le cnam

# Bases de Données

Exercices et Travaux Pratiques

DUT Informatique 2° année

**Cnam**
nicolas.travers@cnam.fr

# Chapitre

# Table des matières

# Chapitre 1

# Création d'une Base de Données

## 1.1 Analyse des besoins

Soit le texte suivant décrivant les besoins pour une base de données gérant les enseignements d'un établissement d'enseignement supérieur.

> Nous avons besoin d'une base de données permettant de gérer les enseignements de notre établissement.
> Il faut prendre en compte les étudiants avec leur nom, prénom, date de naissance et numéro d'identification, qui lors de leurs études dans le supérieur s'inscrivent à différents parcours (L1, L2, L3, M1, M2). Ces parcours ont des dénominations, sont reliés entre eux (représentant les possibilités de passage entre parcours) et sont composés d'un ensemble de cours. A noter que des étudiants de différents parcours peuvent suivre un même cours qui doit correspondre au niveau du parcours associé.
> Ces mêmes cours chaque année ont un enseignant responsable, un coefficient et la moyenne des notes des étudiants. Cette moyenne sera calculée en fin d'année par la moyenne des notes obtenues par les étudiants à ce cours. La moyenne des notes d'un étudiant pondérée par les coefficients sera calculée pour donner une mention à son parcours (Refusé/ Passable/Assez bien/Bien/Très bien).
> Afin de gérer la réservation des salles pour chaque TD d'un parcours, des groupes sont créés dans lesquels seront répartis les étudiants. Les séances de cours forment un emploi du temps qui se verront associer la plage horaire, le type (cours/TD /TP) et une salle attribuée. Les salles sont localisées par un étage et un numéro dans un bâtiment avec une capacité maximale et un type (Amphi, TD, TP).
> Les enseignants interviennent dans différentes séances, il faut prendre en compte, chaque année, leur service en nombre d' heures de cours, de TD et de TP qu'il effectue. Un enseignant peut être titulaire, moniteur ou vacataire

*1.1.1* Donner les acteurs principaux de ce texte ;

*1.1.2* Pour chaque acteur, définir leurs caractéristiques ;

## 1.2 Modèle Entité/Association avec *jMerise*

*1.2.1* Créer les entités correspondantes aux points précédents ;

*1.2.2* Relier les entités à l'aide d'associations ;

## 1.3 Schéma relationnel

*1.3.1* Transformer le schéma E/A en schéma relationnel ;

*1.3.2* Tester la génération automatique du *MPD* avec *jMerise*. Pour cela, cliquer sur la roue crantée bleue, puis sélectionner le 'MPD' ;

*1.3.3* Définir les dépendances fonctionnelles dans chaque Relation ;

*1.3.4* Vérifier si le schéma généré est en 3° forme normale ;

## 1.4 Contraintes

*1.4.1* Donner la clé primaire de chaque relation ;

*1.4.2* Donner la/les clés étrangères de chaque relation ;

*1.4.3* Donner les contraintes de domaines spécifiques à certains attributs ;

## 1.5 LDD : Langage de Description de Données

*1.5.1* Donner les requêtes SQL de création de tables pour : Etudiant, Parcours et Inscription ;

*1.5.2* Générer les requêtes SQL avec *jMerise*. Vérifier le contenu et modifier le si besoin (le code de ce schéma physique est généré pour MySQL).

## 1.6 Création de la base de données avec *Oracle Express©* et *SQLDeveloper*

Pour le bon fonctionnement de ce TP, les deux outils *Oracle Express©* et *SQLDeveloper* doivent être préalablement installés. Veuillez pour cela vous référer au guide d'installation relatif à votre système d'exploitation

*Oracle© Express* est le serveur (SGBD) permettant de gérer l'ensemble des données et son stockage. *SQLDeveloper* est une IHM permettant de se connecter à la base de données et de faire des requêtes. Nous nous servirons pour chaque TP de cet environnement de travail.

Pour installer le serveur Oracle :

— Sous Windows, télécharger Oracle Express (créer un compte Oracle si besoin)

— Sous Docker, télécharger l'image du serveur avec la commande suivante : **sudo docker run -d −−shm-size=2g -p 1521 :1521 −−name tp-oracle traversn/tp-oracle-xe11g-bdd-ecole /start.sh**

.

Si vous avez installé votre serveur (sans Docker), il faut mettre en place l'espace de travail :

*1.6.1* Une fois le serveur *Oracle* démarré, et SQLDeveloper lancé, connectez-vous avec 'test_user' (la création de cet utilisateur se fait grâce au script 0_creation.sql) ;

*1.6.2* Téléchargez les scripts [1] de création de la base de données et exécutez les dans une console '*SQL*'.

1. Disponible ici :`https://chewbii.com/wp-content/uploads/2017/09/BD_TP.zip`. Les fichiers dédiés à la base de données sont : *1_BDD.sql* (schéma), *2_DATA.sql* (données)

# Chapitre 2

# 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.

## 2.1 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 :

*2.1.1* 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;
```

*2.1.2* 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;
```

*2.1.3* 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');
```

*2.1.4* 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');
```

*2.1.5* 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');
```

## 2.2 Triggers supplémentaires

*2.2.1* 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) ;

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

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

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

# Chapitre 3

# 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.

## 3.1 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.

*3.1.1* Mode `READ UNCOMMITTED` : Lecture sale

*3.1.2* Mode `READ COMMITTED` [2] : Ecriture sale

*3.1.3* Mode `REPEATABLE READ` : Tuple fantôme

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

## 3.2 Instructions

Pour ce faire, vous aurez besoin de :

*3.2.1* 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).

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

*3.2.3* 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**;

## 3.3 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.

## 3.4 Transactions

Voici le jeu de transactions qui vous permettra de mettre en valeur les erreurs ou cas demandés (section 3.1).

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 :

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

*3.4.2* 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 ;

# Chapitre 4

# Création d'une application BD

Nous cherchons ici à développer une application gérant différents besoins du système d'information :

— Rôle étudiant : Consultation du planning ;
— Rôle Enseignant : Consulter son planning, consulter son service ;
— Rôle Administratif : Consulter les parcours, les étudiants, ajouter/modifier une note ;

## 4.1 Connexion à la base de données

Créer un objet gérant la connexion à la base de données. Il vous servira dans chacun des programmes suivants.

**Entrainement** : Créer un programme affichant la liste des intitulés de parcours.

## 4.2 Rôle étudiant

Créer une interface graphique qui affiche l'emploi du temps (tableau) d'un parcours (paramètre d'entrée avec année) à partir des informations suivantes :

— Sélectionner un parcours ⇒ liste de tous les cours de ce parcours, trié par date+heure.
Ne pas oublier d'afficher l'intitulé du cours et la salle ;
— Sélectionner en plus un numéro de semaine (liste déroulante) ;

Conseil : (non oglibatoire)

— Créer un objet stockant les informations d'un parcours (id, intitulé, niveau, cours) ;
— Créer un objet stockant les informations d'une séance de cours (id, intitulé, groupe(s), salle, type, jour, horaires) ;
— Créer un objet stockant les informations d'une journée de cours ;
— Créer un objet stockant les informations d'une semaine de cours ;
— Créer un objet qui interroge la base de données et récupère l'ensemble des cours du parcours pour une semaine donnée ;
— Afficher d'abord la liste complète de tous les cours ;
— Découper par journée ;

## 4.3 Rôle enseignant

Créer une interface graphique qui affiche les informations de l'enseignant avec son service (somme des heures enseignées), et un onglet permettant de consulter son planning (même design que pour l'étudiant) ;

Conseil : (non obligatoire)

— Créer un objet stockant les informations d'un enseignant (id, nom, prénom) ;
— Créer un objet stockant l'ensemble des services d'un enseignant (le relier à l'objet précédent) ;
— Créer un objet qui interroge la base de données et récupère l'ensemble du service d'un enseignant ;
— Créer un objet qui interroge la base de données et récupère l'ensemble des cours dans lequel il intervient ;
— Requête pour le service + Requête pour le planning (idem que pour parcours, mais avec idEnseignant)

## 4.4 Bonus : Rôle administratif

Cette vue n'est pas demandée en TP. Toutefois, vous pouvez la faire pour vous entraîner.

Créer une interface graphique permettant de créer un parcours et de modifier les informations de celui-ci (associer un enseignement, un enseignant, un passage) ;

Conseil :

*4.4.1* Créer un objet de gestion d'un étudiant (informations, inscriptions, résultats) ;

*4.4.2* Créer une interface graphique permettant de consulter les informations de l'ensemble des étudiants d'un parcours ;

*4.4.3* Créer une interface graphique permettant de modifier/ajouter/supprimer les informations concernant un étudiant (ou créer un étudiant) ;

*4.4.4* Créer un objet de gestion d'un parcours (informations, enseignements, passages) ;