Bases de données
Suite des commandes SQL
SELECT : jointures et requêtes imbriquées
DDL, Contraintes d'intégrité, Vue, Index
DCL, commit, rollback, autocommit

SOMMAIRE

Dernière édition : septembre 2019

SUITES DES COMMANDES SQL

1 : SELECT multi-tables - Jointures et requêtes imbriquées

Présentation du DDL multi-tables

- On suit la même logique que pour le TP2.
- On y ajoute la table des départements comme dans l'exemple déjà abordé en introduction : les employés.

Table des employés

NE	NOM	FONCTION	DATEMB	SAL	COMM	ND	NEchef
7369	SMITH	CLERK	1980-12-17	800.00	NULL	20	7902
7499	ALLEN	SALESMAN	1981-02-20	1600.00	300.00	30	7698
7521	WARD	SALESMAN	1981-02-22	1250.00	500.00	30	7698
7566	JONES	MANAGER	1981-04-02	2975.00	NULL	20	7839
7654	MARTIN	SALESMAN	1981-09-28	1250.00	1400.00	30	7698
7698	BLAKE	MANAGER	1981-05-01	2850.00	NULL	30	7839
7782	CLARK	MANAGER	1981-06-09	2450.00	NULL	10	7839
7788	SCOTT	ANALYST	1982-12-09	3000.00	NULL	20	7566
7839	KING	PRESIDENT	1981-11-17	5000.00	NULL	10	NULL
7844	TURNER	SALESMAN	1981-09-08	1500.00	0.00	30	7698
7900	JAMES	CLERK	1981-12-03	950.00	NULL	30	7698
7902	FORD	ANALYST	1981-12-03	3000.00	NULL	20	7566
7903	ADAMS	CLERK	1983-01-12	1100.00	NULL	20	7788
7904	MILLER	CLERK	1982-01-23	1300.00	NULL	10	7782

Table des départements

ND	NOM	VILLE
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Création de la BD et de la table des employés : CREATE DATABASE, CREATE TABLE

- La commande **CREATE DATABASE** permet de créer une BD.
- La commande **CREATE TABLE** permet de créer une table.
- On utilise aussi la commande **DROP DATABASE** qui permet de supprimer une BD.

- Le code DDL va permettre de créer la BD et les tables dans la BD (une seule dans notre exemple).
- Sa syntaxe est simple et proche du langage parlé, avec quelques subtilités informatiques !
- Il suffit de lire ce qui est écrit pour quasiment tout comprendre.

```
-- Création de la BD  :
-- on supprime la database, on la recrée, on l'utilise

drop database if exists empdeptTP03sansFK;
create database empdeptTP03sansFK;
use empdeptTP03sansFK;

-- Création de la table

CREATE TABLE emp (
           NE             integer primary key auto_increment,
           NOM            varchar(10) not NULL,
           FONCTION       varchar(9) not NULL,
           DATEMB         date not NULL,
           SAL            float(7,2) not NULL,
           COMM           float(7,2),
           ND             integer not null,
           NEchef         integer
);

CREATE TABLE dept (
    ND        integer primary key auto_increment,
    NOM       varchar(14) not NULL,
    VILLE     varchar(13) not NULL
);
```

Insertion de tuples dans la table « emp » : INSERT

- La commande INSERT permet d'ajouter des tuples.
- Sa syntaxe est simple et proche du langage parlé, avec quelques subtilités informatiques !
- Il suffit de lire ce qui est écrit pour quasiment tout comprendre.

```
-- création des tuples

INSERT INTO emp VALUES
    (7839,'KING','PRESIDENT','1981-11-17',5000,NULL,10,NULL),
    (7698,'BLAKE','MANAGER','1981-05-1',2850,NULL,30,7839),
    (7782,'CLARK','MANAGER','1981-06-9',2450,NULL,10,7839),
    (7566,'JONES','MANAGER','1981-04-2',2975,NULL,20,7839),
    (7654,'MARTIN','SALESMAN','1981-09-28',1250,1400,30,7698),
    (7499,'ALLEN','SALESMAN','1981-02-20',1600,300,30,7698),
    (7844,'TURNER','SALESMAN','1981-09-8',1500,0,30,7698),
    (7900,'JAMES','CLERK','1981-12-3',950,NULL,30,7698),
    (7521,'WARD','SALESMAN','1981-02-22',1250,500,30,7698),
    (7902,'FORD','ANALYST','1981-12-3',3000,NULL,20,7566),
    (7369,'SMITH','CLERK','1980-12-17',800,NULL,20,7902),
    (7788,'SCOTT','ANALYST','1982-12-09',3000,NULL,20,7566),
    (NULL,'ADAMS','CLERK','1983-01-12',1100,NULL,20,7788),
    (NULL,'MILLER','CLERK','1982-01-23',1300,NULL,10,7782);

INSERT INTO dept VALUES
  (10,'ACCOUNTING','NEW YORK'),
  (20,'RESEARCH','DALLAS'),
  (30,'SALES','CHICAGO'),
  (40,'OPERATIONS','BOSTON')
;
```

Affichage mono-table des données : select simple

On peut lister le contenu des tables :

```
SELECT * FROM emp ;
```

Et

```
SELECT * FROM dept ;
```

Affichage multi-table des données : jointure

- On peut aussi afficher tous les employés avec les informations de leurs départements. C'est une jointure.
- Le résultat sera :

NE	NOM	FONCTION	DATEMB	SAL	COMM	ND	NEchef	ND	NOM	VILLE
7369	SMITH	CLERK	1980-12-17	800.00	NULL	20	7902	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	1981-02-20	1600.00	300.00	30	7698	30	SALES	CHICAGO
7521	WARD	SALESMAN	1981-02-22	1250.00	500.00	30	7698	30	SALES	CHICAGO
7566	JONES	MANAGER	1981-04-02	2975.00	NULL	20	7839	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	1981-09-28	1250.00	1400.00	30	7698	30	SALES	CHICAGO
7698	BLAKE	MANAGER	1981-05-01	2850.00	NULL	30	7839	30	SALES	CHICAGO
7782	CLARK	MANAGER	1981-06-09	2450.00	NULL	10	7839	10	ACCOUNTING	NEW YORK
7788	SCOTT	ANALYST	1982-12-09	3000.00	NULL	20	7566	20	RESEARCH	DALLAS
7839	KING	PRESIDENT	1981-11-17	5000.00	NULL	10	NULL	10	ACCOUNTING	NEW YORK
7844	TURNER	SALESMAN	1981-09-08	1500.00	0.00	30	7698	30	SALES	CHICAGO
7900	JAMES	CLERK	1981-12-03	950.00	NULL	30	7698	30	SALES	CHICAGO
7902	FORD	ANALYST	1981-12-03	3000.00	NULL	20	7566	20	RESEARCH	DALLAS
7903	ADAMS	CLERK	1983-01-12	1100.00	NULL	20	7788	20	RESEARCH	DALLAS
7904	MILLER	CLERK	1982-01-23	1300.00	NULL	10	7782	10	ACCOUNTING	NEW YORK

Pour obtenir ce résultat, la requête est :

```
SELECT *
FROM emp e, dept d
WHERE e.nd = d.nd;
```

Cette requête est une jointure :

- Il y a 2 tables derrière le FROM, séparées par une virgule, et renommées en « e » et « d »
- Le WHERE fait le lien entre le « nd » des employés et le « nd » des département.
- Le « nd » des départements, c'est la clé primaire de la table « dept ».
- Le « nd » des employés, c'est une clé étrangère : elle fait référence à la clé primaire de « dept ».

Affichage multi-table des données : jointure et requête imbriquée

- On veut maintenant lister uniquement les employés d'un département de NEW YORK, sans afficher les informations du département.
- Le résultat sera :

```
+------+--------+-----------+------------+---------+------+----+---------+
| NE   | NOM    | FONCTION  | DATEMB     | SAL     | COMM | ND | NEchef |
+------+--------+-----------+------------+---------+------+----+---------+
| 7782 | CLARK  | MANAGER   | 1981-06-09 | 2450.00 | NULL | 10 |   7839 |
| 7839 | KING   | PRESIDENT | 1981-11-17 | 5000.00 | NULL | 10 |   NULL |
| 7904 | MILLER | CLERK     | 1982-01-23 | 1300.00 | NULL | 10 |   7782 |
+------+--------+-----------+------------+---------+------+----+---------+
```

Avec une jointure, on peut écrire :

```
SELECT e.*
FROM emp e, dept d
WHERE e.nd = d.nd
AND d.ville = 'NEW YORK';
```

Avec une requête imbriquée, on peut écrire :

```
SELECT *
FROM emp
WHERE nd in (
   SELECT nd FROM dept
   WHERE ville = 'NEW YORK'
);
```

- Ca donne le même résultat que la jointure.
- Le principe est plus lisible que pour la jointure : on veut tous les employés dont le « nd » appartient (« in ») à la liste des « nd » des département dont la ville est NEW YORK (en l'occurrence, il n'y en a qu'un, mais il pourrrait y en avoir plusieurs).
- Bien que ça puisse paraître plus simple qu'une jointure, il faut éviter cette écriture car elle est potentiellement moins performante.

2 : DDL - notion de vue

Présentation

- Une vue est un nouvel objet enregistré dans la BD, au même titre qu'une table ou qu'un utilisateur.
- Une vue est définie par un SELECT : un SELECT renvoie une table. Une vue correpond à la table renvoyée par son SELECT de définition.
- On peut faire des SELECT sur une vue comme sur une table.

Création d'une vue : CREATE VIEW

- On veut créer la vue des employés de NEW YORK, sans les informations du département.
- Le select pour les employés de NEW YORK est :

```
SELECT e.*
FROM emp e, dept d
WHERE e.nd = d.nd
AND d.ville = 'NEW YORK';
```

Pour créer la vue on écrit :

```
CREATE VIEW empNewYork AS
SELECT e.*
FROM emp e, dept d
WHERE e.nd = d.nd
AND d.ville = 'NEW YORK';
```

Lister les vues crées : SHOW TABLES

- Les vues sont des tables.
- Un « show tables » liste les tables et les vues.

Utiliser une vue

Tous les tuples de la vue

- Si on veut tous les employés de la vue « empNewYork » on écrit :

```
SELECT * FROM empNewYork;
```

Certains tuples de la vue

- Si on veut les employés de la vue « empNewYork » dont le salaire est supérieur à 2000 on écrit :

```
SELECT * FROM empNewYork
WHERE sal > 2000 ;
```

Suppression d'une vue : DROP VIEW

```
DROP VIEW empNewYork ;
```

Modification d'une vue

- Il n'y a pas de modification d'une vue. Pour modifier une vue, on la supprime et on la recrée.

Afficher le code d'une vue : SHOW CREATE VIEW

```
SHOW CREATE VIEW empNewYork ;
```

- Le résultat demande un effort de lecture, mais on retrouve bien le code de création de la vue.

```
| empnewyork | CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL
SECURITY DEFINER VIEW `empnewyork` AS select `e`.`NE` AS `NE`,`e`.`NOM` AS
`NOM`,`e`.`FONCTION` AS `FONCTION`,`e`.`DATEMB` AS `DATEMB`,`e`.`SAL` AS
`SAL`,`e`.`COMM` AS `COMM`,`e`.`ND` AS `ND`,`e`.`NEchef` AS `NEchef` from
(`emp` `e` join `dept` `d`) where ((`e`.`ND` = `d`.`ND`) and (`d`.`VILLE`
= 'NEW YORK')) | cp850                | cp850_general_ci     |
```

- Mis en forme on obtient :

```
select
   `e`.`NE` AS `NE`,
   `e`.`NOM` AS `NOM`,
   `e`.`FONCTION` AS `FONCTION`,
   `e`.`DATEMB` AS `DATEMB`,
   `e`.`SAL` AS `SAL`,
   `e`.`COMM` AS `COMM`,
   `e`.`ND` AS `ND`,
   `e`.`NEchef` AS `NEchef`
from (`emp` `e` join `dept` `d`)
where (
    (`e`.`ND` = `d`.`ND`) and
    (`d`.`VILLE` = 'NEW YORK')
)
```

- Le mot clé « join » à la même signification que la virgule dans la syntaxe qu'on avait utilisée.

3 : DCL - commit et rollback – autocommit – notion de transaction

Première approche de la notion de transaction

- Pour gérer **plusieurs utilisateurs autour du serveur**, il faut gérer ce qu'on appelle des « **transactions** ».
- Une **transaction** c'est **plusieurs modifications de la BD** (plusieurs insert par exemple) qui seront **validées toutes ensemble** ou alors toutes annulées.
- Cela veut dont dire que **le client qui commence une transaction**, donc qui fait une première modification (un insert par exemple), **voit la modification, tandis que les autres clients ne la voit pas.** Ils la verront quand le premier client aura validé sa transaction en entier.

Validation d'une transaction : COMMIT

- Pour valider une transaction, il faut faire un « COMMIT ».

Autocommit

- **Dans MySQL**, toutes les modifications apportées à la BD (les insert, par exemple) sont **commités automatiquement**.
- En effet, une variable du serveur nommée « **autocommit** » qui permet cela. Elle vaut 1 par défaut.
- On peut la **passer à 0 pour arrêter l'autocommit** : dans ce cas, chaque insert, par exemple, sera considéré comme un début de transaction qu'il faudra commiter pour le valider et le rendre visible aux autres utilisateurs.

Affichage de la variable autocommit :

```
select @@autocommit ;
```

Modification de la variable autocommit :

```
set @@autocommit = 0 ;
```

Conséquence d'une transaction en cours

- Quand une **transaction est en cours** (par exemple un insert non commité), le résultat (l'insert) est **visible par le client qui a fait l'insert** mais **non visible par les autres client**.
- De plus, **les contraintes d'intégrité associées à la modifications s'appliqueront quand même pour les autres clients**, même s'ils ne voient pas la modification : c'est comme si ils la voyaient !
- Par exemple si une transaction crée l'employé numéro 1000, un autre client ne pourra pas créer d'employé numéro 1000. Par contre, si la transaction est annulée, la création par l'autre client deviendra possible.

Rollback

- Le rollback annule une transaction en cours : toutes les modifications réalisées (le ou les insert par exemple) et qui n'ont pas été commitées, seront annulées.

Bien comprendre ces notions

- Pour bien comprendre ces notions, il faut les tester à travers deux clients. C'est ce que propose le TP.

4 : DDL - Contraintes d'intégrité et Contraintes d'intégrité référentielles

Les principales contraintes d'intégrité non référentielles (CI)

CI fortes

- **Type** : il fixe le type de valeur attendue : entier, float, varchar, date, etc.
- **Clé primaire** : obligatoire et unique. Sert de référence pour les clés étrangères.
- **Obligatoire** : ne peut pas valoir NULL. Doit dont être renseigné.
- **Unique** : ne peut pas avoir la même valeur pour un autre tuple.

CI faibles

- **Valeur par défaut** : valeur que prendra l'attribut si on ne lui donne pas de valeur.
- **L'auto-incrément** : l'attribut prendra la valeur du précédent +1 si on ne lui donne pas de valeur.

Conséquences des CI fortes

- Si on donne une valeurs qui ne vérifie pas la contrainte, ça crée une erreur et ça ne marche pas.

Conséquences des CI faibles

- Si on ne donne pas de valeur, il y a une valeur par défaut. Si on donne une valeur, c'est la valeur donnée qui l'emporte.

Exemple :

```
CREATE TABLE emp (
           NE            integer primary key auto_increment,
           NOM           varchar(10) not NULL,
           FONCTION      varchar(9) not NULL,
           DATEMB        date not NULL,
           SAL           float(7,2) not NULL,
           COMM          float(7,2),
           ND            integer not null,
           NEchef        integer
);
```

- ***On trouve presque toutes les contraintes présentées :***
  - **type** : integer, varchar(10), date, float(7,2)
  - **clé primaire** : primary key,
  - **obligatoire** : not null,
  - **auto-incrément** : auto_increment

- ***On ne trouve pas :***
  - **unique** : c'est une contrainte assez rare. En général, seul l'attribut clé primaire est unique.
  - **valeur par défaut** : selon les cas, on peut avoir des valeurs par défaut ou pas. Ici, il n'y en n'a pas.

Contrainte d'intégrité référentielle (CIR) - les clés étrangères

Définition

- Une clé étrangère est une contrainte d'intégrité qui s'applique à un attribut qui fait référence à un autre attribut qui est clé primaire de sa table.
- On parle de contrainte d'intégrité référentielle : CIR.
- C'est une contrainte d'intégrité forte.

Exemples

- Dans la table des employés, il y a 2 clés étrangères :
- « nd » qui fait référence à la clé primaire « nd » de la table des département.
- « nechef » qui fait référence à la clé primaire « ne » de la table des employés.

Conséquences des CIR

- Les conséquences des CIR sont nombreuses.
- On en trouve **au niveau du DDL** et **au niveau du DML**.
- A ces niveaux les conséquences sont variables selon les commandes : CREATE, ALTER, DROP, INSERT, UPDATE, DELETE.
- On va les regarder une par une.

Exemple de code :

Création de la BD et des tables :

```
-- Création de la BD  :
-- on supprime la database, on la recrée, on l'utilise

drop database if exists empdeptTP03avecFK;
create database empdeptTP03avecFK;
use empdeptTP03avecFK;

-- Création de la table

CREATE TABLE dept (
    ND          integer primary key auto_increment,
    NOM         varchar(14) not NULL,
    VILLE       varchar(13) not NULL
)engine InnoDB;

CREATE TABLE emp (
    NE          integer primary key auto_increment,
    NOM         varchar(10) not NULL,
    FONCTION    varchar(9) not NULL,
    DATEMB      date not NULL,
    SAL         float(7,2) not NULL,
    COMM        float(7,2),
    ND          integer not null,
    NEchef      integer,
    foreign key (ND)     references dept(ND),
    foreign key (NEchef) references emp(NE)
)engine InnoDB;
```

A noter dans le code :

- On crée « dept » avant « emp »
- foreign key (ND) references dept(ND),
- foreign key (NEchef) references emp(NE)
- engine InnoDB;

Création des tuples :

```
-- création des tuples

INSERT INTO dept VALUES
  (10,'ACCOUNTING','NEW YORK'),
  (20,'RESEARCH','DALLAS'),
  (30,'SALES','CHICAGO'),
  (40,'OPERATIONS','BOSTON')
;

INSERT INTO emp VALUES
    (7839,'KING','PRESIDENT','1981-11-17',5000,NULL,10,NULL),
    (7698,'BLAKE','MANAGER','1981-05-1',2850,NULL,30,7839),
    (7782,'CLARK','MANAGER','1981-06-9',2450,NULL,10,7839),
    (7566,'JONES','MANAGER','1981-04-2',2975,NULL,20,7839),
    (7654,'MARTIN','SALESMAN','1981-09-28',1250,1400,30,7698),
    (7499,'ALLEN','SALESMAN','1981-02-20',1600,300,30,7698),
    (7844,'TURNER','SALESMAN','1981-09-8',1500,0,30,7698),
    (7900,'JAMES','CLERK','1981-12-3',950,NULL,30,7698),
    (7521,'WARD','SALESMAN','1981-02-22',1250,500,30,7698),
    (7902,'FORD','ANALYST','1981-12-3',3000,NULL,20,7566),
    (7369,'SMITH','CLERK','1980-12-17',800,NULL,20,7902),
    (7788,'SCOTT','ANALYST','1982-12-09',3000,NULL,20,7566),
    (NULL,'ADAMS','CLERK','1983-01-12',1100,NULL,20,7788),
    (NULL,'MILLER','CLERK','1982-01-23',1300,NULL,10,7782);
```

A noter dans le code :

- On crée les départements avant les employés.

Engine : innodb et myisam

- Le SGBD MySQL fonctionne avec différents « moteur » = « engine ».
- Les moteurs gèrent plus ou moins certaines choses.

myisam : le moteur historique

- « **myisam** » est le moteur historique. Il ne gère ni les transactions ni l'intégrité référentielle.
- Aujourd'hui, mieux vaut éviter de l'utiliser sauf dans certaines problématiques d'optimisation.

innodb

- « **innodb** » est un moteur qui gère les transactions et l'intégrité référentielle. C'est celui qu'il vaut mieux utiliser aujourd'hui.
- Il existe d'autres moteurs avec les même caractéristiques comme « **berkleydb** ».

Connaître le moteur d'une table : show create table

- Quand on fait un « show create table », le moteur de la table d'affiche à la fin de la commande de création de la table, comme dans le code présenté ici.

Conséquences sur le DDL

Sur les CREATE TABLE

- Une CIR sur une table fait référence à une autre table. Il faut donc que cette autre table ait déjà été créée pour que ce soit possible de déclarer la contrainte.
- En cas de problème, on peut faire un ALTER TABLE pour ajouter les contraintes après avoir créées les tables sans contraintes.

```
ALTER TABLE emp ADD CONSTRAINT
    foreign key (ND)     references dept(ND);
```

Sur les DROP TABLE

- Pour supprimer une table, il faut qu'elle ne soit pas référencée par une autre table.
- On ne peut pas supprimer la table « dept » si on n'a pas déjà supprimée la table « emp ».
- A noter qu'en supprimant toute la database, on n'a pas de problème !
- En cas de problème, on peut faire un ALTER TABLE pour supprimer les contraintes qui empêche la suppression d'une table.
- Pour ça, il faut trouver le nom de la contrainte : on le trouve avec un show create table.

```
ALTER TABLE emp DROP FOREIGN KEY emp_ibfk_2;
```

Sur les ALTER TABLE

- Quand on ajoute une contrainte, il faut qu'elle soit cohérente avec l'état de la base, c'est-à-dire avec les valeurs des tuples présents dans la base.
- Par exemple, si on ajoute une clé étrangère, il faut que les liens entre les tuples soient « propres ».

Exemples de code avec des ALTER TABLE

- Le ALTER TABLE permet d'ajouter et de supprimer des contraintes d'intégrité.
- Voici quelques exemples de code :

➢ ***Ajout de contraintes***

- Ajout d'une clé étrangère : FOREIGN KEY

```
ALTER TABLE emp ADD CONSTRAINT
    foreign key (ND)     references dept(ND);
```

- Ajout d'une clé étrangère : FOREIGN KEY

```
ALTER TABLE emp ADD CONSTRAINT
    primary key (NE);
```

➢ ***Suppression de contraintes***

- Suppression d'une FOREIGN KEY par son nom :

```
ALTER TABLE emp DROP FOREIGN KEY emp_ibfk_2;
```

- Suppression d'une PRIMARY KEY

```
ALTER TABLE emp DROP PRIMARY KEY;
```

- Possible uniquement s'il n'y a pas d'auto_increment.
- Sinon il faut redéfinir l'attribut

➢ ***Redéfinition complète d'un attribut de contraintes***

```
ALTER TABLE emp MODIFY
    NE         integer;
```

- En faisant ça, NE est toujours clé primaire. Mais il n'a plus d'auto-incrément.
- Ensuite on peut supprimer la primary key.

Conséquences sur le DML

Sur les INSERT

- Une CIR sur une table s'applique à tous les tuples de la table.
- Quand on inserte un tuple dont un attribut est clé étrangère, il faut que le tuple correspondant existe.
- Par exemple, si je veux ajouter un employé dont le numéro de département est 50, il faut que le département 50 existe, sinon l'insertion n'aura pas lieu.

Sur les UPDATE

- Si j'update un attribut qui est clé étrangère, il faut que la nouvelle valeur que je donne fasse référence à un tuple qui existe.
- Par exemple, si je veux modifier un employé dont le numéro de département est 10 et lui donner 50 comme numéro de département, il faut que le département 50 existe, sinon la modification n'aura pas lieu.

Sur les DELETE

- Si je delete un tuple, il faut qu'il ne soit pas référencé par une clé étrangère d'un autre tuple.
- Par exemple, si je veux supprimer le département n°10, il faut qu'il n'existe pas d'employés dans le département 10.

5 : DDL - notion d'index

Présentation

- Un index est une **table créée à partir d'un attribut trié d'une autre table et de sa clé primaire**.
- Cette table est triée dans l'ordre de l'attribut indexé : elle permet de faire des recherches plus rapides (**recherche dichotomique**)
- Dans les BD relationnelles, l'indexation est **LA technique qui va permettre d'accélérer l'accès aux données** en permettant de faire des recherches dichotomiques.
- L'autre technique d'optimisation est la mise en mémoire cache pour limiter les accès disques.

Exemple

Soit le table des employés, non indexée :

Adresse	NE	Nom	Fonction	DateEmb	Salaire	Comm	#ND	*NEchef
Ad1	7839	KING	PRESIDENT	17/11/81	5000	NULL	10	NULL
Ad2	7698	BLAKE	MANAGER	01/05/81	2850	NULL	30	7839
Ad3	7782	CLARK	MANAGER	09/06/81	2450	NULL	10	7839
Ad4	7566	JONES	MANAGER	02/04/81	2975	NULL	20	7839
Ad5	7654	MARTIN	SALESMAN	28/09/81	1250	1400	30	7698
Ad6	7499	ALLEN	SALESMAN	20/02/81	1600	300	30	7698
Ad7	7844	TURNER	SALESMAN	08/09/81	1500	0	30	7698
Ad8	7900	JAMES	CLERK	03/12/81	950	NULL	30	7698
Ad9	7521	WARD	SALESMAN	22/02/81	1250	500	30	7698
Ad10	7902	FORD	ANALYST	03/12/81	3000	NULL	20	7566
Ad11	7369	SMITH	CLERK	17/12/80	800	NULL	20	7902
Ad12	7788	SCOTT	ANALYST	09/12/82	3000	NULL	20	7566
Ad13	7876	ADAMS	CLERK	12/01/83	1100	NULL	20	7788
Ad14	7934	MILLER	CLERK	23/01/82	1300	NULL	10	7782

Indexer la clé primaire et le job consiste à créer les deux tables d'index suivantes :

Table des employés

Adresse	NE
Ad11	7369
Ad6	7499
Ad9	7521
Ad4	7566
Ad5	7654
Ad2	7698
Ad3	7782
Ad12	7788
Ad1	7839
Ad7	7844
Ad13	7876
Ad8	7900
Ad10	7902
Ad14	7934

Table d'index

Adresse	Fonction
Ad10	ANALYST
Ad12	ANALYST
Ad8	CLERK
Ad11	CLERK
Ad13	CLERK
Ad14	CLERK
Ad2	MANAGER
Ad3	MANAGER
Ad4	MANAGER
Ad1	PRESIDENT
Ad5	SALESMAN
Ad6	SALESMAN
Ad7	SALESMAN
Ad9	SALESMAN

- Si on cherche des CLERK, on passe par le tableau d'index qui est trié.
- Avec une recherche dichotomique, ça va plus vite.
- Ensuite, on récupère les adresses qui nous amène aux informations du tableau de départ (qui contient toutes les colonnes présentées au début).

Les 3 principaux types d'index MySQL

Index unique

- C'est un index pour lequel il n'y a pas de doublon pour la colonne indexée.
- Il y a 2 index uniques :
  - Index de **clé primaire** : **PRIMARY KEY** dans le show create table.
  - Index de **clé secondaire** (ou candidate) : **UNIQUE** dans le show create table.

Index non unique

- C'est un index pour lequel il y a des doublons pour la colonne indexée.
- C'est le cas de l'exemple des « fonctions ».
- Il y a 2 cas d'utilisation des index non uniques :
  - les **attributs qui interviennent dans des recherches : KEY** dans le show create table.
  - les **clé étrangères : KEY** dans le show create table.

Index de texte

- Ce sont des index qui permette une recherche efficace de mots dans un texte.
- On parle d'index **FULLTEXT**.
- Avec MySQL on utilise ensuite **MATCH** et **AGAINTS** pour les recherche.

Création et suppression des index

3 façons de créer des index :

- ***Avec un CREATE TABLE :***
  - Dans un CREATE TABLE : on peut préciser pour les attributs PRIMARY KEY ou FOREIGN KEY ou KEY ou INDEX.

- ***Avec un ALTER TABLE :***
  - Dans un ALTER TABLE : on ajouter index avec un ADD INDEX :
    - **ALTER TABLE** nomTable ADD INDEX (nomAttribut)
  - On peut aussi ajouter un index ajoutant une clé primaire ou une clé étrangère (déjà vu).

- ***Avec un CREATE INDEX ou CREATE UNIQUE INDEX:***
  - On peut créer directement un index avec un CREATE INDEX pour un index non unique et CREATE UNIQUE INDEX pour un index unique :
    - **CREATE INDEX** nomIndex ON nomTable (nomAttribut)

```
CREATE INDEX idx_emp_sal ON emp (sal DESC) // le tri est décroissant

CREATE UNIQUE INDEX idx_emp_nom ON emp (nom) // on décide que deux
employés ne peuvent pas avoir le même nom !
```

Suppression des index

- **DROP INDEX** nomIndex ON nomTable;

Visualiser les index

SHOW CREATE TABLE

- Permet de lister tous les index.
- Les index sont les « KEY ».
- On a des **PRIMARY KEY**, des **UNIQUE KEY**, des **FOREIGN KEY** et des **KEY** simples.
- Les FOREIGN KEY sont toujours associées à une KEY simples.

SHOW INDEX FROM

```
mysql> show index from emp;
```

- Il y a beaucoup d'attributs.
- Pour rendre l'affichage lisible sous windows :

  bandeau de fenêtre /Bouton droit / propriété / taille mémoire tampon écran : mettre 300

```
mysql> show index from emp;
+-------+------------+----------+--------------+-------------+ ...
| Table | Non_unique | Key_name | Seq_in_index | Column_name | ...
+-------+------------+----------+--------------+-------------+ ...
| emp   |          0 | PRIMARY  |            1 | NE          | ...
| emp   |          1 | ND       |            1 | ND          | ...
| emp   |          1 | NEchef   |            1 | NEchef      | ...
+-------+------------+----------+--------------+-------------+
3 rows in set (0.08 sec)
```

➢ ***Autre solution : \G***

```
mysql> show index from emp\G
```

\G produit un affichage avec une ligne par attribut

```
*************************** 1. row ***************************
        Table: emp
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: NE
          ...
```

Stratégie d'indexation

La clé primaire

- Les clés primaires doivent toujours être indexées pour permettre des jointures efficaces.
- Elles le sont par défaut dans la plupart des SGBD.

Les clés étrangères

- Les clés étrangères peuvent être indexées ou pas.
- MySQL indexe automatiquement les clés étrangères quand on les déclare (rappelons toutefois que MyISAM ne gère pas l'intégrité référentielle).
- Indexer une clé étrangère est utile pour augmenter les possibilité d'optimisation sur les jointures.

Les autres attributs

- Tout attribut peut être indexé.
- Indexer un attribut est utile si une requête régulièrement utilisée fait des restrictions sur cet attribut.
- Rappelons toutefois qu'en pratique, on considère que pour une sélectivité > 20 ou 30%, l'index est inefficace (il ne faut pas que la valeur soit représentée dans plus de 20 ou 30% des cas). Donc, on n'utilise pas ce genre d'index pour un booléen. Ce sont les « index bitmap » qui permettent ça. Il n'y en a pas dans MySQL.

Étapes du TP 3

0 - Création de la BD, des tables et des tuples

0.1 : Créez la BD « empdeptTP03sansFK »

- Créez la BD empdeptTP03sansFK avec les tables « emp » et « dept » (code dans le chapitre 4 du cours).
- Vérifiez que la BD a été créée : SHOW DATABASES
- Vérifiez qu'elle contient les 2 tables : SHOW TABLES
- Vérifiez que les tables sont vides : SELECT.

0.2 : Créez, dans les table « emp » et « dept », les tuples présentés dans le cours

- Créez les tuples présentée dans le cours dans les tables « emp » et « dept »: INSERT INTO
- Affichez les tuples créés : SELECT
- Quels sont les numéros des derniers tuples créés (ADAMS et MILLER) ? Pourquoi ?

1 - Jointure et requêtes imbriquées

1.3 : Affichez les tous les tuples des deux tables.

1.4 : Affichez les employés gagnant plus que 2000.

1.5 : Affichez les employés avec les informations de leurs départements

1.6 : Affichez les employés du département SALES sans les informations de leurs départements.

- Avec une jointure
- Avec une requête imbriquée

1.7 : Affichez les employés de DALLAS qui gagnent plus que 2000 avec les informations de leurs départements.

- Peut-on faire une requête imbriquée ?

2 - Vue

2.1 : création d'une vue

- Créez la vue empSales correspondant aux employés du ou des département SALES.
- Listez les tables de la BD.
- Affichez tous les employés de la vue empSales.
- Affichez tous les employés de la vue empSales dont la commission (COM) est supérieur à 50.
- Affichez le code de la vue empSales.

2.2 : jointure avec une vue

- Créez la vue des employés du département 10 : empD10
- Listez les tables de la BD
- Affichez tous les employés de la vue empD10.
- Affichez tous les employés de la vue empD10 avec les informations de leurs départements (il faut faire une jointure).

3 - Commit, Rollback, Transaction, Autocommit

3.1 : Première gestion de transaction

- Après avoir créée la BD empdepttp03sansFK, fermez tous vos client-calculette.
- Ouvrez deux clients-calculette : CC1 et CC2.
- Dans CC1, affichez tous les départements.
- Dans CC1, ajoutez le département (50, ‘n50’, ‘v50’)
- Dans CC1 et dans CC2, affichez tous les départements. Que constatez-vous ?
- Dans CC1, affichez la variable @@autocommit (SELECT)
- Dans CC1, passez la variable @@autocommit à 0 (SET)
- Dans CC1, ajoutez le département (60, ‘n60’, ‘v60’)
- Dans CC1 et dans CC2, affichez tous les départements. Que constatez-vous ? Comment interprétez-vous la situation ?
- Dans CC2, ajoutez le département (70, ‘n70’, ‘v70’)
- Dans CC1 et dans CC2, affichez tous les départements. Que constatez-vous ? Comment interprétez-vous la situation ?
- Dans CC1, faite un « COMMIT ».
- Dans CC1 et dans CC2, affichez tous les départements. Que constatez-vous ? Comment interprétez-vous la situation ?

3.2 : Deuxième gestion de transaction

- A la suite de l’exercice précédent : dans CC1, ajoutez le département (80, ‘n80’, ‘v80’).
- Dans CC1 et dans CC2, affichez tous les départements. Que constatez-vous ?
- Dans CC2, ajoutez le département (80, ‘n80’, ‘v80’). Que constatez-vous ?
- Dans CC2A la suite de l’opération précédente, attendez sans rien faire. Que constatez-vous après environ 2 minutes ?
- Dans CC2, Après avoir eu le message « Lock wait timeout exceeded; try restarting transaction », ajoutez à nouveau le département le département (80, ‘n80’, ‘v80’). Que constatez-vous ? Rapidement, dans CC1, faites un « ROLLBACK ». Retournez dans CC2. Que constatez-vous ? Comment interprétez-vous la situation.
- Dans CC1 et dans CC2, affichez tous les départements. Cela confirme-t-il bien vos interprétations ?

4 - Contraintes d'intégrité et contraintes d'intégrité référentielles

4.1 : Créez la BD « empdeptTP03avecFK »

- Créez la BD empdeptTP03avecFK avec les tables « emp » et « dept » et les tuples (code des tables dans le chapitre 4 du cours).
- Vérifiez que la BD a été créée : SHOW DATABASES
- Vérifiez qu'elle contient les 2 tables : SHOW TABLES
- Vérifiez que les tables sont bien remplies : SELECT * …

4.2 : Regardez la table « emp »

- Faites un show create table emp.
- Quel est le moteur de la table.
- La clé primaire a-t-elle bien été créée ?
- Les clé étrangères ont-elles bien été créées ?
- L'auto-incrément a-t-il bien été créé ?
- Quelle sera le numéro de NE du prochain employé inserté ?

4.3 : Tests de contraintes d'intégrité référentielle

- Ajoutez le tuple suivant :
  insert into emp values (7900,'OBAMA','CLERK','1982-01-23',1300,NULL,10,7782);
- L'employé a-t-il été ajouté ? Quel problème cela pose-t-il ?

4.4 : Tests de contraintes d'intégrité référentielle

- Ajoutez le tuple suivant :
  insert into emp values (NULL,'OBAMA','CLERK','1982-01-23',1300,NULL,10,7782);
- L'employé a-t-il été ajouté ? Quel est la valeur du NE de l'employé ajouté.

4.5 : Tests de contraintes d'intégrité référentielle

- Ajoutez le tuple suivant :
  insert into emp values (NULL, 'TRUMP','CLERK','1982-01-23',1300,NULL,10,7781);
- L'employé a-t-il été ajouté ? Quel problème cela pose-t-il ?

4.6 : Tests de contraintes d'intégrité référentielle

- Ajoutez le tuple suivant :
  insert into emp values (NULL, 'TRUMP','CLERK','1982-01-23',1300,NULL,10,7781);
- L'employé a-t-il été ajouté ? Quel problème cela pose-t-il ?

4.7 : Tests de contraintes d'intégrité référentielle

- Supprimez le département n° 40.
- Le département a-t-il été supprimé ?

4.8 : Tests de contraintes d'intégrité référentielle

- Supprimez le département n° 30.

- Le département a-t-il été supprimé ? Quel problème cela pose-t-il ?

4.9 : Tests de contraintes d'intégrité référentielle

- Supprimez l'employé nommé OBAMA.
- L'employé a-t-il été supprimé ?

4.10 : Tests de contraintes d'intégrité référentielle

- Supprimez l'employé nommé BLAKE.
- L'employé a-t-il été supprimé ? Quel problème cela pose-t-il ?

5 - Index

5.1 : Créez la BD « empdeptTP03avecFK »

- Créez la BD empdeptTP03avecFK avec les tables « emp » et « dept » et les tuples (code des tables dans le chapitre 4 du cours).
- Vérifiez que la BD a été créée : SHOW DATABASES
- Vérifiez qu'elle contient les 2 tables : SHOW TABLES
- Vérifiez que les tables sont bien remplies : SELECT * …

5.2 : Regardez la table « dept »

- Faites un show create table emp.
- Combien y a-t-il d'index ?
- A quoi correspondent chaque index ?

5.3 : Regardez la table « emp »

- Faites un show create table emp.
- Combien y a-t-il d'index ?
- A quoi correspondent chaque index ?

5.4 : Créez un index

- Ajoutez un index sur le salaire.
- Vérifiez qu'il y a bien été pris en compte. Combien y a-t-il d'index ?