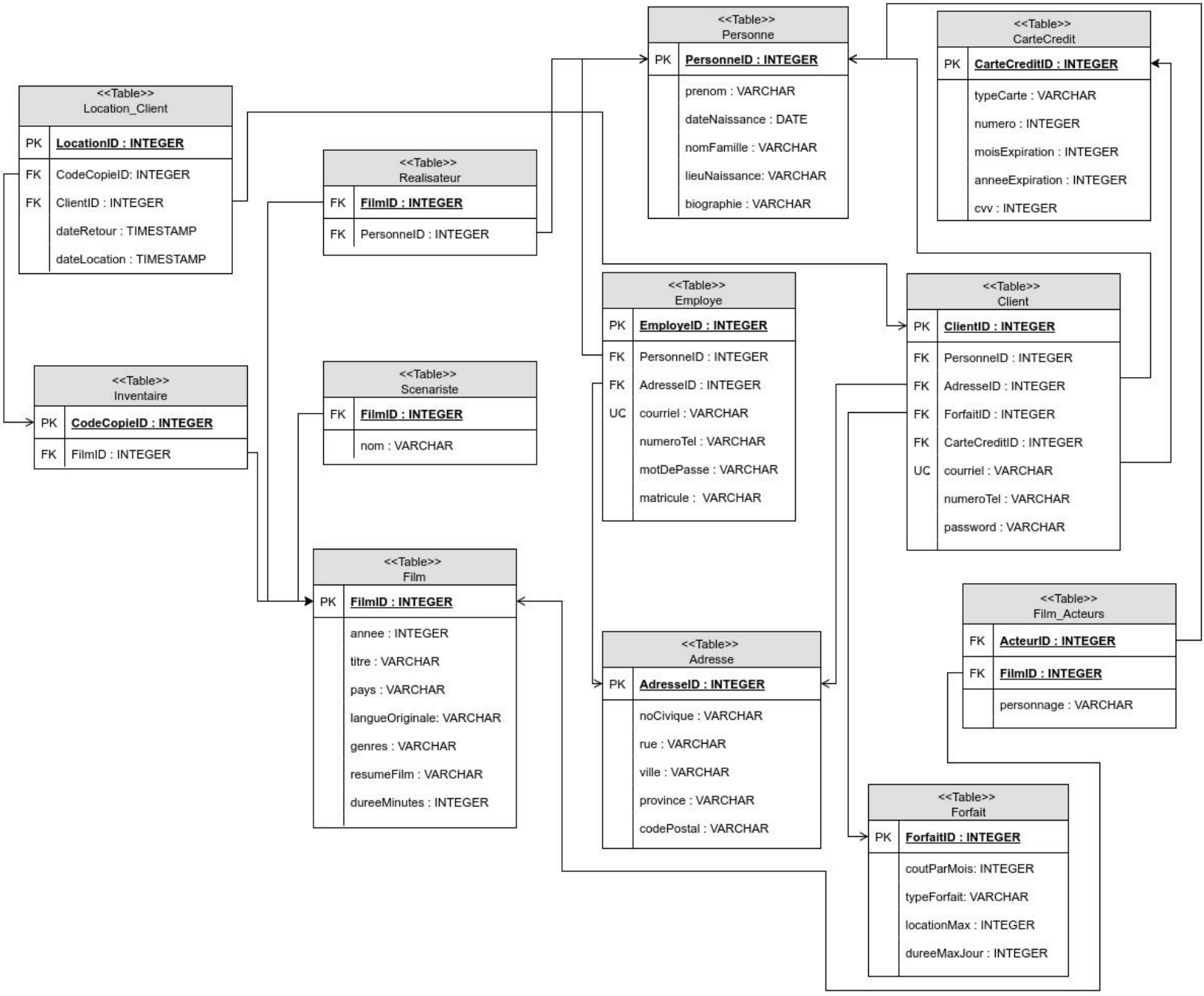
Cours	LOG660 – Base de données haute performance
Session	Été 2017
Nº de laboratoire	1
Groupe	4
Étudiant 1 (nom et CP)	Marc-André Tremblay (TREM23069203)
Étudiant 2 (nom et CP)	Mandaniaina Raharison (RAHM20069506)
Étudiant 3 (nom et CP)	Samuel Lamoureux (LAMS05069509)
Étudiant 4 (nom et CP)	Alexandre Godard (GODA28049507)
Professeur	Lévis Thériault
Chargé de laboratoire	Richard Rail
Date	29 Mai 2017

Rapport	
Schéma conceptuel	**/ 10**
Schéma relationnel	**/ 15**
Justifications	**/ 10**
Contraintes	**/ 15**
Procédures	**/ 10**
Question 1	**/ 5**
Question 2	**/ 5**
Qualité du français	
Code source	
Script de création de tables	**/ 10**
Programme d'insertion Java	**/ 10**
Qualité du code	
Correction interactive	**/ 10**
Total	**/ 100**

1 Schéma conceptuel



2 Schéma relationnel



3 Justification des choix de conception

Pour les pays nous avons choisi un VARCHAR à la place d'un ENUM pour faciliter la recherche.

Pour les éléments qui risquaient d'être réutilisées à plusieurs endroits mais qui allait jamais changer, (genres, langue original du film, pays) nous avons décidé de ne pas créer de nouvelle table vu qu'il n'y aurait qu'un attribut (le nom) et qu'il est seulement utilisé dans une table.

Afin d'éviter l'utilisation de TRIGGER pour la mise à jour de l'inventaire, nous avons simplement une ligne dans la table Inventaire pour chaque copie d'un film. Pour savoir s'il reste des copies disponibles pour film, nous faisons une requête pour avoir les CopiesID du film désirée, et nous allons chercher ensuite dans la table Location_Client la première copie qui a été retournée mais pas encore louée (dateRetour != null)

Nous avons économisé la création d'une table pour les acteurs en ajoutant simplement les attributs lieuNaissance et biographie à la table Personne. Ainsi, dans le cas ou un Acteur est à la fois un Scenariste, les deux auront la même biographie et le même lieu de naissance.

Le mapping entre l'adresse et le client est 1 à 1, car s'il y avait deux clients résidants à la même adresse et qu'un décide de déménager, il est plus simple de simplement changer son adresse que de créer une nouvelle ligne en BD et de changer le lien existant.

4 Conventions de nommage

Nous utilisons la convention *lowerCamelCase* pour nos noms d'attributs. Les noms de clés primaires et étrangères suivent la convention *UpperCamelCase* suivi de "ID", les deux lettres en majuscules.

Le nom des tables suit le format *UpperCamelCase*. Les tables d'associations quant à eux sont composées des deux noms des tables séparées d'un "_".

Les procédures débutent tous par un "p" minuscule, suivi du nom de l'action effectuée.

Les triggers débutent par les initiales de la condition qui déclanche le trigger (Ex.: BI pour Before Insert), suivit d'un "_" et d'un nom descriptif de sa responsabilité.

Le nom des contraintes de clé étrangères sont nommées "FK_(nom de la table contenant la référence)"

5 Règles d'affaires (contraintes)

Contrainte	**Stratégie employée pour implémentation**
Chaque client possède une adresse	La table Adresse est référencée à l'aide d'une FOREIGN KEY dans la table Client
Chaque client est abonné à un forfait	La table Forfait est référencée à l'aide d'une FOREIGN KEY dans la table Client
Chaque client possède les attributs d'une personne	La table Personne est référencée à l'aide d'une FOREIGN KEY dans la table Client
Chaque client a une carte de crédit enregistrée à	La table CarteCredit est référencée à l'aide d'une

son dossier	FOREIGN KEY dans la table
Chaque réalisateur possède les attributs d'une personne	La table Personne est référencée à l'aide d'une FOREIGN KEY dans la table Realisateur
Chaque acteur possède les attributs d'une personne	La table Personne est référencée à l'aide d'une FOREIGN KEY dans la table Acteur
Chaque film possède un réalisateur	La table Realisateur est référencée à l'aide d'une FOREIGN KEY dans la table Film
Chaque location possède une copie de l'inventaire et les informations du client	La table Inventaire est référencée à l'aide d'une FOREIGN KEY dans la table Location_Client, et la table Client est référencée à l'aide d'une FOREIGN KEY dans la table Location_Client
Chaque film peut posséder plusieurs acteurs et chaque acteurs peut jouer dans plusieurs films	La table Film est référencée à l'aide d'une FOREIGN KEY dans la table Film_Acteurs, et la table Acteur est référencée à l'aide d'une FOREIGN KEY dans la table Film Acteurs.
Un client ne peut pas entrer une adresse courriel déjà utilisée	Une UNIQUE KEY a été ajouté à l'attribut "courriel" de la table Client
Un client ne peut pas entrer une carte de crédit expirée	Un TRIGGER est fait lors de la création de la table CarteCredit où la date courante doit être inférieure à la date d'expiration de la carte
Un client ne peut pas avoir moins que 18 ans	Un TRIGGER est fait lors d'un insertion dans la table Client où la date de naissance doit être inférieure à la date courante - 18 ans
Le mot de passe doit contenir au moins 5 caractères alphanumériques	Un CHECK est fait lors d'une insertion dans la table Client et Employe qui valide si le motDePasse entré est conforme à l'expression régulière `^[a-zA-Z0-9]{5,}$`
Lors d'une location, la copie doit être disponible	Un TRIGGER est utilisé pour vérifier avant chaque insertion dans la table Location_Client si la dernière location de cette copie a été retourné (dateRetour n'est pas nul)
Lors d'une location, le forfait du client doit permettre la location	Un TRIGGER est utilisé pour vérifier avant chaque insertion si le client n'a pas déjà en sa possession le nombre maximal de copies alloué par son forfait

6 Opérations à encapsuler

Nom de la procédure	Opération
pAjouterClient	Création de compte pour un nouveau client. On débute par une insertion dans la table Personne, suivi d'une insertion dans la table Adresse, suivi d'une insertion dans la table CarteCredit, et on finit par une insertion dans la table client qui établit la relation entre les données insérées
pLouerFilm	Cette procédure prend en paramètre le ClientID et le FilmID. On doit débuter par trouver quel copie nous allons louer. On commence par vérifier les copies de ce film qui n'ont jamais été louées (edge case). Si nous n'en trouvons pas, nous regardons ensuite les copies qui ont étés retournées mais pas encore louées. Dans les deux cas, nous prenons le premier résultat et nous faisons une insertion de codeCopieID, clientID et dateLocation dans Location_Client. Dans le cas où les deux requêtes nous retournent aucun résultat, nous lançons une exception avec un message comme quoi il ne reste plus de copies disponibles.

7 Planification des tâches

Tâche	% de travail effectué par membre de l'équipe
Modèle conceptuel	Alexandre: 25% Samuel: 25% Mandaniaina: 25% Marc-André: 25%
Modèle relationnelle	Alexandre: 10% Samuel: 25% Mandaniaina : 35% Marc-André: 30%
Création des tables	Alexandre: 5% Samuel: 40% Mandaniaina: 55% Marc-André: 0%
Contraintes et procédures	Alexandre: 25% Samuel: 10%

	Mandaniaina: 65% Marc-André: 0%
Insertion des données	Alexandre: 0% Samuel: 0% Mandaniaina: 0% Marc-André: 100%
Rapport	Alexandre: 95% Mandaniaina: 3% Marc-André: 2%

8 Question théorique 1

Avantage 1: Limite le danger potentiel qu'une requête à la DB affecte l'intégrité des données à l'aide des contraintes créées lors de la normalisation.

Avantage 2: Réduit la duplication de données, ce qui permet d'effectuer des requêtes plus efficaces et réduit la taille des données stockées.

Tous nos tables sont en 3FN. Il n'y a pas de colonnes qui dépendent de quelque chose d'autre que leur clé primaire.

9 Question théorique 2

Record du nombre le plus élevé de films louées en une année par un client

```
CREATE VIEW plusGrandsNombresFilmLoueeEnUneAnneeParClient AS
SELECT c.clientID, MAX(p.prenom) AS prenom, MAX(p.nomFamille) AS nomFamille, COUNT(*)
AS nbLocations, TO_CHAR(lc.dateLocation, 'YYYY') AS annee
FROM Location_Client lc, Client c
INNER JOIN Personne p ON p.personneID = c.personneID
WHERE lc.clientID = c.clientID
GROUP BY c.clientID, TO_CHAR(lc.dateLocation, 'YYYY')
ORDER BY nbLocations DESC;
```

10 Création des TRIGGERS

Vérification de la date d'expiration de la carte

```
CREATE OR REPLACE TRIGGER BI_VerifierDateExpirationCarte
BEFORE INSERT ON CarteCredit
FOR EACH ROW
BEGIN
	IF :NEW.EXP_YEAR < EXTRACT(YEAR FROM SYSDATE) OR :NEW.EXP_YEAR = EXTRACT(YEAR
FROM SYSDATE) AND :NEW.EXP_MONTH < EXTRACT(MONTH FROM SYSDATE) THEN
		RAISE_APPLICATION_ERROR('-20000', 'Carte de crédit expiré');
	END IF;
END;
```

Vérification de l'âge du client

```
CREATE OR REPLACE TRIGGER BI_VerifierAgeClient
BEFORE INSERT ON Client
FOR EACH ROW
DECLARE
	DateNaissanceClient date;
BEGIN
	SELECT DateNaissance INTO DateNaissanceClient FROM Personne WHERE PersonneID =
:NEW.PersonneID;
	IF  DateNaissanceClient > ADD_MONTHS(SYSDATE,-216) THEN
		DELETE FROM Personne WHERE PersonneID = :NEW.PersonneID;
		RAISE_APPLICATION_ERROR('-20000', 'Le client doit avoir au moins 18
ans');
	END IF;
END;
```

Vérification de copie disponible

```
CREATE OR REPLACE TRIGGER BI_VerifierSiLocationDisponible
BEFORE INSERT ON Location_Client
FOR EACH ROW
DECLARE
	EstLoue INTEGER;
BEGIN
	SELECT COUNT(*) INTO EstLoue FROM Location_Client WHERE CodeCopieID =
:NEW.CodeCopieID AND dateRetour IS NULL;
	IF EstLoue <> 0 THEN
		RAISE_APPLICATION_ERROR('-20000', 'La copie doit être disponible pour
pouvoir la louer');
	END IF;
END;
```

Vérification de location admissible avec le forfait

```
CREATE OR REPLACE TRIGGER BI_VerifierSiClientLouePlusQueMax
BEFORE INSERT ON Location_Client
FOR EACH ROW
DECLARE
	NBFilm INTEGER;
	NBFilmMax INTEGER;
BEGIN
	SELECT COUNT(*) INTO NBFilm FROM Location_Client WHERE CLIENTID = :NEW.CLIENTID
AND dateRetour IS NULL;
	SELECT LocationMax INTO NBFilmMax FROM Client INNER JOIN forfait ON
Client.FORFAITID = Forfait.FORFAITID WHERE Client.CLIENTID = :NEW.CLIENTID;
	IF NBFilm >= NBFilmMax THEN
		RAISE_APPLICATION_ERROR('-20000', 'Le client ne peut pas avoir plus de
location que son forfait lui permet');
	END IF;
END;
```

11 Création des procédures stockées

Location d'un film

```
create or replace PROCEDURE pLouerFilm
    (filmID_in IN NUMBER, clientID_in IN NUMBER)
IS
    copieALouer NUMBER;
BEGIN
    BEGIN
        /* Check for copies that have never been rented */
        SELECT inv.codeCopieID INTO copieALouer
        FROM Inventaire inv
        LEFT JOIN Location_Client lc ON lc.codeCopieID = inv.codeCopieID
        WHERE filmID = filmID_in
            AND lc.LOCATIONID IS NULL
            AND ROWNUM = 1;

        INSERT INTO Location_Client (codeCopieID, clientID, dateLocation)
        VALUES (copieALouer, clientID_in, SYSDATE);

    EXCEPTION WHEN NO_DATA_FOUND THEN
            SELECT codeCopieID INTO copieALouer /* Copy for this movie that has been
returned and not yet rented*/
        FROM Inventaire
        WHERE filmID = filmID_in
            AND codeCopieID NOT IN
            (
                SELECT codeCopieID   /* Rented copies */
                FROM Location_Client
                WHERE
                    codeCopieID = ANY (
                        SELECT codeCopieID
                        FROM Inventaire
                        WHERE filmID = filmID_in
                    )
                AND dateRetour IS NULL
             )
            AND ROWNUM = 1;

        INSERT INTO Location_Client (codeCopieID, clientID, dateLocation)
        VALUES (copieALouer, clientID_in, SYSDATE);
    END;
EXCEPTION WHEN NO_DATA_FOUND THEN
      raise_application_error(-20001,'Plus de copies disponibles pour le film ' ||
filmID_in);
END pLouerFilm;
```

Création d'un client

```
create or replace PROCEDURE pCreerClient
      (prenom_in IN VARCHAR2, nomFamille_in IN VARCHAR2, dateNaissance_in IN DATE,
numeroTel_in IN VARCHAR2, courriel_in IN VARCHAR2, password_in IN VARCHAR2,
     noCivique_in IN VARCHAR2, rue_in IN VARCHAR2, ville_in IN VARCHAR2, province_in IN
VARCHAR2, codePostal_in IN VARCHAR2, typeCarte_in IN VARCHAR2, numero_in IN NUMBER,
     exp_month_in IN NUMBER, exp_year_in IN NUMBER, cvv_in IN NUMBER, forfaitID_in IN
NUMBER)
IS
```

```
    adresseID NUMBER;
    personneID NUMBER;
    carteCreditID NUMBER;
BEGIN
    INSERT INTO Adresse (noCivique, rue, ville, province, codePostal)
    VALUES (noCivique_in, rue_in, ville_in, province_in, codePostal_in)
    RETURNING adresseID INTO adresseID;

    INSERT INTO CarteCredit (typeCarte, numero, exp_month, exp_year, cvv)
    VALUES (typeCarte_in, numero_in, exp_month_in, exp_year_in, cvv_in)
    RETURNING carteCreditID INTO carteCreditID;

    INSERT INTO Personne (prenom, nomFamille, dateNaissance)
    VALUES (prenom_in, nomFamille_in, dateNaissance_in)
    RETURNING personneID INTO personneID;

      INSERT INTO Client (adresseID, carteCreditID, personneID, forfaitID, courriel,
numeroTel, password)
         VALUES (adresseID, carteCreditID, personneID, forfaitID_in, courriel_in,
numeroTel_in, password_in);

EXCEPTION
WHEN OTHERS THEN raise_application_error(-20001,'Erreur dans la procédure pCreerClient
- '||SQLCODE||' -ERROR- '||SQLERRM);
END pCreerClient;
```