MySQL

Connection :

mysql ecole -u pterrier -p
mdp

Afficher les tables : « show tables; »

Afficher la structure de la table : « desc nomdelatable;

<=
<>

select nom, prenom from etudiant JOIN note ON note.numetudiant = etudiant.numetudiant;

Natural join :
mysql> select nom, prenom from etudiant NATURAL JOIN note;

Affiche tous les etudiant qui n'ont pas de note :

mysql> SELECT nom, prenom FROM etudiant NATURAL LEFT JOIN note where note is null and numepreuve is null;

Eleves qui n'ont pas de note a l'épreuve 2 :

mysql> SELECT nom, prenom FROM etudiant e LEFT JOIN note n ON e.numetudiant = n.numetudiant and numepreuve='2' where note is null;

moyenne a l'epreuve 2 :

SELECT AVG(note) FROM note NATURAL JOIN epreuve where numepreuve ='2';

group by =>

SELECT agrégat, *attribut*
FROM
WHERE ...
GROUP BY *attribut*

Pour chaque epreuve, la meilleure et la moins bonne note :

SELECT MAX(note), MIN(note), numepreuve FROM note n GROUP BY numepreuve;

Liste de tous les éleves (nom,prenom) avec leur moyenne s'ils en ont une (null sinon) par ordre croissant?

select nom,prenom,AVG(note) FROM etudiant NATURAL LEFT JOIN note group by numetudiant,nom,prenom ORDER BY nom ASC;

liste des etudiants qui passe ou on passé l'anglais ?

SELECT DISTINCT nom,prenom FROM etudiant NATURAL JOIN participation NATURAL JOIN epreuve NATURAL JOIN examen where matiere like 'anglais' order by nom;

Combien d'eleves passent ou ont passe une épreuve en septembre?

SELECT count(DISTINCT numetudiant) FROM etudiant NATURAL JOIN participation NATURAL JOIN epreuve NATURAL JOIN examen where session like 'septembre';

moy sup a 10 ?

select numetudiant, AVG(note) AS Moy FROM note GROUP BY numEtudiant HAVING Moy >= 10;

numero et effectif des groupes qui ont + de 7 eleves ?

select numgroupe, count(numetudiant) AS'eff' FROM participation GROUP BY numgroupe HAVING eff > 7;

<u>Inserer des données :</u>

INSERT INTO nomtable(liste des attributs)
VALUES (liste des valeurs);

L'étudiant Num 32, a eu 14 à l'epreuve 4 :

INSERT INTO note (numetudiant,numepreuve,note) VALUES(32,4,14);

Rajouter dans participation les éléves dont le nom commence par D, et qui sont dans le groupe 5 :

INSERT INTO participation (numgroupe,numetudiant) SELECT 5,numetudiant FROM etudiant NATURAL JOIN participation WHERE nom like 'D%';

CREATION DE TABLES

CREATE TABLE nomtable
(liste d'attributs
liste de propriétés
) ENGINE=myIsam/innodb;

TYPE :

varchar ...
Blob => fichier binaire (ex: file)

2 possibilités pour image ou photo:

stocké url (exige une mis a jour)
ou dans
un blob (très large traitement long)

NOT NULL

Default value

PRIMARY KEY (clé primaire)
REFERENCES unetable(attribut) FOREIGN KEY => (clé etrangere)

Ex:

```
CREATE TABLE etudiant (
numEtudiant INT PRIMARY KEY auto_increment,
nom VARCHAR(50),
prenom VARCHAR(50),
email VARCHAR(100)) ENGINE = InnoDB;
```

ATTENTION : InnoBD pour utilisation de clé étrangére, sinon myIsam!

Ex:

```
CREATE TABLE note(
numEpreuve INT,
numEtudiant INT,
note INT,
PRIMARY KEY (numEpreuve,numEtudiant));
```

pour visualiser la table créer en SQL :
SHOW CREATE TABLE table;

Exo 1:

```
CREATE TABLE entreprise (
    -> numEntreprise INT auto_increment,
    -> nom VARCHAR(50) NOT NULL,
    -> adresse VARCHAR(100) NULL,
    -> PRIMARY KEY (numEntreprise)) ENGINE = InnoDB;
```

Clé étrangére sous MySQL

1) *les 2 tables doivent être en InnoDB.*
2) *Créer un index sur la clé étrangére (Index nomindex (attribut))*
3) *Déclaration de la clé étrangére (attribut Types ... REFERENCES table(clé) ou attribut type FOREIGN KEY (attribut REFERENCES table (clé).*

Exo 2:

```
CREATE TABLE contact(
    -> numContact INT auto_increment,
    -> nom VARCHAR(50) NOT NULL,
    -> prenom VARCHAR(50) NULL,
    -> telephone VARCHAR(13) NULL,
    -> numEntreprise INT NOT NULL,
    -> PRIMARY KEY (numContact),
    -> FOREIGN KEY (numEntreprise) REFERENCES entreprise(numEntreprise),
    -> INDEX idxent (numEntreprise)) ENGINE=InnoDB;
Query OK, 0 rows affected (0.03 sec)
```

Exo 3:

```
CREATE TABLE stage (
    -> numEtudiant INT,
    -> numEntreprise INT NOT NULL,
    -> numContactResp INT NOT NULL,
    -> sujet varchar(250) NULL,
    -> dateDebut DATE NOT NULL,
    -> dateFin DATE NOT NULL,
    -> PRIMARY KEY (numEtudiant),
    -> FOREIGN KEY (numEtudiant) REFERENCES etudiant(numEtudiant),
    -> INDEX idxetud (numEtudiant),
    -> FOREIGN KEY (numEntreprise) REFERENCES entreprise(numEntreprise),
    -> INDEX idxent (numEntreprise),
    -> FOREIGN KEY (numContactResp) REFERENCES contact(numContact),
    -> INDEX idxcon (numContactResp)) ENGINE = InnoDB;
```

Nommage:

Index => nom d'index unique dans la base.
CONSTRAINT => CONSTRAINT nomdelacontrainte FOREIGN KEY cléetrangere REFERENCES table(cléprimaire)

Comportement lors de suppression/modif dans la table parent.

dans la table fille:

...
FOREIGN KEY (...)REFERENCES ... ON DELETE (NO ACTION/RESTRICT/CASCADE) SET NULL (mais NULL au jeux d'enregistremetn lié).

<u>Dans la table note</u> :

FOREIGN KEY (numEtudiant) REFERENCES etudiant(numEtudiant)
ON DELETE CASCADE ON UPDATE CASCADE

<u>Dans la table stage</u> :

FOREIGN KEY (numEtudiant) REFERENCES etudiant(numEtudiant)
..... ON UPDATE CASCADE,....

SUPPRIMER UNE TABLE

DROP TABLE nomtable;

a ne pas confondre avec DELETE FROM table; (Efface le contenu).

MODIFER UNE TABLE

ALTER TABLE nomtable modification;

RENOMMER UNE TABLE

ALTER TABLE nomtable RENAME newname;

ex: renommer la table stage ne stagiaire :

ALTER TABLE stage RENAME stagiaire;

RAJOUTER UN ATTRIBUT

ALTER TABLE table ADD COLUMN definition attribut(idem create table)

ex : Dans la table note, on rajoute un attribut coefficient entier, non null, qui vaut 1 par défault.

ALTER TABLE note ADD COLUMN coefficient INT NOT NULL DEFAULT 1;

PLACER SON ATTRIBUT

ALTER TABLE table ADD COLUMN definition attribut(idem create table) AFTER/FIRST attribut;

SUPPRIMER UNE COLONNE

ALTER TABLE table DROP COLUMN attribut;

MODIFIER UN ATTRIBUT

ALTER TABLE table MODIFY COLUMN nouvelledefinition (complète)

Ex : sujet -> varchar(70) non null

ALTER TABLE stagiaire MODIFY COLUMN sujet varchar(70) NOT NULL;

CHANGE COLUMN anciennom Nouvelledefinition; (compléte)

ex: Dans entreprise nom devient raisonSociale

ALTER TABLE entreprise CHANGE COLUMN nom raisonSociale varchar(50) NOT NULL;

ALTER COLUMN attribut SET DEFAULT valeur;

Suppression clé primaire

ALTER TABLE table DROP PRIMARY KEY;

Ajouter clé primaire

ALTER TABLE table ADD PRIMARY KEY(attributs);

Ex: stage numEtudiant n'est plus clé primaire a la place numStage:

ALTER TABLE stage DROP PRIMARY KEY;
ALTER TABLE stage ADD COLUMN numStage INT NOT NULL FIRST;
ALTER TABLE stage ADD PRIMARY KEY(numStage);
ALTER TABLE stage MODIFY COLUMN numStage INT NOT NULL auto_increment;

INDEX

ALTER TABLE stage :

- *=> DROP INDEX nomIndex;*
- *=> ADD INDEX nomIndex(attributs);*
- *=> DROP FOREIGN KEY nom constrainte;*

=> ADD FOREIGN KEY (...) REFERENCES....;
=> ADD CONSTRAINT nom FOREIGN KEY(...) REFERENCES...;

Table association :

numPresident enléve la clé étrangere et la recréer vers étudiant

numTrésorier/numSecrétaire
Rajouter FK vers étudiant +index;

ALTER TABLE association DROP FOREIGN KEY association_idxfk1;

ALTER TABLE association ADD FOREIGN KEY(numPresident) REFERENCES etudiant(numEtudiant);

ALTER TABLE association ADD INDEX (numSecrétaire);

ALTER TABLE association ADD INDEX (numSecrétaire) REFERENCES etudiant(numEtudiant);

Les sous-requêtes

1 Valeur

1 ensemble de valeur
1 ligne
1 tableau

SELECT ...
FROM ...
WHERE attribut=(SELECT ... FROM ... WHERE ...);

Numero des éléves + note dont la note est supérieur a la moyenne de la classe pour l'epreuve 2.

SELECT numEtudiant, note FROM note WHERE numEpreuve = 2 AND Note > (SELECT AVG(note) FROM note WHERE numEpreuve = 2);

Un Ensemble de valeurs:

IN |
NOT IN |

ANY |
ALL |
EXISTS
NOT EXISTS

Ex:

Etudiants qui n'ont pas de note a l'epreuve 2?

SELECT numEtudiant FROM etudiant WHERE numetudiant NOT IN (SELECT numEtudiant FROM note WHERE numEpreuve=2 AND note IS NOT NULL);

Numero de l'etudiant qui a la meilleure ntoe a l'epreuve 1 ?

SELECT numEtudiant, note FROM note WHERE numEpreuve=1 AND note >= ALL(SELECT note FROM note WHERE numEpreuve=1);

ROW (Attri1, attri2,...) = (SELECT a1,a2... FROM ... WHERE)

Attri1 et a1 => même nombre de valeurs.

Pour toutes les epreuves le numero (et la note) de l'eleve qui a eu la meilleure note?

SELECT numEtudiant, numEpreuve, note FROM note WHERE (numEpreuve, note) IN (SELECT numEpreuve,Max(note) FROM note GROUP BY numEpreuve);

La listes des examens qui n'ont pas d'epreuve?

SELECT numExamen FROM examen WHERE numExamen NOT IN (SELECT numexamen FROM epreuve);

Pour chaque etudiant, quelle est l'epreuve ou il a la moins bonne note?(avec sa note)

SELECT ne.numEtudiant, numEpreuve, note FROM note ne WHERE note is not null AND note = (SELECT MIN(note) FROM note WHERE numetudiant = ne.numetudiant);

ou

*SELECT * FROM note WHERE (numEtudiant,note) IN (SELECT numEtudiant, MIN(note) FROM note GROUP BY numEtudiant);*

Quelle est l'epreuve qui a la meilleure moyenne?

SELECT numEpreuve, AVG(note) AS 'moy' FROM note GROUP BY numEpreuve HAVING moy >= ALL (SELECT AVG(note) AS 'm' FROM note GROUP BY numEpreuve);

etudiants qui ont une note pour toutes les épreuves (notées)?

SELECT numEtudiant, count(note) AS 'nbrnote' FROM note GROUP BY numEtudiant HAVING nbrnote = (SELECT count(DISTINCT numEpreuve) FROM note);

Les vues

Une table virtuelle définie par une requete de type select, et qui va être recalculer a chaque fois qu'on en a besoin.

CREATE VIEW rayon_livres AS SELECT * FROM produits WHERE type LIKE 'livre';

SELECT reference FROM rayon_livres;

SELECT reference FROM (SELECT * FROM produits WHERE type LIKE 'livre') AS rayon_livre;

créer une vue moy_by_epreuve, qui contient les moyennes par epreuve

CREATE VIEW moy_by_epreuve AS SELECT numepreuve,AVG(note) AS 'moy_ep'FROM note GROUP BY numepreuve;

Quelle est l'epreuve qui a la meilleur moy?

SELECT numepreuve FROM moy_by_epreuve WHERE moy_ep = (SELECT MAX(moy_ep) FROM moy_by_epreuve);

Metablilité a jour d'une vue (updatability) en insertion/modification/suppression

numEtudiant;numEpreuve,note?
CreerVUE V_notes_epr_2

CREATE VIEW V_notes_epr_2 AS SELECT numEtudiant,numEpreuve,note FROM note WHERE numEpreuve like 2;

inserer une nouvelle note?
INSERT INTO V_notes_epr_2 VALUES(6,2,20);

CreerVUE V_notes_epr_3

CREATE VIEW V_notes_epr_3 AS SELECT * FROM note WHERE numEpreuve=3 WHITH CHECK OPTION

CHECK OPTION => (verifie si l'insert correspondbien a l 'epreuve 3!!)

INSERT INTO V_notes_epr_3 VALUES(7,2,18);
ERROR 1369 (HY000): CHECK OPTION failed 'ecole.V_notes_epr_3'

INSERT INTO V_notes_epr_3 VALUES(7,3,18);
Query OK, 1 row affected (0.01 sec)

V_notes_grpe_1

=> notes (tous les attributs) des etudiants appartenant au groupe1 (sans check option);
participation

s'en servir pour inserer une nouvelle note

CREATE VIEW v_notes_grpe_1 AS SELECT n.* FROM note n NATURAL JOIN participation p WHERE numgroupe = 1;

INSERT INTO v_notes_grpe_1 (numEtudiant,numEpreuve,note) VALUES (13,3,20);

Commandes relatives aux verrous et aux transactions

Syntaxes de `START TRANSACTION`, `COMMIT` et `ROLLBACK`

Par défaut, MySQL est lancé en mode `autocommit`. Cela signifie que chaque modification effectué est enregistré immédiatement sur le disque par MySQL.

Si vous utilisez des tables supportant les transactions (comme `InnoDB`, `BDB`), vous pouvez configurer MySQL en mode non-`autocommit` grâce à la commande:

```
SET AUTOCOMMIT=0
```

A partir de là, vous devez utiliser `COMMIT` pour enregistrer les modifications sur le disque ou `ROLLBACK` pour ignorer les modifications apportées depuis le début de la transaction.

Si vous souhaitez sortir du mode `AUTOCOMMIT` pour une série d'opérations, vous pouvez utiliser les commandes `BEGIN` ou `BEGIN WORK` :

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summmary=@A WHERE type=1;
COMMIT;
```

`BEGIN` et `BEGIN WORK` peuvent être utilisé à la place de `START TRANSACTION` pour initialiser une transaction. `START TRANSACTION` a été ajouté en MySQL 4.0.11; C'est une syntaxe SQL-99, et il est recommandé de l'utiliser pour lancer une transaction. `BEGIN` et `BEGIN WORK` sont disponibles pour MySQL 3.23.17 et 3.23.19, respectivement.

Notez que si vous n'utilisez pas de table transactionnelles, les modifications seront validées automatiquement, indépendamment du mode de validation.

Si vous faites un `ROLLBACK` après avoir modifié une table non transactionnelle, vous obtiendrez (`ER_WARNING_NOT_COMPLETE_ROLLBACK`) comme message d'alerte. Toutes les tables supportant les transactions seront restaurées, mais aucune des autres tables ne changera.

Si vous utilisez `START TRANSACTION` ou `SET AUTOCOMMIT=0`, il est recommandé d'utiliser les "`binary log`" de MySQL à la place des anciens logs de modifications pour les sauvegardes. Les transactions sont stockées dans les logs binaires en un seul bloc, après `COMMIT`, pour être sûr que les transactions qui ont été annulées ne soient pas enregistrées. See Section 5.9.4, « Le log binaire ».

Vous pouvez changer le niveau d'isolation des transactions avec `SET TRANSACTION ISOLATION LEVEL ...`. See Section 13.4.6, « Syntaxe de SET TRANSACTION »

ex :

1er Terminal :

START TRANSACTION;
UPDATE compte SET sold=solde -100 WHERE idCompte =1;

2eme Terminal :

START TRANSACTION;
UPDATE compte SET sold=solde -100 WHERE idCompte =1;

INTERBLOCAGE

SUICIDE de TRANSACTION en cas de time out;
ou
ASSASINA : kill par l'admin DBA.

Niveau d'isolation :

START TRANSACTION
START TRANSACTION
(2)(2)(2)
SELECT *FROM compte;
INSERT INTO compte
.....;
(2)(2)
SELECT * FROM compte;
COMMIT;

SELECT * FROM compte;

iptables-read (lecture reproductive) : 2 fois le mm select (lecture).
read-commited : voit au fur et a mesure les validations
read-uncommited : dés que je fait une action je la voie ().

SELECT @@tx_isolcation;
@@ => Variables de session
tx_isolation => nom de la variabe

SET TRANSACTION ISOLATION LEVEL « niveau » (serializable,repeatable-read,read-commited,uncommited)GLOBAL (pour toute les sessions).

START TRANSACTION;

SELECT * FROM compte; (2 comptes)
=>auto_incr=3;

SELECT *FROM COMPTE

INSERT INTO compte (idClient,solde) auto_incr?

COMMIT;

autocommit=0; => on sort de l'autocommit
par défault autocommit=1;

Variables :

session:
@@toto

créer une var :
@variable:=valeur;

SELECT ... FROM WHERE... INTO @variabe;

Afficher :

SELECT @mavariable;

PREPARE nom FROM 'SELECT .... FROM ... WHERE .... = ?'

EXECUTE nom USING @var;

Inconvenient de PREPARE : 1 seul requete

Procedure stocker :

CREATE PROCEDURE non(pram,param...)

requetes;

PARAMETRE IN non type
OUT
IN OUT

Exemple :
Procedure « Affiche solde ».
En entré un numero de compte.

CREATE PROCEDURE Affiche_solde(IN Numcpte INT)
SELECT solde FROM compte WHERE idCompte = Numcpte ;

Appel :

call Affiche_solde(1);
mysql> call Aff_solde(1);

```
+-------+
| solde |
+-------+
|  1276 |
+-------+
1 row in set (0.06 sec)

Query OK, 0 rows affected (0.06 sec)
```

Block de procedure :

```
DELEMITER //
CREATE PROCEDURE affich(PARAM)
        BEGIN
                .....;
                ......;
                .......;
        END;//

DELEMITER ;
```

Dans le BEGIN ... END; on peu faire des structures de contrôle :

```
BEGIN
DECLARE nom type; (Variable interne)
        IF test
                THEN ....;
                ELSE ....;
        END IF;

END;//
```

Exemple :

```
DELEMITER //
CREATE PROCEDURE virement( IN numCpteEmet, numCpteRecep, montant)
        BEGIN
                DECLARE soldeEmet int;

                SELECT solde FROM compte WHERE idCompte = numCpteEmet INTO
@soldeEmet ;

                IF soldeEmet > montant
                THEN
                        BEGIN
                        UPDATE compte SET solde=(solde+montant) WHERE idCompte
                        = numCpteRecep;
                        UPDATE compte SET solde=(solde-montant) WHERE idCompte
                        = numCpteEmet;
                        END
                ELSE
                        SELECT 'solde insuffisant' AS 'Message';
```

```
            END IF;
        END;//

DELEMITER ;
```

On peu integrer des transaction dans les procédures stocké.
On peu appeler une autre procédures stocké dans une procédures stocké.

Les Déclencheurs et les triggers

CREATE TRIGGER nom BEFORE/AFTER INSERT/UPDATE/DELETE ON table;

old.attribut
new.attribut