le cnam

Bases de Données - PL/SQL, Trigger, Concurrence

Exercices et Travaux Pratiques

FIP1

Cnam
nicolas.travers@cnam.fr

Chapitre

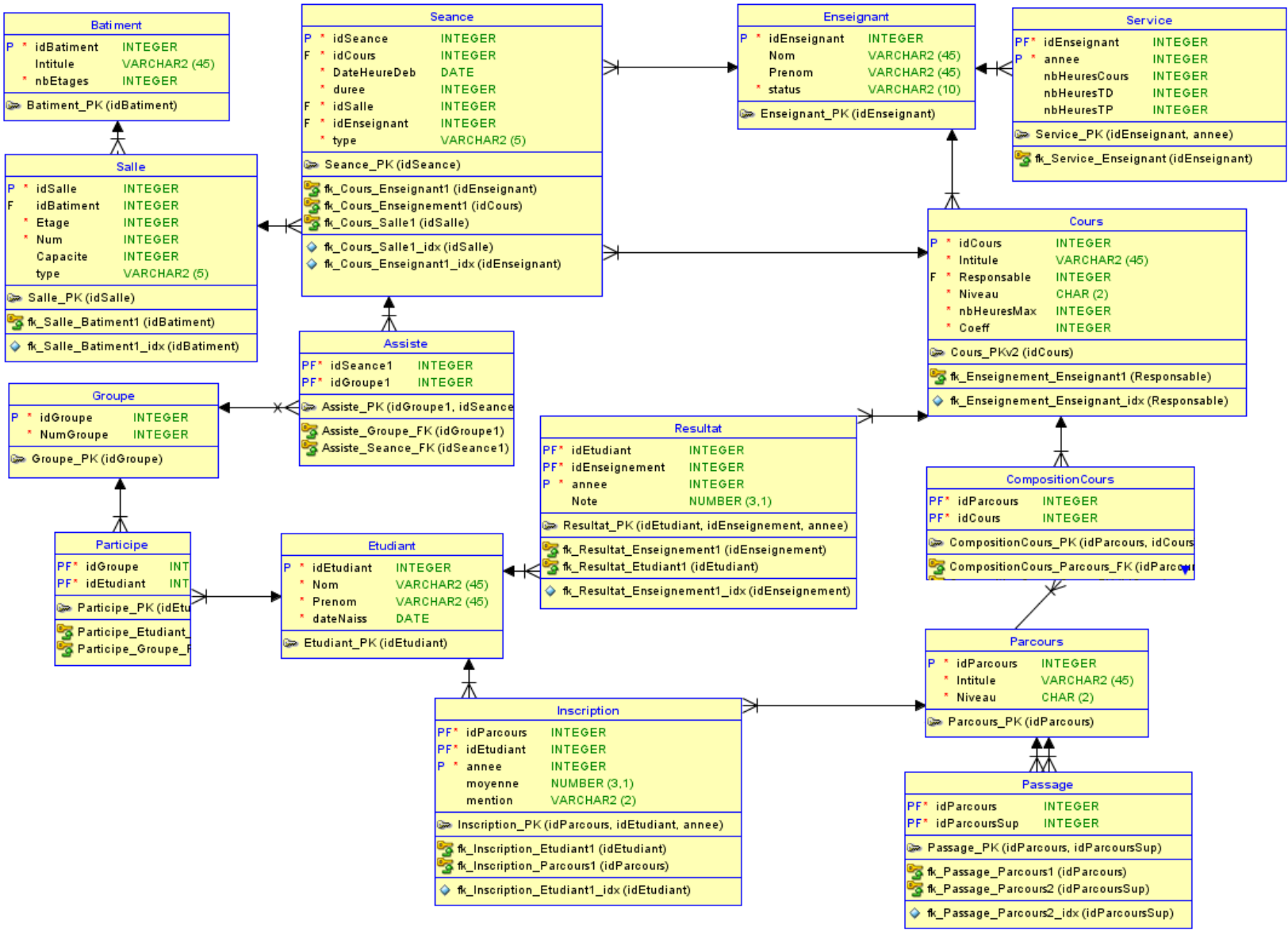
Table des matières

Chapitre 1

Environnement des Travaux Pratiques

1.1 Base de données

Dans le cadre du Travaux Pratique, nous utiliserons la base de données 'Ecole' avec le TABLESPACE 'TEST_BD' avec le schéma suivant :



La base de données utilisée gère l'emploi du temps d'une école d'enseignement supérieur. Elle intègre les étudiants, les matières, les enseignants et les différents cursus.

1.2 Installation

1.2.1 Installation classique

Dans le cas où vous souhaiteriez faire une installation classique, vous devrez :

— Installer `Oracle Express 11g` et `SQLDeveloper`
— Lancer le serveur
— Sous `SQLDeveloper`, créer une connexion "SYSTEM" avec le serveur
— Lancer les scripts de création du TableSpace, utilisateur et droits d'accès
— Créer une connexion "TEST_USER" avec le serveur
— Créer la base de données et insérer les données

Les scripts de création sont disponibles ici :

— Création du TableSpace, utilisateurs et droits : `http://chewbii.com/wp-content/uploads/2016/02/creation.sql_.zip`
— Création des tables et insertions de tuples : `http://chewbii.com/wp-content/uploads/2016/02/BD_TP.zip`

Un guide est disponible à cette adresse : `http://chewbii.com/wp-content/uploads/2016/02/FIP1_guide.pdf`

1.2.2 Installation via Docker

Nous allons installer une base de données dont l'environnement est déjà tout préparé (TABLESPACE, utilisateur, BDD). L'installation via l'interface `Kitematic` est assez simple, nous allons détailler celle en ligne de commande.

Pour une installation en ligne de commande, veuillez faire les instructions suivantes :

— Il est possible que vous ayez à rajouter "*sudo*" devant les instructions ci-dessous.
— Importer l'image (fichier local[1]) : **docker import /data/tp-oracle-11g images-tp-oracle**
— Lancer une instance de l'image (container) :
docker run -d −−shm-size=2g -p 1521 :1521 -p 8080 :8080 −−name tp-oracle images-tp-oracle /start.sh

Un guide docker est disponible à cette adresse :
`http://chewbii.com/docker/` et `http://chewbii.com/wp-content/uploads/2016/09/guide-docker.pdf`
Voici quelques instructions de bases :

— `docker images` : Lister les images
— `docker ps` : Lister les containers demarrés
— `docker ps -a` : Lister tous les containers (même non demarrés)
— `docker start <<container id visible avec "ps -a">>` : Relancer un container éteint
— `docker stop <<container id visible avec "ps">>` : Eteindre un container

Le nom de l'image est disponible sur le `docker hub` avec ce nom : `traversn/tp-oracle-xe11g-bdd-ecole:latest`.

1.2.3 Lancer docker pour utilisation ultérieure

Maintenant que l'image a été téléchargée (*docker import*) et instanciée dans un container (*docker run*), il faudra :

1.2.1 Vérifier que le serveur tourne : `docker ps`
Si cela ne retourne pas le container (tp-oracle), il faut le lancer

1.2.2 `docker start <<container id>>` (visible avec "ps -a")

1.3 Connexion au serveur via SQLDeveloper

Une fois l'image installée, vous pouvez vous connecter avec le logiciel `SQLDeveloper`.

La base de données a les informations suivantes :

— **host** : localhost (ou IP en fonction de la version de docker)
— **port** : 1521 (sauf s'il a été redirigé par docker)
— **login** : test_user
— **mdp** : test

1. Si le fichier n'existe pas, le chercher sur docker-hub :**docker pull –name images-tp-oracle traversn/tp-oracle-xe11g-bdd-ecole :latest**

Chapitre 2

Procédures Stockées - Travaux Pratiques

Dans le cadre de ces travaux pratiques, nous utiliserons SQLDeveloper pour nous connecter à la base de données Oracle (Serveur 11gR2). Pour cela, veuillez vous référer au guide d'utilisation de SQLDeveloper disponible avec ce support d'exercices (seule la section 2.3 nous intéresse actuellement).

Pour que la sortie standard du SGBD soit retournée vers SQLDevelopper (sinon, aucun retour ne sera affiché), veuillez exécuter la commande suivante :

```
SET SERVEROUTPUT ON;
```

2.1 Programme simple

2.1.1 Exécuter le programme suivant ;

```
DECLARE
        id int;
BEGIN
        SELECT idEtudiant INTO id
        FROM Etudiant WHERE nom = 'Jaquemin' and prenom= 'Olivier';
        DBMS_OUTPUT.PUT_LINE('Identifiant de Jaquemin Olivier : ' || id);
END;
```

2.1.2 Modifier le programme précédent pour que le nom et prénom soient des variables définie à l'extérieur grâce à 'ACCEPT' ;

2.2 Procédures simples

2.2.1 Créer une procédure 'affIdEnseignant' qui affiche l'identifiant d'un enseignant pour un nom et prénom donné. La tester avec le programme ci-dessous :

```
ACCEPT a_nom prompt 'Nom : ';
ACCEPT a_prenom prompt 'Prenom : ';
EXECUTE affEnseignant('&a_nom', '&a_prenom');
```

2.2.2 Créer une procédure 'affNbEtudiant' qui affiche le nombre d'étudiants inscrits à un parcours (id) et une année donnée. Afficher l'intitulé du parcours correspondant ;

2.3 Fonctions

2.3.1 Créer une fonction 'retResponsable' qui retourne l'identifiant du responsable pour un nom de cours donné. Tester avec le programme suivant :

```
ACCEPT v_intitule prompt 'Intitulé du cours : ';
DECLARE
        idResp number;
BEGIN
        idResp := retResponsable('&v_intitule');
        DBMS_OUTPUT.PUT_LINE('&v_intitule  : '|| idResp);
END;
```

2.3.2 Créer une fonction 'moyennePonderee' qui retourne la moyenne pondérée des résultats d'un étudiant (utiliser la fonction 'retEtudiant') et année donnés. Tester avec le programme suivant ('Jaquemin', 'Olivier' et 2013) :

```
ACCEPT v_nom prompt 'Nom : ';
ACCEPT v_prenom prompt 'Prenom : ';
ACCEPT v_annee prompt 'Annee : ';
DECLARE
        ret number;
BEGIN
        ret := moyenneponderee('&v_nom', '&v_prenom', &v_annee);
        DBMS_OUTPUT.PUT_LINE('&v_nom &v_prenom (&v_annee) : '|| ret);
END;
```

2.4 Curseurs

2.4.1 Créer une procédure 'affEtudiants' qui affiche la liste des noms et prénoms d'étudiants inscrits d'une année donnée en paramètre. Tester avec l'exécution suivante :

```
execute aff_Etudiants (2014);
```

2.4.2 Donner la procédure qui affiche, pour le nom d'un parcours, niveau et année donnés, le nom des étudiants inscrits et leur moyenne pondérée en utilisant la fonction 'moyennePonderee'. Tester avec ;

```
EXECUTE affParcours ('Informatique', 'M1', 2014);
```

2.4.3 Créer une procédure 'affNotes' qui affiche les notes avec l'intitulé du cours correspondant pour un étudiant (nom et prénom) et année donnés ;

2.4.4 Créer une procédure 'affMatieres' qui affiche les intitulés et coefficients des matières qu'un étudiant (nom et prénom) suit durant une année donnée. Trier les matières par coefficients décroissants ;

2.4.5 Créer une procédure 'affInterventions' qui affiche pour chaque niveau de parcours croissant les intitulés de cours et de parcours dans lequel intervient un enseignant donné (nom et prénom) ;

2.4.6 Créer une procédure 'MAJ_BULLETIN' qui met à jour toutes les moyennes des inscriptions (utiliser la fonction 'moyennePondere') des étudiants d'un parcours donné (id). Tester la procédure avec ceci :

```
UPDATE Inscription SET Moyenne = NULL WHERE idParcours = 1;
EXECUTE MAJ_BULLETIN(1);
SELECT * FROM INSCRIPTION WHERE idParcours = 1;
```

2.5 Exceptions

2.5.1 Modifier la fonction 'retEtudiant' pour récupérer les erreurs lorsque l'étudiant n'est pas trouvé, ou si plusieurs sont retournés (homonymes). Retourner la valeur -1 dans ce cas ;

2.5.2 Modifier la procédure 'affMatieres' en utilisant la fonction 'retEtudiant' et afficher un message d'erreur lorsque la valeur '-1' est retournée ;

2.5.3 Refaire l'exercice mais au lieu de retourner '-1', lever une exception que vous créer pour cet étudiant non trouvé.

2.6 Procédure complexe

2.6.1 Créer une procédure 'affBulletins' qui pour un étudiant donné, affiche l'intitulé de chaque parcours (avec l'année d'inscription croissante) qu'il a suivi et pour chaque parcours les résultats obtenus pour chaque matière (avec intitulé) ;

2.6.2 Créer une procédure 'affCours' qui affiche les informations d'un cours à une date donnée[1] selon le schéma suivant :
HeureDeb, HeureFin, Salle/Batiment, Groupe1/Groupe2/Groupe3...

2.6.3 Créer une procédure 'affPlanning' qui affiche le planning d'un parcours donné (niveau, intitulé, annee), organisé par cours ;

1. Pour la vérification d'une date, il faut utiliser : trunc(dateHeureDeb) = to_date(inCurDate, 'DD/MM/YYYY')

Chapitre 3

Déclencheurs

Les déclencheurs (Trigger) sont des programmes déclenchés sur événements de la base de données (mises à jour, modifications, création, interactions...). Un déclencheur doit être associé à un objet de la base de données (table, index, tablespace, base de données...).

Pour créer ce trigger, vous pouvez dans le menu cliquer sur "déclencheurs' (click droit) et "nouveau déclencheur". Choisissez la table dont provient l'événement, un nom pour le trigger (explicite si possible), voir si le trigger doit être déclenché avant ou après l'événement, le type d'événement (UPDATE/INSERT/DELETE). Une fois fait, le template du trigger est créé et vous pouvez remplir le programme.

Une fois fait, le trigger sera déclenché à chaque événement sur la table concerné. Pour pouvoir vérifier le bon fonctionnement de celui-ci, veuillez trouver associé à chaque question, un jeu de test (requêtes de mises à jour) permettant de soulever (ou pas) une erreur. Vous devrez les exécuter pour constater le déclenchement du trigger.

3.1 Triggers

On souhaite donc créer à la base de données les contraintes qui ne sont pas exprimable sous forme de clés primaires, clés étrangères, types, domaines de valeurs ou CHECK. Créer les Trigger correspondant aux contraintes suivantes :

3.1.1 Le responsable d'un Cours est un enseignant titulaire ;

```
DELETE FROM Cours WHERE idCours = 10;
DELETE FROM Cours WHERE idCours = 11;
SELECT * FROM Cours;
-- Requête pour un Vacataire
INSERT INTO Cours VALUES (10,'pratique de mysql_Boisson', 3,'M1',20,2);
SHOW ERRORS;
-- Requête pour un Titulaire
INSERT INTO Cours VALUES (11,'pratique de mysql_Mourrier', 2,'M1',20,2);
SHOW ERRORS;
SELECT * FROM Cours;
```

3.1.2 L'étage d'une salle ne peut être supérieur au nombre d'étages du batiment ;

```
DELETE FROM Salle WHERE idSalle = 7;
DELETE FROM Salle WHERE idSalle = 8;
INSERT INTO Salle VALUES (7, 1, 10, 1, 30, 'TD');
SHOW ERRORS;
INSERT INTO Salle VALUES (8, 1, 2, 1, 35, 'TD');
SHOW ERRORS;
SELECT * FROM Salle;
```

3.1.3 On ne peut avoir de séances de 'Cours' dans une salle de 'TP', et on ne peut avoir de 'TP' que dans des salles de 'TP' ;

```
DELETE FROM seance WHERE TRUNC(dateHeureDeb) = TO_DATE('2015-10-15', 'YYYY-MM-DD');
SELECT * FROM seance WHERE TRUNC(dateHeureDeb) = TO_DATE('2015-10-15', 'YYYY-MM-DD');
-- cours CM en salle de TP
INSERT INTO Seance VALUES (200,1,TO_DATE('2015-10-15 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),90,5,1,'Cours');
-- cours TP en salle de amphi
INSERT INTO Seance VALUES (201,1,TO_DATE('2015-10-15 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),90,2,1,'TP');
-- cours CM en salle de cours
INSERT INTO Seance VALUES (201,1,TO_DATE('2015-10-15 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),90,2,1,'Cours');
SELECT * FROM Seance WHERE TRUNC(dateHeureDeb) = TO_DATE('2015-10-15', 'YYYY-MM-DD');
```

3.1.4 Deux séances de cours ne peuvent avoir lieu dans la même salle en même temps ;

```
DELETE FROM seance WHERE TRUNC(dateHeureDeb) = TO_DATE('2017-10-15', 'YYYY-MM-DD');
SELECT idSeance, idCours, TO_CHAR(dateHeureDeb, 'YYYY-MM-DD HH24:MI:SS') as Debut,
    TO_CHAR(dateHeureDeb + Duree/24/60, 'YYYY-MM-DD HH24:MI:SS') as Fin
FROM seance WHERE TRUNC(dateHeureDeb) = TO_DATE('2017-10-15', 'YYYY-MM-DD');
INSERT INTO Seance VALUES (201,1,TO_DATE('2017-10-15 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),120,2,1,'Cours');
INSERT INTO Seance VALUES (202,1,TO_DATE('2017-10-15 10:00:00', 'YYYY-MM-DD HH24:MI:SS'),120,2,1,'Cours');
INSERT INTO Seance VALUES (203,1,TO_DATE('2017-10-15 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),90,2,1,'Cours');
INSERT INTO Seance VALUES (204,1,TO_DATE('2017-10-15 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),240,2,1,'Cours');
INSERT INTO Seance VALUES (204,1,TO_DATE('2017-10-15 10:00:00', 'YYYY-MM-DD HH24:MI:SS'),30,2,1,'Cours');
INSERT INTO Seance VALUES (204,1,TO_DATE('2017-10-15 11:00:01', 'YYYY-MM-DD HH24:MI:SS'),90,2,1,'Cours');
```

3.1.5 Pour une année civile, la somme des durées des séances ne peut dépasser le nombre d'heures maximum du cours associé ;

```
/* restauration de l'état initial*/
UPDATE Cours set nbHeuresMax = 8 WHERE idCours = 1;
DELETE FROM Seance WHERE TRUNC(dateHeureDeb) = TO_DATE('2017-10-15', 'YYYY-MM-DD');
/* Pour tester le Trigger */
UPDATE Cours set nbHeuresMax = 6 WHERE idCours = 1;
INSERT INTO Seance VALUES (201,1,TO_DATE('2017-10-15 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),180,2,1,'Cours');
INSERT INTO Seance VALUES (202,1,TO_DATE('2017-10-15 11:30:00', 'YYYY-MM-DD HH24:MI:SS'),180,2,1,'Cours');
INSERT INTO Seance VALUES (203,1,TO_DATE('2017-10-15 15:00:00', 'YYYY-MM-DD HH24:MI:SS'),180,2,1,'Cours');
```

3.2 Triggers supplémentaires

3.2.1 Le service des enseignants vacataires ne peut dépasser 60 heures d'enseignement équivalent TD (1h de cours = 1.5h TD, 1.5h de TP = 1h TD) ;

3.2.2 Le service des enseignants moniteurs ne peut dépasser 96h de TD ;

3.2.3 La moyenne associée à une inscription est égale à la moyenne pondérée des résultats de l'élève obtenus à ce parcours cette année là ;

3.2.4 Pour une séance de cours, le nombre d'étudiants présents (cf. groupe) ne peut dépasser la capacité de la salle ;

Chapitre 4

Concurrence - Travaux Pratiques

Ce TP de concurrence devra être réalisé sous la base de données **MySQL** avec le moteur de stockage InnoDB (le seul qui soit capable de supporter tous les niveaux d'isolation).

Pour ce faire, il vous faut vous connecter sous l'environnement *WampServer* avec la console (bouton gauche sur l'icone de WampServer, puis 'MySQL', enfin 'Console'), ou mysql sous linux.

4.1 But du Travaux Pratique

Dans ce TP, vous devrez trouver pour chaquer niveau d'isolation, un enchainement d'opérations (une histoire) dans 2 sessions différentes (deux terminaux) entre deux transactions [1], pour mettre en valeur l'erreur/le cas demandé.

4.1.1 Mode `READ UNCOMMITTED` : Lecture sale

4.1.2 Mode `READ COMMITTED` [2] : Ecriture sale

4.1.3 Mode `REPEATABLE READ` : Tuple fantôme

4.1.4 Mode `SERIALISABLE` [3] : *Dead lock*

4.2 Instructions

Pour ce faire, vous aurez besoin de :

4.2.1 Pour pouvoir constater des problèmes de concurrences, il vous faut deux sessions différentes. Il vous faudra donc ouvrir 2 terminaux sous mysql. Pour les différencier, nous utiliserons la commande :
'`PROMPT SESSION1> `' (resp SESSION2).

4.2.2 Pour chaque session, il faut enlever le mode de validation automatique :
`SET AUTOCOMMIT = 0;`

4.2.3 Lorsque vous devrez changer de mode d'isolation :

- `SET SESSION TRANSACTION ISOLATION LEVEL` **READ UNCOMMITTED**;
- `SET SESSION TRANSACTION ISOLATION LEVEL` **READ COMMITTED**;
- `SET SESSION TRANSACTION ISOLATION LEVEL` **REPEATABLE READ**;
- `SET SESSION TRANSACTION ISOLATION LEVEL` **SERIALIZABLE**;

4.3 Schéma de données et Données

Schéma Pour tester la concurrence entre deux transactions, nous allons utiliser les deux tables suivantes :

1. Ne pas oublier de supprimer le mode auto-commit et ni de vérifier le niveau d'isolation dans CHAQUE session
2. Sous MySQL, la lecture sale est résolue grâce au *versioning*, qui permet de garder l'image de la donnée avant la transaction
3. Comme MySQL est en versionning, le deadlock ne peut arriver que sur une histoire de la sorte $w_1(x)w_2(y)w_1(y)w_2(x)$

```
USE TEST; -- Sélectionne la base de données
DROP TABLE Spectacle; -- supprime les tables si elles existaient déjà
DROP TABLE Client;
DROP TABLE Reservation;

CREATE TABLE Spectacle (id_spectacle INT NOT NULL PRIMARY KEY,
                        places_offertes INT NOT NULL,
                        places_libres INT NOT NULL,
                        tarif DECIMAL(10,2) NOT NULL
            ) ENGINE=InnoDB;

CREATE TABLE Client (id_client INT NOT NULL PRIMARY KEY,
                     Solde FLOAT NOT NULL
            ) ENGINE=InnoDB;

CREATE TABLE Reservation (id_client INT NOT NULL,
                          id_spectacle INT NOT NULL,
                          places_reservees INT NOT NULL,
                          PRIMARY KEY (id_client, id_spectacle),
                          KEY spectacle (id_spectacle)
            ) ENGINE=InnoDB;
```

Pour que la base de données reste cohérente, il faut pour cela que pour une spectacle donné : $sum(places\_reservees) = (places\_offertes - places\_libres)$.

Données L'état d'origine de notre base de données est donné par les requêtes suivantes :

```
DELETE FROM Reservation;
DELETE FROM Client;
DELETE FROM Spectacle;
INSERT INTO Client VALUES (1, 50);
INSERT INTO Client VALUES (2, 50);
INSERT INTO Spectacle VALUES (1, 250, 250, 20);
COMMIT;
SET SESSION TRANSACTION ISOLATION LEVEL ... ;
```

Entre chaque test de concurrence sur notre schéma, la base de données doit avoir cet état pour être cohérent. Ainsi, vous pourrez mettre à zéro la base en exécutant ce script.

4.4 Transactions

Voici le jeu de transactions qui vous permettra de mettre en valeur les erreurs ou cas demandés (section 4.1).

Attention ! Une **séquence de requêtes n'est pas interchangeable**. Les séquences ne sont pas intégrées dans des procédures, pour permettre de décomposer les opérations dans le temps et favoriser la concurrence. Vous devrez donc vérifier vous-même les conditions (T_1 et T_2) avec les valeurs LOCALES '@' à la session, et le cas échéant, faire un ROLLBACK et arrêter la transaction.

Pour faciliter la découverte d'erreur, nous vous invitons à modéliser chaque transaction sous forme de séquences d'opérations :

4.4.1 Donner pour chaque transaction les séquences d'opérations possibles. Noter, le cas échéant, les conditions d'application de cette transaction (exemple : @nb_places < 2) ;

4.4.2 Présenter par la suite, chaque histoire sous forme de suite d'opérations grâce aux transactions que vous venez de produire[4].

T_1 **Réservation de 2 places pour le client 1**

4. Tester sous MySQL pour vous assurer du bon fonctionnement

$R_1(sp_1)$ SELECT places_libres, tarif INTO @nb_libres, @tarif
FROM Spectacle WHERE id_spectacle = 1 ;
r_1 *Vérifier si "SELECT @nb_libres - 2 ;" < 0 alors ROLLBACK ;*
$W_1(sp_1)$ UPDATE Spectacle SET places_libres = @nb_libres - 2 WHERE id_spectacle = 1 ;
$W_1(re_1)$ INSERT INTO Reservation VALUES (1, 1, 2) ;
$R_1(so_1)$ SELECT Solde INTO @solde FROM Client WHERE id_client = 1 ;
r_1 *Vérifier si "SELECT @solde - 2 * @tarif ;" < 0 alors ROLLBACK ;*
$W_1(cl_1)$ UPDATE Client SET Solde = @solde - 2 * @tarif WHERE id_client = 1 ;
c_1 COMMIT ;

T_2 **Réservation de 5 places pour le client 2**

$R_2(sp_1)$ SELECT places_libres, tarif INTO @nb_libres, @tarif
FROM Spectacle WHERE id_spectacle = 1 ;
r_2 *Vérifier si "SELECT @nb_libres - 5 ;" < 0 alors ROLLBACK ;*
$W_2(sp_1)$ UPDATE Spectacle SET places_libres = @nb_libres - 5 WHERE id_spectacle = 1 ;
$W_2(re_2)$ INSERT INTO Reservation VALUES (2, 1, 5) ;
$R_2(so_2)$ SELECT Solde INTO @solde FROM Client WHERE id_client = 2 ;
r_1 *Vérifier si "SELECT @solde - 5 * @tarif ;" < 0 alors ROLLBACK ;*
$W_2(cl_2)$ UPDATE Client SET Solde = @solde - 5 * @tarif WHERE id_client = 2 ;
c_2 COMMIT ;

T_3 **Vérification du nombre de places réservées**

$R_3(re_{1,2})$ SELECT SUM(places_reservees) AS places_reservation
FROM Reservation WHERE id_spectacle = 1 ;
$R_3(sp_1)$ SELECT (places_offertes - places_libres) AS places_spectacle
FROM Spectacle WHERE id_spectacle = 1 ;
c_3 COMMIT ;

T_4 **Mise à jour des places pour spectacle et client**

$W_4(sp_1)$ UPDATE Reservation SET places_reservees = places_reservees + 10
WHERE id_client = 2 AND id_spectacle = 1 ;
$W_4(cl_1)$ UPDATE Spectacle SET places_offertes = places_offertes + 10, places_libres = places_libres + 10
WHERE id_spectacle = 1 ;
c_4 COMMIT ;