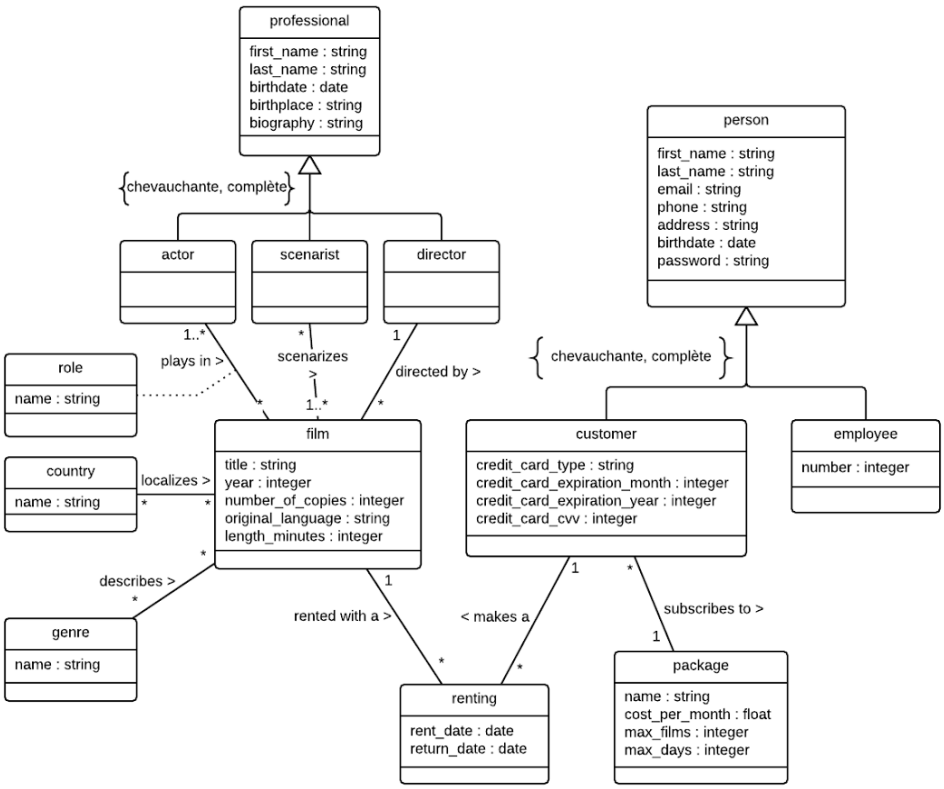
Laboratoire 1 : Conception du schéma relationnel

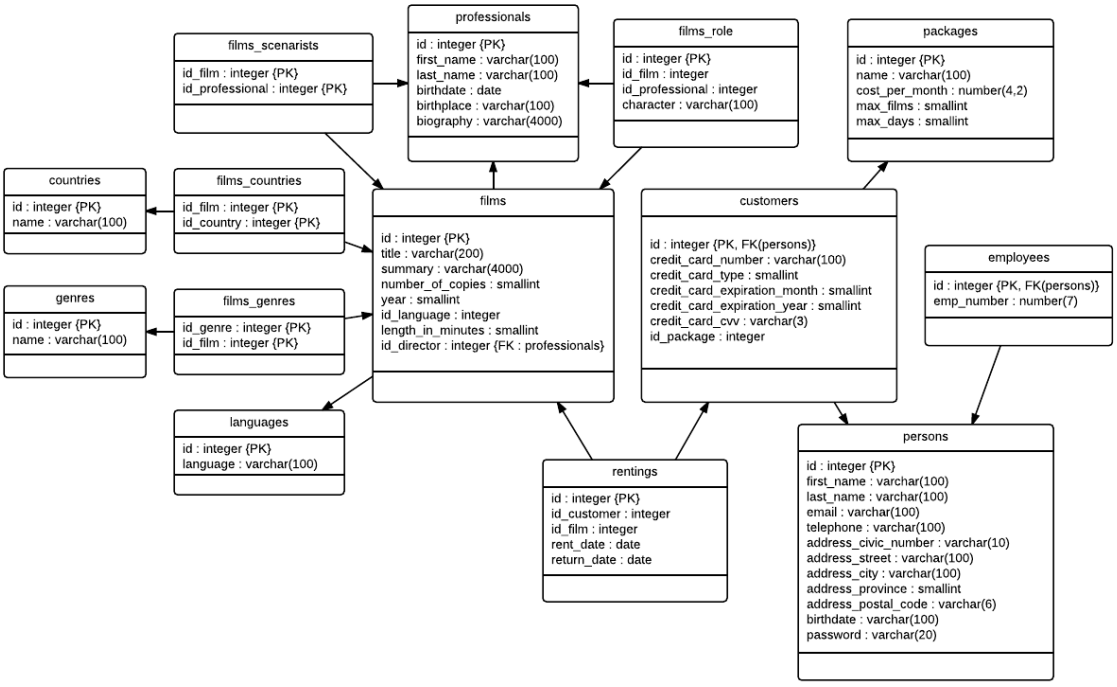
Cours	LOG660 – Base de données haute performance
Session	hiver 2015
N° de laboratoire	1
Groupe	01
Étudiant 1 (nom et CP)	Étienne Beaudry-Auger BEAE05029000
Étudiant 2 (nom et CP)	Martin Desharnais DESM21099102
Étudiant 3 (nom et CP)	Samuel Milette-Lacombe MILS26059100
Professeur	Christian Desrosiers
Chargé de laboratoire	Richard Rail
Date	3 février 2015

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

Les informations sur les professionnels sont regroupées

Un client devant pouvoir obtenir certaines informations sur le réalisateur et les acteurs d'un film, une table « professionals » a été créée afin de contenir cette information partagée. Il a été décidé que cette information serait aussi disponible pour les scénaristes. Ce partage de l'information a comme avantage qu'un même professionnel peut occuper plusieurs fonctions sans que ses informations ne soient dupliquées.

Les acteurs sont implicites

Un acteur pouvant jouer plus d'un personnage dans un film, nous le concept d'acteur a été remplacé par celui de rôle. Pour obtenir la liste des acteurs d'un film, il suffit d'interroger la table « films_role » afin d'obtenir la liste des professionnels distincts ayant un rôle dans le film.

Les informations sur les personnes sont regroupées
Les informations à conserver sur un client et un employé étant similaires, une table « persons » a été créée afin de contenir cette information partagée. Ce partage de l'information a comme avantage qu'une même personne peut être à la fois un employé et un client sans que ses informations ne soient dupliquées.

Une location ne concerne qu'un seul film
La location de chaque film entraîne une location distincte. De cette façon, il n'y a pas à gérer le cas où un client ne retourne qu'un film sur une location en comportant deux.

Un concept unique représente les locations et les retours
Les actes de louer et de retourner un film étant intimement reliés, les deux concepts ont été unifiés sous l'égide de la location. Une location où la date de retour et nulle est une location en cours. Lors d'un retour, la date est inscrite dans le champ correspondant afin de clore la location.

Le nombre de copies disponibles d'un film est implicite
La table « films » contient le nombre total de copies d'un film, mais pas le nombre de copies actuellement disponibles. Cette information peut être calculée à partir du nombre de copies totales et de la sommation des locations en cours pour ledit film. L'information étant calculée, elle ne peut donc pas être désynchronisée avec la réalité.

Utilisation d'une représentation compacte pour les domaines restreints
Les colonnes à domaines restreints utilisent une représentation numérique compacte et performante. Cet encodage s'adapte aisément au concept d'énumération présent dans la plupart des langages de programmation.

4 Conventions de nommage

1. Les déclencheurs sont préfixés de « trig_ ».
2. Les procédures stockées sont préfixées de « proc_ » suivi de l'action et du nom des tables impactées. La séparation se fait via « _ ». La même règle s'applique pour les « triggers ».
3. Les séquences sont préfixées de « seq_ ».
4. Les vérifications de condition « check » sont préfixées par « ck_ ».
5. Les clés primaires artificielles portent le nom « id ».
6. Les clés étrangères sont définies par « id_<nom_de_colonne> ».
7. Pour le langage de programmation nous utilisons la convention de Golang : https://golang.org/doc/effective_go.html

5 Règles d'affaires (contraintes)

Validation de la province de la personne
Il a été décidé que l'identifiant de la province serait un nombre entre 0 et 9. Chaque nombre est associé à une province particulière. Il s'agit d'un modèle de valeurs énumératves représentant les valeurs suivantes: enum province = AB 0 | BC 1 | MB 2 | NB 3 | NL 4 | NS 5 | ON 6 | PE 7 | QC 8 | SK 9.

Validation de la carte de crédit du client
Le type de carte de crédit du client est validé avec une contrainte check avec encore une fois un nombre énumératif correspondant à l'une des valeurs suivantes: enum credit_card_type = VISA 0 | MASTER_CARD 1 | AMERICAN_EXPRESS 2.

Les champs formant la date d'expiration du client doivent aussi être validés mais cette fois à l'aide d'un déclencheur puisqu'une comparaison avec SYSDATE nécessaire et cela est impossible dans une contrainte CHECK. Le déclencheur trig_validate_customer_columns valide le mois et l'année de l'expiration de la carte de crédit du client avant que l'insertion soit faite.

Validation du format et de la longueur du mot de passe
Le mot de passe d'une personne dans la table Persons est validée par une contrainte CHECK utilisant un regex pour vérifier qu'il s'agit bien d'un mot de passe alphanumérique et qu'il fait bien 5 caractères de longs.

Validation du maximum de locations d'un client
Le cas 1 parle de forfait qui régit le nombre maximal de locations que le client peut avoir. Il faut alors un déclencheur `trig_check_max_renting` exécuté avant l'insertion d'une location (Renting) pour vérifier si le nombre de films loués par le client n'a pas atteint la limite de son forfait. Si c'est le cas, le déclencheur doit lancer une exception et bloquer l'insertion.

Validation de l'âge d'une personne
Le cas 1 indique les conditions nécessaires pour que certaines données soient valides. Le client doit notamment avoir 18 ou plus. Le déclencheur `trig_validate_customer_columns` intègre une validation avant une insertion dans la table Customers, puisqu'une vérification à l'aide du SYSDATE d'Oracle ne peut être faite avec un simple CHECK. Étant donné que l'âge peut être obtenue seulement à partir de la date de naissance de la personne reliée au client (champ birthdate), un SELECT sur la table Persons avec une jointure ave la table Customers est nécessaire. La jointure se fait d'ailleurs avec l'ID du client qui est le même que pour la personne qui y est reliée.

6 Opérations à encapsuler

Le cas d'utilisation 1 parle de l'abonnement des clients. Il s'agit donc d'une procédure stockée pour l'ajout d'un client. Cette fonction, nommée `proc_add_customer` consiste simplement en l'exécution d'une insertion sur la table Customers en utilisant un ID de la séquence de Customers et les champs en paramètres.

Le cas 4 parle de location de film. Il faut alors une procédure stockée pour l'insertion de locations (Renting) dans la base de données. Cette fonction nommée `proc_add_renting` fait simplement une requête d'insertion dans la table Rentings en utilisant un ID de la séquence de Rentings et les champs en paramètres constituants les valeurs des colonnes de la table.

7 Planification des tâches

Tâche	Ressource
Tâche 1 : Schema conceptuel (15 %)	
Faire le diagramme de classe	Étienne
Tache 2 : Schéma relationnel (15 %)	
Faire le diagramme UML pour le modèle relationnel	Martin
Créer le script SQL	Martin & Samuel
Tache 3 : Contraintes et procédures (20%)	
Contraintes et procédures cas 1	Samuel
Contraintes et procédures cas 4 :	Martin
Tache 4 : Insertion des données (50 %)	
Écriture de la définition dans le ORM, Configuration de l'ORM, Installation des drivers OCI8 d'Oracle	Étienne & Samuel
Extraction des données XML	Étienne & Samuel

8 Question théorique 1

La normalisation des tables permet d'éviter la redondance des données et facilite les mises à jour des données. Il est considéré comme une bonne pratique d'éviter la duplication des données, car cela permet de garder le contrôle de la qualité de l'information. Cela permet aussi de maintenir à jour une source de donnée au lieu de plusieurs ce qui diminue considérablement le travail de la maintenance des données du système. Finalement, le système évite de gaspiller l'espace mémoire en s'assurant que toute information sauvegardée apporte une valeur ajoutée à l'information.

En bref, cela :

- évite la redondance des données;
- facilite les mises à jour des données.

Tables en troisième forme normale
Les tables suivantes sont en troisième normale.

- countries
- employees
- films
- films_actors
- films_countries
- films_genres
- films_scenarists
- genres
- languages
- packages
- professionals
- rentings

Tables qui ne sont pas en troisième forme normale
Les tables suivantes ne sont pas en troisième forme normale :

- customers
    - credit_card_number ⇒ credit_card_type, credit_card_expiration_month, credit_card_year, credit_card_cvv
- persons
    - address_civic_number, address_street, address_city ⇒ address_postal_code
    - address_city ⇒ address_province
    - address_postal_code ⇒ address_city

9 Question théorique 2

La vue `films_available` permet d'afficher pour chaque film, son nombre de copies total et son nombre de copies louées.

```
CREATE OR REPLACE VIEW films_available AS

SELECT films.title, films.number_of_copies AS total, count(rentings.id) as rented

FROM films

LEFT JOIN RENTINGS on films.id = rentings.id_film

GROUP BY films.title, films.number_of_copies;
```

10 Création des TRIGGERS

[Insérez ici le code SQL permettant la création des TRIGGERS de la tâche 3 de l'énoncé.]

```
CREATE OR REPLACE TRIGGER trig_validate_customer_columns

BEFORE INSERT ON customers

FOR EACH ROW

DECLARE

v_birthdate date;

BEGIN

     IF     (TO_DATE(:NEW.credit_card_expiration_year     ||     '-'     ||
:NEW.credit_card_expiration_month,  'YYYY-MM')  <  TO_DATE(TO_CHAR(SYSDATE,
'YYYY-MM'), 'YYYY-MM')) THEN

      RAISE_APPLICATION_ERROR(-20001, 'Cannot insert customer because his
credit card is expired.');

    END IF;

    SELECT birthdate INTO v_birthdate from persons WHERE id = :NEW.id;

    IF (MONTHS_BETWEEN(SYSDATE, v_birthdate)/12 < 18) THEN

     RAISE_APPLICATION_ERROR(-20001, 'Cannot insert customer because the
related person is not 18 years old.');

    END IF;
```

```
END trig_validate_customer_columns;
```

```
CREATE OR REPLACE TRIGGER trig_check_max_renting
BEFORE INSERT ON rentings
FOR EACH ROW
DECLARE
    current_package_max_rentings number;
    current_rentings_number number;
BEGIN
     SELECT packages.max_films INTO current_rentings_number
     FROM packages JOIN customers ON customers.id_package = packages.id
     WHERE customers.id = :NEW.id_customer;


     SELECT COUNT(*) INTO current_rentings_number
     FROM rentings JOIN customers ON customers.id = rentings.id_customer
        WHERE customers.id = :NEW.id_customer and rentings.return_date IS
NULL;
      IF (current_rentings_number > current_package_max_rentings) THEN
      RAISE_APPLICATION_ERROR(-20001, 'Cannot insert renting because the
customer has exceeded the number of rented films of his package');
     END IF;
END trig_check_max_renting;
```

11 Création des procédures stockées

[Insérez ici le code SQL permettant la création des PROCEDURES de la tâche 3 de l'énoncé.]

```
CREATE OR REPLACE PROCEDURE proc_add_customer(
    p_credit_card_number varchar,
    p_credit_card_type smallint,
    p_credit_card_expiration_month smallint,
    p_credit_card_expiration_year smallint,
    p_credit_card_cvv varchar,
    p_id_package integer)
    IS
    BEGIN
        INSERT INTO customers
            (id,
            credit_card_number,
            credit_card_type,
            credit_card_expiration_month,
            credit_card_expiration_year,
            credit_card_cvv,
            id_package)
        VALUES
            (seq_customers.NEXTVAL,
            p_credit_card_number,
            p_credit_card_type,
            p_credit_card_expiration_month,
            p_credit_card_expiration_year,
            p_credit_card_cvv,
            p_id_package);
    END proc_add_customer;

CREATE OR REPLACE PROCEDURE proc_add_renting(
    p_id_customer integer,
    p_id_film integer
      ) AS
    BEGIN
    INSERT INTO rentings
        (id,
        id_customer,
        id_film,
        rent_date)
    VALUES
        (seq_rentings.nextval,
        p_id_customer,
        p_id_film,
        SYSDATE);
    END proc_add_renting;
```