le cnam

Bases de Données Avancées

Exercices et Travaux Pratiques

DUT Informatique 2° année

nicolas.travers@cnam.fr

Chapitre

Table des matières

1 BDD **3**
- 1.1 Analyse des besoins . . . 3
- 1.2 Modèle Entité/Association . . . 3
- 1.3 Schéma relationnel . . . 3
- 1.4 Contraintes . . . 3
- 1.5 LDD : Langage de Description de Données . . . 3

2 SQLDeveloper **4**
- 2.1 Modélisation . . . 4
- 2.2 Création de la base de données . . . 4

3 Trigger **5**
- 3.1 . . . 6
- 3.2 . . . 6
- 3.3 . . . 6
- 3.4 . . . 7
- 3.5 . . . 8

4 Concurrence - Travaux Pratiques **11**
- 4.1 L'environnement de travail . . . 11
- 4.2 Schéma de données et Données . . . 11
- 4.3 Transactions . . . 12
- 4.4 Concurrence . . . 13

5 Création d'une application BD **14**
- 5.1 Connexion à la base de données . . . 14
- 5.2 Rôle étudiant . . . 14
- 5.3 Rôle enseignant . . . 14
- 5.4 Bonus : Rôle administratif . . . 14

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

1.2.1 Créer le schéma Entité/Association correspondant ;

1.3 Schéma relationnel

1.3.1 Transformer le schéma E/A en schéma relationnel ;

1.3.2 Définir les dépendances fonctionnelles dans chaque Relation ;

1.3.3 Normaliser 3ř 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 ;

1.5 LDD : Langage de Description de Données

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

Chapitre 2

Mise en pratique avec SQLDeveloper

Pour la suite des TPs, l'utilisation de `Oracle Express` et `SQLDeveloper` est requise. Veuillez suivre le guide d'installation/connexion disponible sur `Claroline` "Oracle_SQLDeveloper_Guide.pdf".

2.1 Modélisation

A l'aide de *DataModeler*[1], créer le modèle des tables à partir du schéma Entité/Association réalisé à la question 1.2. Pour cela, il faut ouvrir un nouveau modèle (Menu "File/New Model"), une fois la fenêtre ouverte double-click sur "Add diagram". Vous pouvez maintenant créer les entités et les cardinalités qui les relient.

2.2 Création de la base de données

Générer les scripts de création de la Base de Données (cf. Guide / DataModeler). Enregister le fichier, puis fermer le modèle.

Lancer le script à l'aide d'une console SQL de SQLDeveloper (cf. Guide / SQL). Pour exécuter l'ensemble des requêtes utilisez la touche F5.

1. *DataModeler* est un outils de SQLDeveloper, le guide détails les fonctionnalités utiles dans le cadre de ce TP

Chapitre 3

Déclencheurs

Afin aller sur la fenêtre de création d'un TRIGGER pour une table, il faut sélectionner la table dans la base de données (menu schéma en bas à gauche), faire click-droit sur la table et '*Alter Table*' et choisir l'onglet '*Triggers*' (en bas de la fenêtre).

Pour créer le Trigger, sélectionner le mode de déclenchement (AFTER DELETE/AFTER UPDATE/AFTER INSERT...) et clicker sur '*Add Trigger*'. Il faut ensuite préciser le nom du Trigger et la procédure à exécuter. Une fois le Trigger terminé, clicker sur '*Apply*'. Les jeux de tests de chaque trigger est founi après chaque contrainte ci-dessous.

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

— Le responsable d'un enseignement est un enseignant titulaire ;

```
delete from enseignement where idEnseignement = 10;
delete from enseignement where idEnseignement = 11;
select * from enseignement;

insert into enseignement values (10,'pratique de mysql_Boisson', 'Boisson','M1',20,2);
show errors;
insert into enseignement values (11,'pratique de mysql_Mourrier', 'Mourier','M1',20,2);
show errors;

select * from enseignement;
```

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

— 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 cours where date = '2014-10-15' limit 3;
select * from cours where date = '2014-10-15';

-- cours CM en salle de TP
insert into cours values (1,1,'2014-10-15','09 :00 :00','01 :30 :00',5,1,'Cours');
show errors;

-- cours TP en salle de amphi
insert into cours values (1,1,'2014-10-15','10 :00 :00','01 :30 :00',2,1,'TP');
show errors;

-- cours CM en salle de cours
insert into cours values (1,1,'2014-10-15','11 :00 :00','01 :30 :00',2,1,'Cours');
show errors;

select * from cours where date = '2014-10-15';
```

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

```
delete from cours where date = '2014-10-15' limit 6;

-- cours CM en salle de cours (OK)
select * from cours where date = '2014-10-15';
insert into cours values (1,1,'2014-10-15','10 :00 :00','01 :30 :00',2,1,'Cours');

-- cours CM en salle de cours (salle déjà occupée)
select * from cours where date = '2014-10-15';
insert into cours values (1,1,'2014-10-15','10 :30 :00','01 :30 :00',2,1,'Cours');

select * from cours where date = '2014-10-15';
```

=======

3.1 Le responsable d'un enseignement est un enseignant titulaire

Jeu de test :

```
delete from enseignement where idEnseignement = 10;
delete from enseignement where idEnseignement = 11;

select * from enseignement

call insert_into_enseignement (10,'pratique de mysql_Boisson', 'Boisson','M1',20,2);
show errors;
call insert_into_enseignement (11,'pratique de mysql_Mourrier', 'Mourier','M1',20,2);
show errors;

select * from enseignement;
```

la procédure 'insert_into_enseignement' est définie ci-dessous :

```
DROP PROCEDURE IF EXISTS insert_into_enseignement;
DELIMITER
CREATE PROCEDURE `insert_into_enseignement`
        (id Integer,intitule varchar (45),nomResp varchar (45),niveau varchar (45),duree integer,coeff integer)
BEGIN
        Declare continue handler for 1644 /* insertion non effectuée */
                Set @flag1644 = false; /* false si insertion non effectuée */
        select idEnseignant into @idResp from Enseignant where Nom = nomResp ;
        set @flag1644 = true;
        insert into enseignement values (id,intitule,@idResp,niveau,duree,coeff) ;
        if (@flag1644)then
                select (concat('insertion du cours <' , intitule , '> réussie '));
        else
                select (concat('insertion du cours <' , intitule , '> refusée mais l exécution peut se poursuivre '));
        end if;
END

DELIMITER ;
```

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

3.3 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'

Jeu de test :

```
/* procedure d'insertion avec traitement des refus d'insertion dans Mysql
-------------------------------------------------------------------------- */
DROP PROCEDURE IF EXISTS insert_into_cours;
DELIMITER
CREATE PROCEDURE `insert_into_cours`
        (idEns integer, idGroupe integer, date date, heure time, duree time ,idSalle integer,idE integer,type varchar (20))
BEGIN
        declare continue handler for sqlstate '45001'
                set @flag_etat = 45001;
        declare continue handler for sqlstate '45002'
                set @flag_etat = 45002;

        set @flag_etat = 0;
        insert into cours values (idEns,idGroupe, date, heure, duree,idSalle,idE,type) ;
```

```
        if (@flag_etat = 45001) then
                select (concat('insertion du cours refusée <' , @flag_etat , '>'));
        elseif (@flag_etat = 45002) then
                select (concat('insertion du cours refusée <' , @flag_etat , '>'));
        end if;
END

DELIMITER ;

/* Requêtes de test */
delete from cours where date = '2014-10-15' limit 3;
select * from cours where date = '2014-10-15';

-- cours CM en salle de TP
call insert_into_cours (1,1,'2014-10-15','09 :00 :00','01 :30 :00',5,1,'Cours');
show errors;

-- cours TP en salle de amphi
call insert_into_cours (1,1,'2014-10-15','10 :00 :00','01 :30 :00',2,1,'TP');
show errors;

-- cours CM en salle de cours
call insert_into_cours (1,1,'2014-10-15','11 :00 :00','01 :30 :00',2,1,'Cours');
show errors;

select * from cours where date = '2014-10-15';
```

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

Jeu de test :

```
/* procédure d'insertion de la contrainte 3
modifiée pour ajout de la contrainte 4
------------------------------------------------------------------------------ */

DROP PROCEDURE IF EXISTS insert_into_cours;
DELIMITER
CREATE PROCEDURE `insert_into_cours`
        (idEns integer, idGroupe integer, date date, heure time, duree time ,idSalle integer,idE integer,type varchar (20))
BEGIN
        declare continue handler for sqlstate '45001' /*Cours en salle de TP*/
                set @flag_etat = 45001;
        declare continue handler for sqlstate '45002' /*TP ailleurs qu'en salle de TP*/
                set @flag_etat = 45002;
        declare continue handler for sqlstate '45003' /*salle déjà occupée*/
                set @flag_etat = 45003;

        set @flag_etat = 0;
        insert into cours values (idEns,idGroupe, date, heure, duree,idSalle,idE,type) ;

        if (@flag_etat = 45001) then
                select S.type into @v_type from Salle S WHERE S.idSalle = idSalle ;
                select (concat('insertion du cours refusée <' , @flag_etat , '>',type,' en salle de ',@v_type));
        elseif (@flag_etat = 45002) then
                select S.type into @v_type from Salle S WHERE S.idSalle = idSalle ;
                select (concat('insertion du cours refusée <' , @flag_etat , '>',type, ' dans une salle de ',@v_type));
        elseif (@flag_etat = 45003) then
                select * from cours C where C.date = date;
                select (concat('insertion du cours de ', heure ,' refusée <' , @flag_etat , '> ','salle ',idSalle ,' déjà occupée'));
```

```
        end if;
END

DELIMITER ;

/* Requêtes de test
----------------*/
delete from cours where date = '2014-10-15' limit 6;

-- cours CM en salle de cours (OK)
select * from cours where date = '2014-10-15';
call insert_into_cours (1,1,'2014-10-15','10 :00 :00','01 :30 :00',2,1,'Cours');

-- cours CM en salle de cours (salle déjà occupée)
select * from cours where date = '2014-10-15';
call insert_into_cours (1,1,'2014-10-15','10 :30 :00','01 :30 :00',2,1,'Cours');

select * from cours where date = '2014-10-15';
```

3.5 Le total cumulé des séances de cours ne peut dépasser le nombre d'heures maximum d'un enseignement

Jeu de test :

```
/*Procedure d'insertion sans cours avec contraintes 3,4,5*/
/*-----------------------------------------------------------*/
DROP PROCEDURE IF EXISTS insert_into_cours;
DELIMITER
CREATE PROCEDURE `insert_into_cours`(idEns integer, idGroupe integer, date date, heure time, duree time ,idSalle integer,idE
    integer,type varchar (20))
BEGIN

declare continue handler for sqlstate '45001' /*Cours en salle de TP*/
        set @flag_etat = 45001;
declare continue handler for sqlstate '45002' /*TP ailleurs qu'en salle de TP*/
        set @flag_etat = 45002;
declare continue handler for sqlstate '45003' /*salle déjà occupée*/
        set @flag_etat = 45003;
declare continue handler for sqlstate '45004' /*trop heures de cours*/
        set @flag_etat = 45004;

set @flag_etat = 0;
insert into cours values (idEns,idGroupe, date, heure, duree,idSalle,idE,type) ;

case @flag_etat
        when 45001 then
                select S.type into @v_type from Salle S WHERE S.idSalle = idSalle ;
                select (concat('insertion du cours refusée <' , @flag_etat , '>',type,' en salle de ',@v_type));
        when 45002 then
                select S.type into @v_type from Salle S WHERE S.idSalle = idSalle ;
                select (concat('insertion du cours refusée <' , @flag_etat , '>',type, ' dans une salle de ',@v_type));
        when 45003 then
                select (concat('insertion du cours de ', heure ,' refusée <' , @flag_etat , '> ','salle ',idSalle ,' déjà occupée'));
        when 45004 then
                select E.intitule into @v_intitule from Enseignement E where E.idEnseignement = idEns;
                select (concat('insertion du cours de ', heure ,' refusée <' , @flag_etat , '> ','trop d heures de cours pour l enseignement ',
                    @v_intitule));
end case;

END
```

```
DELIMITER ;

/* Requêtes de Test */
>>>>>>> .r321

/* restauration de l'état initial*/
update Enseignement set nbHeuresMax = 30 where idEnseignement = 2;
delete from cours where date ='2014-10-16' limit 2;

/*choix et préparation d'un enseignement pour le test ==> enseignement nř2 */
select C.idEnseignement,E.Intitule, E.nbHeuresMax, sec_to_time(sum(time_to_sec(C.duree))) AS duree_des_cours
        from cours C , Enseignement E
        Where C.idEnseignement = E.idEnseignement
        group by C.idEnseignement ;

        update Enseignement set nbHeuresMax = 37 where idEnseignement = 2;

select C.idEnseignement,E.Intitule, E.nbHeuresMax, sec_to_time(sum(time_to_sec(C.duree))) AS duree_des_cours
        from cours C , Enseignement E
        Where C.idEnseignement = E.idEnseignement
        group by C.idEnseignement ;

/* choix des cours à insérer et à modifier*/
select C.idEnseignement,E.Intitule, E.nbHeuresMax, sec_to_time(sum(time_to_sec(C.duree))) AS duree_des_cours
        from cours C , Enseignement E
        Where C.idEnseignement = E.idEnseignement
                and C.idEnseignement = 2;

select idEnseignement, date, sec_to_time(sum(time_to_sec(duree))) AS duree_des_cours from cours where idEnseignement = 2 group
    by date;
select idEnseignement, date, duree from cours where idEnseignement = 2 and date ='2014-09-16';


/*----------------------------*/
/*test du trigger d'insertion*/
/*----------------------------*/
call insert_into_cours (2,5,'2014-10-16','10 :00 :00','02 :00 :00',2,2,'Cours');

/*remise à l'état initial du cours pour test du trigger d'update*/
update cours set duree = '01 :30 :00'
        where idEnseignement = 2
                and date = '2014-09-16'
                and heureDeb= '16 :30 :00'
                and idGroupe =6;

select idEnseignement, date, sec_to_time(sum(time_to_sec(duree))) AS duree_des_cours from cours where idEnseignement = 2 group
    by date;
select idEnseignement, date, heureDeb from cours where idEnseignement = 2 and date = '2014-09-16';

/*-------------------------*/
/*test du trigger d'update*/
/*-------------------------*/

select C.duree, C.idSalle,C.idEnseignant, C.type into @duree, @salle,@idE,@type
from cours C
where idEnseignement =2
        and idGroupe = 6
        and date = '2014-09-16'
        and heureDeb= '16 :30 :00';

call update_cours (2,6,'2014-09-16','16 :30 :00',ADDTIME(@duree, '04 :00 :00'), @salle,@idE,@type);
```

```
select * from cours
where idEnseignement = 2
        and idGroupe = 6
        and date = '2014-09-16'
        and heureDeb= '16 :30 :00';
```

Chapitre 4

Concurrence - Travaux Pratiques

4.1 L'environnement de travail

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').

4.1.1 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).

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

4.1.3 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**;

4.2 Schéma de données et Données

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

```
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.

4.3 Transactions

Pour chaque transactions ci-dessous, 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.

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

4.3.2 Présenter par la suite, chaque histoire sous forme de suite d'opérations grâce aux transactions que vous venez de produire [1].

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

$R_1(sp_1)$	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;

1. Tester sous MySQL pour vous assurer du bon fonctionnement

4.4 Concurrence

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

4.4.1 Mode `READ UNCOMMITTED` : Lecture sale

4.4.2 Mode `READ COMMITTED` [3] : Ecriture sale

4.4.3 Mode `REPEATABLE READ` : Tuple fantôme

4.4.4 Mode `SERIALISABLE` [4] : *Dead lock*

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

Chapitre 5

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 ;

5.1 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.

5.2 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 ;

5.3 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)

5.4 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 :

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

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

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

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