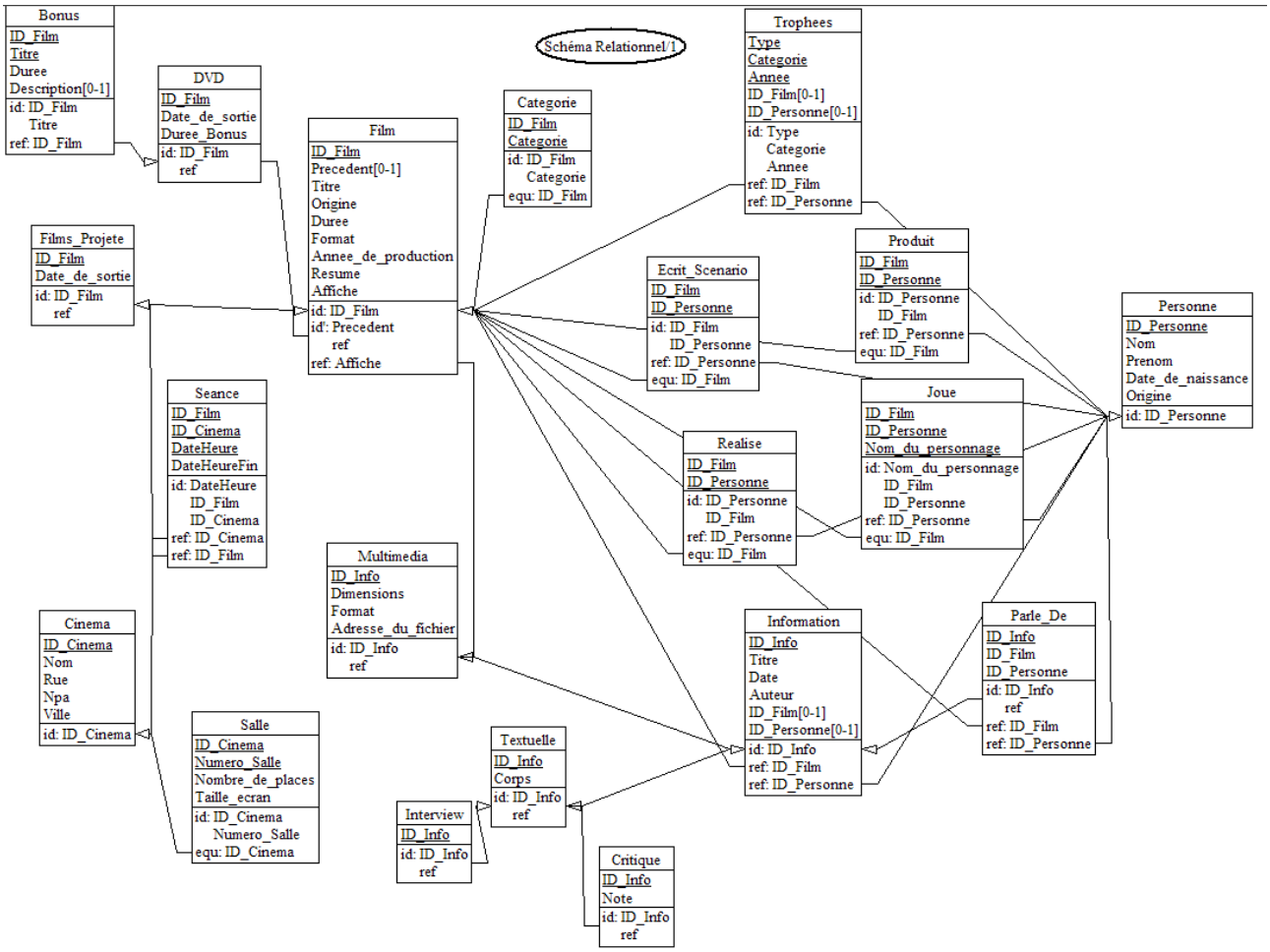
1 Annexes

Avec ce rapport il faut rendre en annexe le **script SQL corrigé** qui permet de **créer la base de données** selon votre modèle relationnel ainsi que de **la peupler** avec un nombre de tuples suffisant.

Modifications :

- La table salle est désormais identifieé par ID_Cinema et Numero_Salle pour permettre l'insertion de plusieurs salles avec une même taille d'écran et le nombre de places.
- Nous avons ajouté à la table seance un attribut DateHeureFin pour pouvoir comparer deux attributs dans un check.
- Nous avons ajouté à la table dvd l'attribut Duree_Bonus.
- Nous avons ajouté à la table bonus l'attribut Duree.



2 Requêtes SQL

Pour chacune des requêtes SQL demandées vous devez fournir **l'énoncé** de la requête en français en plus de **la requête** elle-même.

Requête SQL avec utilisation du mot clé DISTINCT

Enoncé: Tous les types de catégories qui existent dans la base de données.

Requête: SELECT DISTINCT Categorie FROM Categorie;

Requête SQL avec bloc emboîté (sous-requête) utilisant le mot clé [NOT] IN

Enoncé: Titre des films dans lesquels jouent des acteurs nés en 1980.

Requête: SELECT Titre FROM FILM WHERE ID_Film IN(SELECT ID_Film FROM Joue WHERE ID_Personne IN (SELECT ID_Personne FROM Personne WHERE to_char(Date_de_naissance, 'yyyy') ='1980'));

Requête SQL avec bloc emboîté avec utilisation du mot clé [NOT] EXISTS

Enoncé: Nom et prénom des personnes qui ont gagné au moins un Trophée.

Requête: SELECT p.Nom, p.Prenom FROM Personne P WHERE EXISTS (SELECT * FROM Trophees t WHERE t.ID_Personne = p.ID_Personne);

Requête SQL en utilisant plusieurs fois la même table (alias)

Enoncé: Titre des films qui sont une suite.

Requête: SELECT f1.Titre FROM Film f1, Film f2 WHERE f1.Precedent = f2.ID_Film;

Requête SQL en utilisant les clauses GROUP BY et HAVING

Enoncé: Catégorie et Nombre de films appartenant cette dernière parmi les catégories contenant plus de deux films.

Requête: SELECT Categorie, count(ID_Film) as "Nombre de films" FROM Categorie GROUP BY Categorie HAVING count(ID_Film) > 2

Requête SQL avec opérations ensembliste (UNION ou INTERSECT ou EXCEPT)

Enoncé: Nom, prénom et fonction des personnes ayant participé au film Casino Royale.

Requête: SELECT nom, prenom, 'Scénariste' as "Fonction" FROM Personne JOIN Ecrit_Scenario USING (ID_personne) JOIN Film USING (ID_Film) WHERE Titre = 'Casino Royale' UNION

SELECT nom, prenom, 'Producteur' as "Fonction" FROM Personne JOIN Produit USING (ID_personne) JOIN Film USING (ID_Film) WHERE Titre = 'Casino Royale' UNION

SELECT nom, prenom, 'Réalisateur' as "Fonction" FROM Personne JOIN Realise USING (ID_personne) JOIN Film USING (ID_Film) WHERE Titre = 'Casino Royale' UNION

SELECT nom, prenom, 'Acteur' as "Fonction" FROM Personne JOIN Joue USING (ID_personne) JOIN Film USING (ID_Film) WHERE Titre = 'Casino Royale' ORDER BY "Fonction" ASC, nom ASC, prenom ASC;

Requête SQL d'insertion de tuples à partir d'une requête SQL

Enoncé: Ajouter au film ayant comme identifiant le numéro 8, les catégories du film ayant l'identifiant numéro 2.

Requête: INSERT INTO categorie (id_film, categorie) SELECT '8', categorie FROM categorie WHERE id_film = '2'

Requête SQL de mise à jour de tuples à partir d'une requête SQL

Enoncé: Inverser les notes données par l'auteur Sophie Andrey, (car il y a eu un mal entendu sur le sens de l'échelle.)

Requête: UPDATE critique c set (note) = (SELECT 6 - c.note FROM Information i, textuelle t WHERE i.id_info = t.id_info AND t.id_info = c.id_info AND i.auteur = 'Sophie Andrey')

3 CHECK

Pour chacune des CHECK demandés vous devez fournir **l'énoncé** de la contrainte implémenté par le CHECK en langage naturel, **la raison** pour laquelle cette requête devrait être implémentée par un CHECK plutôt que par un autre mécanisme et **la requête** qui est utilisé pour créer le CHECK.

CHECK 1 - contrainte de domaine utilisant le mot clés IN ou BETWEEN

Enoncé: Vérifier que la note donnée pour la critique est entre 0 et 6.

Raison: S'assurer que le domaine de valeur est valide.

Requête: check(Note BETWEEN 0 AND 6)

CHECK 2 - Contrainte par rapport à d'autres attributs de la même table

Enoncé: L'attribut DateHeureFin de la table séance doit être plus grand que l'attribut DateHeure.

Raison: La fin d'une séance doit être obligatoirement après son début.

Requête: check(DateHeureFin > DateHeure).

CHECK 3 - utilisant une combinaison logique de conditions (AND, OR, NOT)

Enoncé: Vérifier qu'un trophée est soit attribué à un film soit à une personne. Il ne peut être attribué à un film et une personne en même temps.

Raison: On ne veut pas qu'un trophée soit attribué à aucun film ou à personne. On ne souhaite pas non plus qu'un trophée soit attribué à une personne et à un film en même temps. Un trophée touche les personnes indirectement par que le fait que ces derniers ont participés à son succès et inversement.

Requête: check((ID_Film IS NULL AND ID_Personne IS NOT NULL) OR (ID_Personne IS NULL AND ID_Film IS NOT NULL))

4 TRIGGER

Pour chacune des triggers demandés vous devez fournir **l'énoncé** de la contrainte implémenté par le trigger en langage naturel, **la raison** pour laquelle cette requête devrait être implémentée par un trigger plutôt que par un autre mécanisme, **la description** de ce que le trigger fait et **la requête** qui est utilisé pour le créer.

CHECK 1 - Trigger de type STATEMENT LEVEL

Enoncé: L'objectif de ce trigger est de maintenir correct la somme totale de la durée des Bonus DVD.

Raison: Parce que dans le cadre de cette modification, plusieurs tables sont modifiées.

Description: Après l'ajout ou la modification d'un tuple dans la table Bonus, le trigger est déclenché et mets à jour l'attribut Duree_Bonus dans la table DVD.

Requête:

```
CREATE OR REPLACE TRIGGER ControleDureeDVD
AFTER INSERT OR UPDATE OR DELETE ON Bonus
BEGIN
  UPDATE dvd set Duree_Bonus = NVL((SELECT sum(duree) FROM Bonus WHERE bonus.id_film =
        dvd.id_film), 0);
END;
```

CHECK 2 - Trigger de type FOR EACH ROW LEVEL

Enoncé: Vérifier l'intégrité de la valeur entrée dans le champs Duree_bonus.

Raison: Cela permet d'effectuer une requête sur une autre table, ce que les autres moyens ne permettent pas

Description: Lorsque l'attribut Duree_bonus de la table DVD est modifié, si cette nouvelle valeur ne correspond pas à la valeur réelle calculée, une erreur est affichée.

Requête:

```
create or replace
TRIGGER ModificationDureeBonus AFTER UPDATE OF Duree_Bonus ON DVD
FOR EACH ROW
DECLARE
dureeBonus dvd.duree_bonus%TYPE;
BEGIN
SELECT SUM(duree) INTO dureeBonus FROM bonus WHERE bonus.id_film = :new.id_film;
IF dureeBonus <> :new.duree_bonus OR dureeBonus IS NULL AND :new.duree_bonus <>0 THEN
raise_application_error(-20002,'Durée des bonus incorrect!');
END IF;
END;
```

CHECK 3 - Trigger de type FOR EACH ROW LEVEL

Enoncé: Un film ne peut être projeté avant qu'il ne soit produit.

Raison: Le trigger nous permet de profiter de requêtes et d'avoir accès à la valeur future qui sera insérée

Description: Ce trigger est appelé avant l'insertion ou la mise à jour d'un tuple de la table Film_Projete, il vérifie si la valeur entrée est antérieur à l'année de production. Si ce n'est pas le cas, il affiche une erreur.

Requête :

```
CREATE OR REPLACE TRIGGER CheckDateSortieValid
BEFORE INSERT OR UPDATE ON FILMS_PROJETE
FOR EACH ROW
DECLARE
anneeProd film.annee_de_production%TYPE;
BEGIN
  SELECT annee_de_production INTO anneeProd FROM FILM WHERE id_film = :new.id_film;
  IF anneeProd > :new.date_de_sortie THEN
    raise_application_error(-20001,' Un film ne peut être projeté avant qu il ne soit produit!');
  END IF;
END;
```