

**Efrei L'3 Groupe C Promotion 2018**

**Adrien Poupa – Edgar Buob**

**Rapport projet Bases de données 2**

**SGBD choisi : MySQL**

## I – Gestion des vues

1. Créer la vue renfermant tous les étudiants ayant eu des épreuves en Informatique ainsi que les notes obtenues.

```
CREATE VIEW `NotesInformatiques` AS
SELECT `notation`.`NumEtu`, `epreuve`.`NumEpreuve`, `Note`, `Nom`,
`Prenom`, `DateNais`, `Rue`, `CP`, `Ville`, `DateEpreuve`, `Lieu`
FROM `notation`, `etudiant`, `epreuve`
WHERE `notation`.`numetu` = `etudiant`.`numetu` AND
`notation`.`numepreuve` = `epreuve`.`numepreuve` AND `CodeMat` = 'INF'
```

NumEtu	NumEpreuve	Note	Nom	Prenom	DateNais	Rue	CP	Ville	DateEpreuve	CodeMat
110	21031	11	Dupont	Albert	1990-06-01	Rue de Crimee	69001	Lyon	2003-10-30	Salle191L
222	21031	12	West	James	1993-09-03	Studio	NULL	Hollywood	2003-10-30	Salle191L
300	21031	20	Martin	Marie	1998-06-05	RuedesAcacias	69130	Ecully	2003-10-30	Salle191L
421	21031	2	Durand	Gaston	1990-11-15	Rue de Lyon	75002	Paris	2003-10-30	Salle191L
575	21031	13	Titgoutte	Justine	1995-02-28	CheminduChâteau	69630	Chaponost	2003-10-30	Salle191L
667	21031	9	Dupond	Noémie	1997-09-18	RuedeDôle	69007	Lyon	2003-10-30	Salle191L
110	21032	NULL	Dupont	Albert	1990-06-01	Rue de Crimee	69001	Lyon	2004-06-01	Salle192L
222	21032	16	West	James	1993-09-03	Studio	NULL	Hollywood	2004-06-01	Salle192L
300	21032	14	Martin	Marie	1998-06-05	RuedesAcacias	69130	Ecully	2004-06-01	Salle192L
421	21032	NULL	Durand	Gaston	1990-11-15	Rue de Lyon	75002	Paris	2004-06-01	Salle192L
575	21032	14	Titgoutte	Justine	1995-02-28	CheminduChâteau	69630	Chaponost	2004-06-01	Salle192L
667	21032	10	Dupond	Noémie	1997-09-18	RuedeDôle	69007	Lyon	2004-06-01	Salle192L

2. Donner la moyenne et le nombre d'épreuves en informatique de chaque étudiant ayant passé au moins une épreuve dans cette matière.

```
SELECT AVG(`note`) AS `moyenne`, COUNT(`note`) AS `Nombre epreuves`,
`prenom`, `nom`
FROM `notesinformatiques`
GROUP BY `numetu`
```

moyenne	Nombre epreuves	Prenom	Nom
9.0000	1	Albert	Dupont
14.0000	2	James	West
17.0000	2	Marie	Martin
2.0000	1	Gaston	Durand
13.5000	2	Justine	Titgoutte
9.5000	2	Noémie	Dupond

3. Donner le nombre d'étudiants ayant eu au moins une moyenne de 10 en Informatique

```
SELECT COUNT(*) FROM (
  SELECT AVG(`note`) AS `moyenne`, `prenom`, `nom`
  FROM `notesinformatiques`
  GROUP BY `numetu`
  HAVING moyenne >= 10) alias;
```

COUNT(*)
3

4. Donner les noms d'étudiants ainsi que leur moyenne ayant eu une moyenne supérieure ou égale à 10 en Informatique et classés par ordre de mérite.

```
SELECT AVG(`note`) AS `moyenne`, COUNT(`note`) AS `Nombre epreuves`,
`prenom`, `nom`
FROM `notesinformatiques`
GROUP BY `numetu`
HAVING moyenne >= 10
ORDER BY moyenne DESC
```

moyenne ▼	Nombre epreuves	Prenom	Nom
17.0000	2	Marie	Martin
14.0000	2	James	West
13.5000	2	Justine	Titgoutte

5. Donner les noms d'étudiants qui n'ont passé aucune épreuve en Informatique en utilisant les sous requêtes puis la jointure externe.

```
SELECT * FROM `etudiant` WHERE `NumEtu` NOT IN
(SELECT `NumEtu` FROM `notesinformatiques`)
```

ou

```
SELECT * FROM `etudiant` WHERE `NumEtu` NOT IN (SELECT `notation`.`NumEtu`
FROM `notation`, `etudiant`, `epreuve`
WHERE `notation`.`numetu` = `etudiant`.`numetu` AND
`notation`.`numepreuve` = `epreuve`.`numepreuve` AND `CodeMat` = 'INF')
```

NumEtu	Nom	Prenom	DateNais	Rue	CP	Ville
999	Phantom	Marcel	1990-01-30	NULL	NULL	NULL

## II – Gestion de la confidentialité (Vues et Droits d'accès)

1. Créer la vue *EtudiantsLyonnais* renfermant tous les étudiants lyonnais.

```
CREATE VIEW `EtudiantsLyonnais` AS SELECT * FROM `etudiant` WHERE `Ville`='Lyon'
```

NumEtu	Nom	Prenom	DateNais	Rue	CP	Ville
110	Dupont	Albert	1990-06-01	RuedeCrimee	69001	Lyon
421	Durand	Gaston	1990-11-15	RuedelaMeuse	69008	Lyon
667	Dupond	Noémie	1997-09-18	RuedeDôle	69007	Lyon

2. Donner la moyenne en informatique des étudiants lyonnais

```
SELECT AVG(`note`) AS `moyenne`
FROM (
    SELECT `notation`.`NumEtu`, `epreuve`.`NumEpreuve`, `Note`,
    `Nom`, `Prenom`, `DateNais`, `Rue`, `CP`, `DateEpreuve`, `Lieu`,
    `CodeMat`
    FROM `notation`, `EtudiantsLyonnais`, `epreuve`
    WHERE `notation`.`numetu` = `EtudiantsLyonnais`.`numetu` AND
    `notation`.`numepreuve` = `epreuve`.`numepreuve`
) AS `t`
WHERE `t`.`CodeMat`='INF'
```

moyenne
7.5000

3. L'étudiant Dupond a quitté l'établissement. Est-il possible de le supprimer depuis la vue *EtudiantsLyonnais*? Vérifier ?

On ne peut pas faire :

```
DELETE FROM `etudiantslyonnais` WHERE `Nom`='Dupond'
#1451 - Cannot delete or update a parent row: a foreign key
constraint fails (`projet_bdd`.`notation`, CONSTRAINT `FkNotation1`
FOREIGN KEY (`NumEtu`) REFERENCES `etudiant` (`NumEtu`))
```

On a donc une erreur liée à une contrainte : on ne peut pas supprimer un étudiant qui a encore une note dans la table notation.

4. L'étudiant Durand déménage lors de son stage à 1, Rue de Lyon Paris 75002. Est-il possible de mettre à jour ses informations depuis la vue *EtudiantsLyonnais* ? Vérifier ?

```
UPDATE `etudiantslyonnais` SET `Rue`='Rue de Lyon', `Ville`='Paris',
`CP`=75002 WHERE `etudiantslyonnais`.`Nom` = 'Durand'
```

Pas de problème, il sort bien de la vue *EtudiantsLyonnais*.

5. Est-il possible d'ajouter l'étudiant (700, Bill, Gates, 01-09-1980, Rue de Paris, 69005, Lyon) ?  
Vérifier ?

```
INSERT INTO Etudiant VALUES (700, 'Bill', 'Gates', 1980-09-01, 'Rue de Paris', 69005, 'Lyon')
```

L'étudiant apparaît dans EtudiantsLyonnais

```
INSERT INTO EtudiantsLyonnais VALUES (700, 'Bill', 'Gates', '1980-09-01', 'Rue de Paris', 69005, 'Lyon')
```

L'étudiant apparaît bien dans Etudiant et EtudiantsLyonnais

6. Donner la vue MoyenneLyonnais contenant la moyenne par matière des étudiants lyonnais.

```
CREATE VIEW `MoyenneLyonnais` AS
SELECT AVG(`note`) AS `moyenne`, `CodeMat`
FROM `notation`, `EtudiantsLyonnais`, `epreuve`
WHERE `notation`.`numetu` = `EtudiantsLyonnais`.`numetu` AND
`notation`.`numepreuve` = `epreuve`.`numepreuve` GROUP BY `CodeMat`
```

moyenne	CodeMat
11.5000	ECO
7.5000	INF
13.5000	STA

7. Est-il possible de mettre à jour une note de l'étudiant Dupont ?

On peut mettre à jour une note de cet étudiant, comme suit :

```
UPDATE `projet_bdd`.`notation` SET `Note` = '10' WHERE
`notation`.`NumEtu` = (SELECT `NumEtu` FROM `EtudiantsLyonnais`
WHERE `Nom`='Dupont') AND `notation`.`NumEpreuve` = '21031';
```

Mais on ne peut pas directement depuis la vue EtudiantsLyonnais

8. Donnez un droit d'accès en consultation à la vue EtudiantsLyonnais à votre binôme. Vérifier le contenu de la vue depuis les 2 comptes.

Pour simuler deux accès, on ouvre 2 fois phpmyadmin sur le même poste, dans deux onglets différents

Il n'y a pas de problème à déplorer.

9. Mettez à jour le nom de rue de Dupond qui passe à Rue de la Meuse, 69008, Lyon. Vérifier sur les 2 comptes si la mise à jour est faite. Le second compte essaye de remettre Dupond à son ancienne adresse. Que se passe-t-il ? Quoi faire ?

Sur la session 1, on lance

```
UPDATE `etudiantslyonnais` SET `Rue`='Rue de la Meuse',
`Ville`='Lyon', `CP`=69008 WHERE `etudiantslyonnais`.`Nom` =
'Dupond'
```

Apparaît sur la session 2 après rechargement de la page ou mise à jour des données.

Sur la deuxième session

```
UPDATE `etudiantslyonnais` SET `Rue`='RuedeDôle', `Ville`='Lyon',
`CP`=69007 WHERE `etudiantslyonnais`.`Nom` = 'Dupond'
```

Apparaît sur la session 1 après rechargement de la page

Idem sans recharger la page ou mise à jour après la première session : pas de problème à déplorer

10. Votre binôme insère un nouvel étudiant lyonnais. Que doit-il faire pour que vous puissiez consulter ce nouvel étudiant dans la base de données ?

Je dois recharger la page ou mettre à jour les données pour voir ce nouvel étudiant.

11. Une erreur de saisie entre 2 notes de 2 étudiants sur une même épreuve doit être corrigée. Comment procéder ?

Pour échanger 2 notes :

```
UPDATE
notation AS notation1
JOIN notation AS notation2 ON
(notation1.NumEtu = 110 AND notation1.NumEpreuve = 11031 AND
notation2.NumEtu = 222 AND notation2.NumEpreuve = 11031)
SET
notation1.Note = notation2.Note,
notation2.Note = notation1.Note
;
```

Ici, on échange les notes des étudiants 110 (Dupont) et 222 (West) pour l'épreuve 11031.

Bien sûr, on pourrait utiliser un nom au lieu d'un ID en utilisant une sous requête du type

```
SELECT `NumEtu` FROM `Etudiant` WHERE `Nom`='Dupont'
```

Source :

<http://www.microshell.com/database/sql/swap-values-in-2-rows-sql>

12. Que doit-on faire pour éviter des mises à jour erronées des notes des étudiants par vous ou votre binôme (conflit de mise à jour) ?

Il faut utiliser les transactions (COMMIT et ROLLBACK), ne plus être en AUTOCOMMIT.

### III – Gestion de la concurrence d'accès, des transactions et reprise sur panne

#### 3.2 – Analyse de programme

On s'intéresse ici à prévoir le contenu de la base à chaque instant au cours du déroulement d'une séquence d'ordres SQL.

Ordres	T ds trans $p_1$	T vue par $p_2$
	(1,2)	(1,2)
INSERT (3,4)	(1,2) (3,4)	(1,2)
INSERT (1,2)	(1,2) (3,4) (1,2)	(1,2)
ROLLBACK	(1,2)	(1,2)
COMMIT	(1,2)	(1,2)
INSERT (1,2)	(1,2) (1,2)	(1,2)
EXIT	(1,2)	(1,2)

Ordres	T ds trans $p_1$	T vue par $p_2$
	(1,2)	(1,2)
ROLLBACK	(1,2)	(1,2)
INSERT (1,2)	(1,2) (1,2)	(1,2)
ALTER TABLE T ADD (C INTEGER)	(1,2,0) (1,2,0)	(1,2)
UPDATE T SET B = 3 WHERE A = 1	(1,3,0) (1,3,0)	(1,2)
panne : tapez ctrl-\	(1,2)	(1,2)

ordres	T ds trans $p_1$	T vue par $p_2$
	(1,2)	(1,2)
INSERT (3,4)	(1,2) (3,4)	(1,2)
INSERT (5,6)	(1,2) (3,4) (5,6)	(1,2)
SET AUTOCOMMIT ON	(1,2) (3,4) (5,6)	(1,2)
INSERT (7,8)	(1,2) (3,4) (5,6) (7,8)	(1,2) (3,4) (5,6) (7,8)
ROLLBACK	(1,2) (3,4) (5,6) (7,8)	(1,2) (3,4) (5,6) (7,8)
INSERT (1,2,3,4) <i>Ici, on part du principe qu'on ne peut pas insérer de quadruplet, et que l'insertion est ignorée</i>	(1,2) (3,4) (5,6) (7,8)	(1,2) (3,4) (5,6) (7,8)
DELETE FROM T WHERE A = 1	(3,4) (5,6) (7,8)	(3,4) (5,6) (7,8)
<i>panne : tapez ctrl-\</i>	(3,4) (5,6) (7,8)	(3,4) (5,6) (7,8)

### 3.3 – Construction de programme

1. Concrétisez l'exemple en considérant un ou deux n-uplets dans chaque table.

```
INSERT INTO BANQUE VALUES ("Joe", 3000);
INSERT INTO VOYAGE VALUES ("Joe", "Dallas", 1000);
UPDATE BANQUE SET MONTANT = MONTANT-145 WHERE CLIENT ="Joe";

INSERT INTO BANQUE VALUES ("Joe", 2556);
INSERT INTO VOYAGE VALUES ("Joe", "Dallas", 45);
UPDATE BANQUE SET MONTANT = MONTANT-567 WHERE CLIENT ="Joe";

INSERT INTO BANQUE VALUES ("Jack", 254);
INSERT INTO VOYAGE VALUES ("Jack", "New York", 26);
UPDATE BANQUE SET MONTANT = MONTANT-567 WHERE CLIENT ="Jack";
```

2. Énoncez la contrainte que doivent satisfaire les données de votre exemple.

Si on considère que la banque n'autorise pas de découvert, alors Montant > 0. On utilise un CHECK :

```
ALTER TABLE banque
ADD CONSTRAINT montant_positif CHECK (montant>0);
```

On utilise également une clé étrangère dans voyage, référençant le client dans la table banque, si on part de l'hypothèse que le client dans la banque est une clé primaire.

```
ALTER TABLE voyage
ADD FOREIGN KEY (client)
REFERENCES banque (client);
```

3. Ce programme est-il correct ? Sinon, donnez tous les contre-exemples (pannes), en donnant à chaque fois le contenu de la base, et le déroulement du programme. Expliquez. Testez sur machine en simulant les pannes (lesquelles ? comment ?).

Le programme est incorrect : on doit faire des transactions en utilisant COMMIT/ROLLBACK pour tester les modifications avant de les entrer en BDD (actuellement en autocommit)

Pannes possibles :

- Si le client n'existe pas dans la banque et qu'on insère un voyage ;
- Si fait un UPDATE alors qu'on ne sait pas si l'entrée existe ;
- Concurrence d'accès.

4. Réécrivez ce programme pour qu'il soit robuste aux pannes. Donnez toutes les variantes possibles. Testez sur machine en simulant les pannes.

Pour faire une insertion du début

```
START TRANSACTION ;
INSERT INTO BANQUE VALUES ('Jules', 1000) ;
INSERT INTO VOYAGES VALUES ('Jules', 'Saint Lucien', 100) ;
UPDATE BANQUE SET MONTANT -= 100 WHERE CLIENT = 'Jules' ;
COMMIT;
```

Pour faire une insertion si le client existe

```
START TRANSACTION ;
INSERT INTO VOYAGES VALUES ('Jules', 'Saint Lucien', 100) ;
UPDATE BANQUE SET MONTANT -= 100 WHERE CLIENT = 'Jules' ;
COMMIT;
```

Grâce aux contraintes primary key de client dans la banque et foreign key dans voyages, on ne peut pas créer un voyage pour lequel le client n'existerait pas dans la banque. De plus, le montant du compte en banque reste positif grâce au CHECK plus haut.

5. Considérons mieux votre programme. Est-il possible qu'un des ordres échoue sans faire échouer la transaction ? Votre programme est-il donc fiable ? Indiquez intuitivement ce qu'il faudrait faire pour qu'il le devienne. Est-ce le même problème pour toutes les variantes ?

Il faut unifier les deux solutions présentées à la solution précédente :

```
START TRANSACTION ;
INSERT INTO BANQUE VALUES (client, montant)
SELECT * FROM (SELECT 'Jules') AS tmp
WHERE NOT EXISTS (
    SELECT client FROM BANQUE WHERE client = 'Jules'
) LIMIT 1;
INSERT INTO VOYAGES VALUES ('Jules', 'Saint Lucien', 100) ;
UPDATE BANQUE SET MONTANT -= 100 WHERE CLIENT = 'Jules' ;
COMMIT;
```

Source:

<http://stackoverflow.com/questions/3164505/mysql-insert-record-if-not-exists-in-table>

### 3.4 – Introduction au contrôle de concurrence

#### 3.4.1 – Analyse de programme

Remplir les tableaux dans l'ordre dans lequel les ordres sont effectivement exécutés. Puis vérifiez sur machine.

1 (p<sub>1</sub>) : DELETE T WHERE B=1;

2 (p<sub>1</sub>) : COMMIT;

Client	ordre	T ds trans p <sub>1</sub>	T ds trans p <sub>2</sub>	T vue par p <sub>3</sub>
		11	11	11
		22	22	22
P1	DELETE T WHERE B=1	22	11	11
			22	22
P1	COMMIT	22	22	22

1 (p<sub>1</sub>) : DELETE T WHERE B=1;

2 (p<sub>2</sub>) : UPDATE T SET B=B-1;

3 (p<sub>1</sub>) : COMMIT;

4 (p<sub>2</sub>) : COMMIT;

Client	ordre	T ds trans p <sub>1</sub>	T ds trans p <sub>2</sub>	T vue par p <sub>3</sub>
		11	11	11
		22	22	22
P1	DELETE T WHERE B=1	22	11	11
			22	22
P2	UPDATE T SET B=B-1;	11	10	11
		22	21	22
P1	COMMIT	22	21	22
P2	COMMIT	21	21	21

1 (p<sub>2</sub>) : UPDATE T SET B=B-1;

2 (p<sub>1</sub>) : DELETE T WHERE B=1;

3 (p<sub>2</sub>) : COMMIT;

4 (p<sub>1</sub>) : COMMIT;

Client	ordre	T ds trans p <sub>1</sub>	T ds trans p <sub>2</sub>	T vue par p <sub>3</sub>
		11	11	11
		22	22	22
P2	UPDATE T SET B=B-1	11	10	11
		22	21	22
P1	DELETE T WHERE B=1	22	11	11
			22	22
P2	COMMIT	10	10	10
		21	21	21
P1	COMMIT	10	10	10

- 1 (p<sub>1</sub>) : UPDATE T SET A=1 WHERE B=2;  
 2 (p<sub>2</sub>) : DELETE T WHERE A=1;  
 3 (p<sub>1</sub>) : COMMIT;  
 4 (p<sub>2</sub>) : COMMIT;

Client	ordre	T ds trans p <sub>1</sub>	T ds trans p <sub>2</sub>	T vue par p <sub>3</sub>
		11 22	11 22	11 22
P1	UPDATE T SET A=1 WHERE B=2	11 12	11 22	11 22
P2	DELETE T WHERE A=1	11 11		11 12
P1	COMMIT	11 12	11 12	11 12
P2	COMMIT			

*Note : une case vide signifie que la table est vide.*

### 3.4.2 – Demandes de verrous mutuellement bloquantes

*Détailler les problèmes liés à l'exécution concurrente des séquences s1 et s2 par des utilisateurs autorisés différents. Quelles sont les solutions que propose Oracle pour gérer ce type de problème.*

Tant que la séquence 1 est en exécution, la séquence 2 ne peut pas s'exécuter. En cas de blocage, Oracle fait un rollback automatique pour prévenir toute corruption de données.