BASE DE DONNEES RELATIONNELLE
Algèbre relationnelle et SQL

3 : DDL - DML

Manuels de référence du SQL :
Voir après le sommaire

Bertrand Liaudet

SOMMAIRE

Edition : septembre 2019

Manuels de référence du SQL

Voir le cours n°1 : select mono tables.
http://bliaudet.free.fr/article.php3?id_article=152

Le SQL est standard : la syntaxe est la même pour tous les SGBD. Les exemples sont proposés en général avec MySQL. Les spécificités sont présentées par SGBD.

INTRODUCTION

CRUD

Principes

- Les trois opérations fondamentales de gestion d'une base de données sont les :
  - La création
  - La modification
  - La suppression
- Ces opérations correspondent au sigle : **CMS** On parle aussi de « **SIUD** » pour Select, Insert, Update et Delete ou aussi « **CRUD** » pour **Create**, **Replace**, **Update**, **Delete**.
- **CRUD** est le terme le plus usuel. Il concerne aussi bien le DDL que le DML ou le DCL.

CRUD des tables : Le DDL : Data Definition Language

- CREATE TABLE
- ALTER TABLE
- DROP TABLE

CRUD des tuples : Le DML : Data Manipulation Language

- INSERT INTO
- UPDATE
- DELETE FROM

CRUD des utilisateurs et de leurs droits : Le DCL : Data Control Language

- CREATE USER
- ALTER USER
- DROP USER
- GRANT : créer des droits
- REVOKE : supprimer des droits

Commit et Rollback

- A ces opérations, s'ajoutent les opérations qui vont permettre de garantir la cohérence des données à travers la gestion des transactions : **COMMIT et ROLLBACK**.

SQL : langage standard pour tous les SGBD

- Les opérations du DDL, DML et DCL sont standardisées : ce sont les même sous MySQL, ORACLE, SQL-server, PostgreSQL, etc.
- Toutefois, il peut y avoir des spécificités selon les SGBDs.
- Il convient donc d'en revenir à la documentation officielle de chaque SGBD.

CREATE DATABASE - Création de la base de données (ou schéma)

- La manipulation de la BD (CREATE DATABASE, DROP DATABASE) ne relève pas de l'algèbre relationnelle mais d'opérations d'administration qui sont **spécifiques à chaque SGBD**.
- Le principe est toutefois de pouvoir créer, modifier ou supprimer une BD (**CRUD**) en précisant au minimum son nom, et ensuite de pouvoir la consulter, c'est-à-dire consulter les objets qu'elle contient : les tables, les utilisateurs, etc.
- On peut ensuite paramétrer de nombreux éléments de la BD : la taille de l'**espace physique** qui lui sera alloué, son propriétaire, le **moteur** (engine) pour MySQL, etc.
- Ces paramétrages sont **spécifiques** à chaque SGBD.
- Avec MySQL, la création d'une BD est simple et s'apparente, en première approche, à la création d'un dossier dans un OS. La suppression est aussi simple.

SQL : CREATION DE LA STRUCTURE DE LA BD : DDL

1. Gestion des tables: DDL

Une fois le MR réalisé, il reste à créer la base de données. Créer d'abord les tables, puis les tuples.

CRUD des tables : Le DDL : Data Definition Language

- CREATE TABLE
- ALTER TABLE
- DROP TABLE

DDL et environnement graphique : phpMyAdmin, WorkBench

- Les commandes du DDL peuvent se faire aisément dans un environnement graphique, particulièrement pour le ALTER TABLE.
- Les environnements graphiques montrent en général la commande SQL à laquelle correspond la manipulation graphique.

Attention

- La modification et la destruction des tables et des colonnes doivent être manipulées avec prudence: une table peut contenir des milliers de données. Il ne faut pas les supprimer ou modifier une table sans précaution.
- Le DDL génère un COMMIT automatiquement : il n'y a pas de possibilité de ROLLBACK.

Show Create Table : spécificité MySQL

- La commande SHOW CREATE TABLE permet de voir le code de création de la table tel qu'il est enregistré par le serveur.
- Ainsi, on voit apparaître le nom de contraintes, les index, la valeur de l'auto-incrément, le moteur, etc.

```
SHOW CREATE TABLE maTable
```

2. Create Table et Drop Table

CRUD : CREATE TABLE = Création des tables

Création d'une table

- La commande a la syntaxe suivante :

```
CREATE table NomTable  (
   attribut_1  type [contrainte d'intégrité],
   attribut_2  type [contrainte d'intégrité],
   … ,
   attribut_n  type [contrainte d'intégrité] ,
   [contrainte d'intégrité]
);
```

- Le **type** est **obligatoire**
- Les **contraintes** sont **facultatives**.
- **L'ordre** dans la liste est **au choix**.

La table est une coquille vide : c'est une structure. Elle permettra ensuite de créer des tuples.

Exemple

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

CREATE TABLE EMP (
 NE          integer primary key auto_increment,
 NOM         varchar(10) not NULL,
 JOB         enum ('PRESIDENT','MANAGER', 'SALESMAN', 'CLERK',
'ANALYST'),
 DATEMB      date,
 SAL         float(7,2),
 COMM        float(7,2),
 ND          integer not null, foreign key(ND) references
DEPT(ND),
 NEchef      integer , foreign key(NEchef) references EMP(NE)
);
```

Documentation MySQL

http://dev.mysql.com/doc/refman/8.0/en/create-table.html

CRUD : CREATE TABLE … AS SELECT = create table + insert into

Principe

- On peut créer une table et la remplir à partir d'un select.

https://dev.mysql.com/doc/refman/8.0/en/create-table-select.html

Version sans attributs définis dans la table

```
CREATE TABLE nomTable AS SELECT … ;
```

- Cette table contiendra les attributs et les tuples du SELECT.
- Les types seront ceux des tables du SELECT.
- Les contraintes « obligatoire » et « valeur par défaut » seront ceux des tables du SELECT.
- Les clés primaires, clés étrangères et contraintes d'unicité ne sont pas conservées.
- Pour y ajouter des contraintes d'intégrité, il faudra faire des ALTER TABLE.

Version sans attributs définis dans la table

```
CREATE TABLE nomTable (...) AS SELECT … ;
```

- Les types et les contraintes seront ceux définis dans le CREATE TABLE.
- Les tuples seront ceux du SELECT.

CRUD : DROP TABLE = Suppression d'une table

https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_9003.htm

```
DROP TABLE NomTable ;
```

3 Alter table - Modification des tables

Principe de la syntaxe : ADD, MODIFY, DROP

ALTER TABLE NomTable ADD ou MODIFY ou DROP

Ce qu'on veut ajouter, comme dans la création

On peut faire des ALTER pour :

- ajouter : ADD
- modifier : MODIFY
- supprimer : DROP

On peut ajouter, modifier ou supprimer :

- des attributs
- des contraintes

Ajouter, modifier, supprimer des attributs

Ajouter un ou plusieurs attributs à la table :

```
ALTER TABLE NomTable ADD (
        attribut_1  type [contrainte],
        attribut_2  type [contrainte],
        … ,
        attribut_n  type [contrainte]
);
```

Modifier un attribut de la table

```
ALTER TABLE NomTable  MODIFY
      attribut_1  type [contrainte]
;
```

- La modification permet d'annuler les contraintes de type NOT NULL ou auto_increment.

Supprimer un attribut de la table

```
ALTER TABLE NomTable DROP attribut ;
```

➢ ***Attention***

- La modification et la suppression des attributs doivent être manipulées avec prudence : une table peut contenir des milliers de données. Il ne faut pas les supprimer ou modifier une table sans précaution.

Ajout et suppression de contraintes d'intégrité

Ajouter une contrainte

```
ALTER TABLE NomTable
   ADD CONSTRAINT  [contrainte] ;
```

- ***Exemple***

```
ALTER TABLE emp ADD
   CONSTRAINT keynd FOREIGN KEY(ND) REFERENCES DEPT(ND);
```

Supprimer la clé primaire

```
ALTER TABLE NomTable
   DROP primary key ;
```

- On ne peut supprimer la clé primaire que si ce n'est pas un auto incrément, et uniquement si elle n'est pas référencée par une clé étrangère.

Suppression d'une contrainte nommée

```
ALTER TABLE NomTable
   DROP CONSTRAINT nom_de_contrainte;
```

- ***Exemple***

```
ALTER TABLE emp DROP foreign key  KEYND;
```

Manuel de référence de l'alter table

Manuel de référence MySQL : http://dev.mysql.com/doc/refman/5.7/en/alter-table.html
Autre SGBD : à chercher !

4. Les types

Présentation globale des différents types

La connaissance des types est plus ou moins spécifique au SGBD avec lequel on travaille.

Les types par SGBD

ORACLE

http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/datatype.htm

MySQL

http://dev.mysql.com/doc/refman/5.7/en/numeric-type-overview.html

PostgreSQL

http://www.postgresql.org/docs/9.0/static/datatype.html

SQL Server

http://technet.microsoft.com/fr-fr/library/ms172424%28SQL.90%29.aspx

http://dev.mysql.com/doc/refman/5.7/en/data-types.html

Les types numériques

- **BOOL**, BOOLEAN, BIT
- **INT** (M), INTEGER (M), TINYINT (M), SMALLINT (M), MEDIUMINT (M), BIGINT (M)
- **FLOAT**(precision), FLOAT (M,D), DOUBLE (M,D), DOUBLE PRECISION (M,D), REAL (M,D) avec : M : nb chiffres max, D : nb chiffres après la virgule
- **DECIMAL** (M,D), DEC (M,D)
- avec M : taille de l'affichage, D: nombre de décimal, M>=D, si M=D alors –1 < n < 1.

Les types date

http://dev.mysql.com/doc/refman/5.7/en/date-and-time-type-overview.html

- **DATE**, **DATETIME**, **TIME**
- **TIMESTAMP** : pour gérer automatiquement la date de création ou de mise à jour.
- **YEAR**(2|4)

Type chaîne

http://dev.mysql.com/doc/refman/5.7/en/string-type-overview.html

- **CHAR**(M), NCHAR(M), CHARACTER : taille statique, 255 max.
- **VARCHAR**(M), NATIONAL VARCHAR(M) : taille dynamique, 2000 max.
- **TEXT**, TINYTEXT, MEDIUMTEXT, LONGTEXT : taille dynamique, jusqu'à 4GO (pour le LONGTEXT, sinon 65535 caractères pour le TEXT).

Type chaine : enum et set

http://dev.mysql.com/doc/refman/5.7/en/enum.html
http://dev.mysql.com/doc/refman/5.7/en/set.html

- **ENUM**('value1','value2',...) : permet de proposer une liste de valeurs pour l'attribut. Remarque : les valeurs proposées seront les seules possibles. L'enum se comporte donc comme un « check ».
- **SET**('value1','value2',...) : zéro, une ou plusieurs valeurs parmi une liste

BLOB

- **B**inary **L**ong **OB**ject: : pour stocker des mégas ou des gigas d'octets.
- **BLOB,** TINYBLOB, MEDIUMBLOB, LONGBLOB : taille dynamique, jusqu'à 4GO.

BFILE, FILESTREAM

- Types pour gérer des données binaires stockées dans un fichier externe à la base. BFILE : ORACLE. FILESTREAM : SQL Server.

Autres types

- Selon les SGBD, on trouvera différents autres types pour gérer des données particulières. Notons particulièrement :
  - Un **type XML** (SQL Server) : pour gérer des données au format XML. Ce type est associé à des opérateurs spécifiques pour pouvoir interroger les données XML à partir de la BD.
  - Des **types géographiques** (SQL Server) : pour gérer des données cartographiques et géographiques.
  -

Transtypage : cast, convert, round, truncate, etc.

Principes

Il peut être utile de passer d'un type à un autre dans les calcul : c'est ce qu'on appelle le « transtypage ». On dit qu'on « cast » la variable.
Il existe différentes techniques pour faire cela qui peuvent varier selon les SGBD.

cast

```
SELECT cast('19.4' as signed );                 // 19
SELECT cast( 4/3 as decimal(2,1) );   // 1.3
```

convert

```
SELECT convert ( 4/3, decimal(2,0) );  // 1
SELECT convert ( 4/3, decimal(2,1) );  // 1.3
SELECT convert ( 4/3, decimal(3,2) );  // 1.33
SELECT convert ( 4/3, decimal(2,2) );  // 0.99 !!! ATTENTION !!!
SELECT convert ( 4/3, decimal(1,2) );  // ERREUR !!!
```

Remarque

Pour faire des arrondis ou pour tronquer la partie décimale, il existe en général des fonctions qui rendent très bien le service.

- ***Par exemple, sous MySQL :***

ceil (nb) : arrondi vers le haut ;
floor (nb) : arrondi vers le bas ;
round (nb) arrondi au plus proche ;
truncate (nb, nbDécimales) : arrondi en vers le bas en précisant le nombre de décimales conservées ;
http://dev.mysql.com/doc/refman/5.7/en/mathematical-functions.html

5. Contraintes d'intégrité : DDL

Présentation

Notion de contrainte d'intégrité

- Une contrainte d'intégrité est un élément de définition des données qui s'applique à un attribut et qui fixe une contrainte de valeur pour les données de l'attribut.
- C'est une régle que le serveur vérifiera à chaque action du DDL et du DML pour s'assurer que le modèle et les données sont conformes.
- Les contraintes d'intégrité vont permettre de :
  - définir les **clés primaires** et les **clés étrangères**.
  - préciser les **valeurs possibles** pour les attributs.

Exemple de syntaxe

Le CREATE table intègre les contraintes d'intégrité comme suit :

```
CREATE table NomTable  (
       attribut_1  type [contrainte d'intégrité],
       attribut_2  type [contrainte d'intégrité],
       … ,
       attribut_n  type [contrainte d'intégrité] ,
       [contrainte d'intégrité]
);
```

- Le type n'est pas classé dans les contraintes d'intégrité mais c'en est une, obligatoire de surcroît.

Les principales contraintes

Les différentes contraintes que l'on peut déclarer en plus du type sont les suivantes :

- **PRIMARY KEY :** permet de définir les clés primaires. Cette contrainte garantit le fait que la valeur est différente de NULL et qu'elle est unique dans la table.
- **FOREIGN KEY :** permet de définir les clés étrangères. Cette contrainte fait référence à une clé primaire dans une autre table. C'est cette contrainte qu'on appelle « contrainte d'intégrité référentielle ». **Cette contrainte ne peut être créée qu'à condition qu'elle fasse référence à une clé primaire déjà créée dans la BD.**
- **NOT NULL :** impose le fait que la valeur de l'attribut doit être renseignée.
- **UNIQUE :** impose le fait que chaque tuple de la table doit, pour l'attribut concerné, avoir une valeur différente de celle des autres ou NULL.
- **DEFAULT :** permet de proposer une valeur par défaut. Si rien n'est saisi à la création du tuple, c'est la valeur par défaut qui sera fournie. A noter qu'on peut saisir NULL s'il y a une valeur par défaut.
- **CHECK ou ENUM :** permet de définir un ensemble de valeurs possible pour l'attribut. Cette contrainte garantit le fait que la valeur de l'attribut appartiendra à cet ensemble. CHECK est standard. MySQL ne gère pas les CHECK. MySQL gère un ENUM qui n'est pas standard.

Le type

- Cette contrainte permet d'éviter qu'on saisisse n'importe quelle valeur.
- Toutefois selon les SGBD, plus ou moins de vérifications seront effectuées.
  - Si on cherche à entrer du texte à la place d'un numérique, l'INSERT est refusé.
  - Si on cherche à entrer une date au mauvais format, l'INSERT est refusé. Les date doivent avoir le format : 'yyyy-mm-jj' et le nombre de jour et nombre de mois doit être cohérent.

MySQL est assez permissif

- Si on cherche à entrer un réel à la place d'un entier, la partie décimale est supprimée.
- Si on cherche à entrer un réel à la place d'une chaine de caractères, le réel est considéré comme une chaine de caractères.
- Si on entre un numérique avec des apostrophes ou des guillemets à la place d'un numérique, il est accepté.

PRIMARY KEY, NOT NULL, UNIQUE et CHECK

- Ces quatre contraintes ont le même type de conséquence :
  - si on cherche à donner une valeur à un attribut qui n'est pas conforme à ce qui est précisé dans la définition de l'attribut :
    - valeur NULL s'il est défini NOT NULL ou PRIMARY KEY,
    - valeur existant déjà s'il est défini UNIQUE ou PRIMARY KEY,
    - valeur n'appartenant pas au domaine spécifié par le CHECK ou l'ENUM
  - alors le SGBD renvoie un message d'erreur et ne modifie pas la base de données.
- Ainsi, un premier niveau de cohérence des données est maintenu.

Auto-Incrément

Présentation

- Les SGBD-R permettent de gérer des auto-incréments qui sont utiles pour une numérotation automatique des clés primaires.
- Le principe d'un auto-incrément est que la nouvelle valeur d'un attribut est égale à la dernière valeur entrée + 1.
- La syntaxe des auto-incréments est très différente selon les SGBD.
  - Dans **MySQL**, il suffit d'ajouter la contrainte « **auto_increment** » à un attribut.
  - Sous **ORACLE**, il faut créer un objet « Sequence » qu'on utilise ensuite pour gérer l'auto-incrémentation d'un attribut.
  - Sous **SQL-Server**, il faut utiliser la notion d'IDENTITY.

Syntaxe DDL MySQL

```
CREATE TABLE nomTable (
 attribut        integer auto_increment,
 …
```

Clés constituées de plusieurs attributs

```
PRIMARY KEY (attribut1, attribut2)
```

Différents cas de CHECK

Syntaxe

```
CHECK (condition de recherche)
```

La syntaxe des conditions de recherche est la même que celle qu'on trouvera au moment de l'interrogation de la base de données.

Exemples :

```
job char(9) CHECK( job IN ('SALESMAN', 'MANAGER', 'PRESIDENT'))
```

- ou alors

```
CHECK (attribut < 100),
CHECK (attribut <100 and attribut >50),
CHECK (attribut between 50 and 100),
CHECK (attribut not between 50 and 100)
CHECK (attribut1 > attribut2)
```

- A noter que MySQL ne gère pas les CHECK. Toutefois, si on code avec un CHECK c'est considéré comme du commentaire par MySQL.

timestamp : enregistrement automatique de la date de modification

- Selon les SGBD, d'autres possibilités de définition de contraintes d'intégrité seront possibles. Elles sont à analyser au cas par cas en fonction du SGBD sur lequel on travaille.

MySQL : date de création automatique

```
Create table test ( …
    dateCrea     timestamp default now( ),
… )
```

- L'attribut sera affecté automatiquement à « now » à la création d'un tuple.

MySQL : date de modification automatique

```
Create table test ( …
     dateModif      timestamp default now( ) on update
current_timestamp default now( ),
 … )
```

- L'attribut sera affecté automatiquement à « now » à la création d'un tuple puis remis à now à chaque update.
-
- A noter que MySQL ne permet pas l'usage des deux possibilités dans une même table.

6. Foreign Key - Contrainte d'intégrité référentielle : CIR

1 - Notion de contrainte d'intégrité référentielle

Définition

- Une CIR garantit que la valeur de l'attribut fait référence à une clé primaire existant dans une autre table.

Syntaxe

➢ ***Syntaxe générale***

```
[CONSTRAINT nomContrainte]
FOREIGN KEY (nomAttMaître )
REFERENCES nomTableJointe( nomAttJoint )
```

➢ ***Exemple***

```
CREATE TABLE EMP (
        …
FOREIGN KEY (ND) REFERENCES DEPT(ND),
) ;
```

- Une CIR concerne deux attributs (et donc en général deux tables) :
  - Une CIR s'applique à **un attribut maître** : celui qui fait référence à un autre attribut.
  - Une CIR concerne **un attribut joint** : celui qui est référé par l'attribut maître.

Principale conséquence

- Cette contrainte garantit que la valeur fait bien référence à une clé primaire existant dans une autre table.
  - Si on cherche à entrer une valeur qui ne fait pas référence à une clé primaire existant dans une autre table, l'INSERT est refusé.

Toutes les conséquences d'une CIR

- La présence de CIR à des conséquences :
  - Sur la création, modification, suppression des tables.
  - Sur la création et la modification de tuples propriétaires de la CIR.
  - Sur la modification et la suppression de tuples référés par une CIR.

2 - Conséquences sur les tables

La table jointe doit être créée avant la table maître.

- **Exemple :** il faut créer la table des départements avant celle des employés car la table des employés fait référence à la table des départements.

La table maître doit être supprimée avant la table jointe.

- **Exemple :** il faut supprimer la table des employés avant celle des départements car la table aucune table ne fait référence à la table des employés tandis que la table des départements est référencée par la table des employés.

Création des tables dans n'importe quel ordre

- Pour pouvoir créer les tables dans n'importe quel ordre (par ordre alphabétique par exemple), il suffit de créer les CIR après avoir créer les tables (avec un ALTER TABLE).

Suppression des tables dans n'importe quel ordre

- Pour pouvoir supprimer les tables dans n'importe quel ordre, il suffit de supprimer les CIR avant de supprimer les tables (avec un ALTER TABLE).

3 - Conséquences sur les tuples

Le tuple joint doit être créé avant le tuple maître

- **Exemple :** il faut créer le tuple département 10 avant de créer un employé qui travaille dans le département 10.

Un tuple joint ne peut pas être supprimé

- **Exemple :** pour supprimer le département 10 il faut qu'il n'y ait plus d'employé qui travaillent dans le département 10.

La clé primaire d'un tuple joint ne peut pas être modifié

- **Exemple :** pour modifier le numéro du département 10 il faut qu'il n'y ait plus d'employé qui travaillent dans le département 10.
- En cas de tentative d'une telle suppression, 3 cas peuvent se présenter :

4 - Alternatives aux interdictions

ON DELETE CASCADE

- Pour permettre la **suppression d'un tuple joint**, on peut ajouter ON DELETE CASCADE dans la définition de la clé étrangère. Ainsi, tous les tuples maitres sont supprimés.
- Il faudra alors commencer par détruire le ou les tuples maîtres correspondant.

➢ ***Exemple :***

```
ND integer not null,
foreign key(ND) references DEPT(ND) ON DELETE CASCADE
```

Si on supprime un département, on supprimera aussi tous les employés du départements… ce qui n'est certainement pas un bon choix de modélisation !
Le DELETE CASCADE correspond à une situation de composition : quand on supprime le composé, tous les composant qui font référence au composé sont supprimés.

ON DELETE SET NULL

- Pour permettre la **suppression d'un tuple joint**, on peut ajouter ON DELETE SET NULL dans la définition de la clé étrangère. Ainsi, tous les tuples maitres seront mis à NULL. Donc plus aucun tuple maitres ne fera référence au tuple joint supprimé.

➢ ***Exemple :***

```
ND integer not null,
foreign key(ND) references DEPT(ND) ON DELETE SET NULL
```

Dans notre exemple, cela signifie que si on supprime un département, les employés du département auront désormais la valeur NULL comme numéro de département. C'est possible à condition que le numéro de département de l'employé ne soit pas déclaré NOT NULL.

➢ ***ON UPDATE CASCADE***

- Pour permettre la **modification d'un tuple joint**, on peut ajouter ON UPDATE CASCADE dans la définition de la clé étrangère. Ainsi, tous les tuples maitres seront mis à la nouvelle valeur du tuple joint. Donc plus aucun tuple maitre ne fera référence à l'ancienne valeur du tuple joint.

➢ ***Exemple :***

```
ND integer not null,
foreign key(ND) references DEPT(ND) ON UPDATE CASCADE
```

Dans notre exemple, cela signifie que si on modifie la clé primaire d'un département, on modifiera aussi les numéros de départements des employés de ce département.

5 – On Delete Restrict

- On voit souvent dans les codes : ON DELETE RESTRICT. Cette précision ne sert à rien ! Elle est équivalente à ne rien mettre et au cas général du fonctionnement des CIR !

6 – Clé étrangère constituée de plusieurs attributs

- Une clé étrangère peut faire référence à une clé primaire constituée de plusieurs attributs.

```
foreign key(a1, a2) references TAB(a1, a2)
```

7 - Contrôle des contraintes

Nommage automatique des contraintes

- Les contraintes sont nommées automatiquement par le serveur.
- Le code :

```
CREATE TABLE employes (
  ...
foreign key(ND) references departements(ND)
);
```

- Devient avec un show create table :

```
| employes | CREATE TABLE `employes` (
CONSTRAINT `employes_ibfk_1` FOREIGN KEY (`ND`) REFERENCES `departements`
(`ND`),
)
```

- Le nom 'employes_ibfk_1' a été ajouté.

Nommage manuel des contraintes

- Pour nommer les contraintes, il suffit d'ajouter « **CONSTRAINT nomContrainte** » devant la déclaration de la contrainte.

➢ ***Exemple :***

```
CREATE TABLE employes (
  ...
CONSTRAINT keyND FOREIGN KEY(ND) REFERENCES departements(ND)
);
```

Activation et désactivation des contraintes

- Nommer toutes les contraintes permet ensuite de **désactiver** et de **réactiver** les contraintes en y faisant références par leur nom.
- Selon les SGBD, on peut désactiver une contrainte avec un ALTER TABLE en précisant son nom.
- On peut aussi désactiver toutes les contraintes d'un coup pour faire des modifications en masse.

➢ ***Désactiver une contrainte nommée***

```
ALTER TABLE NomTable
   DISABLE CONSTRAINT nom_de_contrainte;
```

➢ ***Activer une contrainte nommée***

```
ALTER TABLE NomTable
   ENABLE CONSTRAINT nom_de_contrainte;
```

➢ ***Désactiver et réactiver toutes les contraintes avec MySQL***

- La variable foreign_key_checks permet de faire ça.

```
SHOW VARIABLES LIKE '%foreign%';
SET FOREIGN_KEY_CHECKS = OFF
```

- On peut mettre OFF ou 0 pour désactiver les contraintes. Puis ON ou 1 pour les réactiver.

7. Les vues

Présentation

- Une commande **SELECT** peut être conservée dans un objet de la BD appelé "**VIEW**" (vue).

- En tant que table, une vue n'a qu'une existence virtuelle :
  - Les tuples n'ont pas d'existence physique ;
  - La vue est recalculée à chaque utilisation ;

- La vue est équivalente à une table dans sa structure externe : **elle a des attributs** comme une table.
- La vue est équivalente à une table pour la consultation : **on peut faire des SELECT** sur une vue comme sur une table. Toutefois, le contenu de la vue n'a pas d'existence physique : il est recalculé à chaque utilisation de la vue qui se comporte donc comme un select.
- **Sous certaines conditions, on peut ajouter, modifier ou supprimer des tuples dans une vue** et par conséquent dans la ou les tables associées à la vue.

Gestion des vues

1 : CREATE VIEW

La syntaxe de la création d'une vue est la suivante :

```
CREATE VIEW nom_vue AS
   SELECT …
```

ou

```
CREATE or REPLACE VIEW nom_vue AS
   SELECT …
```

2 : DROP VIEW

La syntaxe de la suppression d'une vue est la suivante :

```
DROP VIEW nom_vue ;
```

3 : SHOW TABLES : lister les vues existantes

```
SHOW TABLES ;
```

Ou bien

```
SELECT table_schema, table_name FROM information_schema.views;
```

4 : SHOW CREATE VIEW : consulter le code d'une vue

```
SHOW CREATE VIEW nom_vue;
```

5 : SELECT : utilisation d'une vue

Une vue s'utilise comme une table :

```
SELECT * fROM nom_vue;
```

Exemple

Création de la vue des départements contenant au moins un ANALYST (c'est un job) :

```
CREATE or REPLACE VIEW deptAvecAnalystes as
   Select distinct nd from emp
   where job = 'ANALYST'
   order by nd;
```

Utilisation de la vue

```
SELECT * fROM deptAvecAnalystes;
```

Les 2 usages des vues

1 : décomposer les requêtes en sous-requêtes

Une vue peut remplacer n'importe quelle table dans un select.
Les vues permettent ainsi de décomposer l'écriture des requêtes complexes.
Le select principal joue le rôle de programme principal. Les vues correspondant à des sous-requêtes qui peuvent prendre la place d'une table ou d'un attribut si la vue ne retourne qu'une ligne.

- ***Exemple***

Tous les employés travaillant dans un département qui contient au moins un 'ANALYST' (c'est un métier).
On commence par créer la vue correspondant à la requête : « tous les départements contenant au moins un ANALYST (cf. exemple précédent) :

```
CREATE or REPLACE VIEW deptAvecAnalyste as
Select distinct nd from emp
where job = 'ANALYST'
order by nd;
```

Ensuite on traite la requête principale en exploitant la nouvelle vue :

```
Select ne, nom from emp
where nd in (select nd from deptAvecAnalyste);
```

ou :

```
Select distinct e.ne, e.nom from emp e, deptAvecAnalyste d
where e.nd =d.nd;
```

2 : Faciliter et sécuriser l'accès aux données par les utilisateurs

On peut créer des vues et ne donner des droits d'accès que sur ces vues : ça favorise la sécurisation et ça facilite l'accès, les vues étant orientées « usage ».
La vue permet aussi de restreindre l'accès aux tuples.
Pour que cet usage soit pertinent, il faut pouvoir appliquer les opérations du DML (INSERT, UPDATE, DELETE) à une vue.

- ***Exemple***

On veut limiter les droits d'accès de toto@localhost à la consultation du n°, du nom et du job des employés des départements 10 et 20 (table emp, BD empdept) :

```
Create view vueEmp as
Select ne, nom, job
From emp where nd in (10, 20);
```

```
Grant SELECT on empdept.vueEmp to toto@localhost;
```

Les vues modifiables

INSERT, UPDATE et REPLACE dans une vue

- ***Vue mono-table***

On peut faire des INSERT, des UPDATE et des REPLACE dans une vue mono-table : ils passeront dans la table correspondante, à condition que toutes les contraintes d'intégrité soient correctement prises en compte.

- ***Vue multi-table***

On peut faire des INSERT, un UPDATE ou un REPLACE dans une vue multi-table à condition que toutes les contraintes d'intégrité de la table correspondante soient correctement prises en compte. Mais on ne peut créer, modifier ou remplacer qu'un seul tuple à la fois.

DELETE dans une vue

- ***Vue mono-table***

On peut faire des DELETE dans une vue mono-table : ils passeront dans la table correspondante, à condition que toutes les contraintes d'intégrité soient correctement prises en compte.

- ***Vue multi-tables***

On peut pas faire de DELETE dans une vue multi-tables.

Opération qui rendent une vue non modifiable

- Group by,
- Les Fonction de groupe : count(), sum(), etc.
- Distinct,
- Union (car il intègre un distinct)
- Les attributs calculés
- Jointure externe
- L'utilisation d'un « rownum » : ORACLE

Exemples

```
UPDATE emp3000
SET deptno=10
WHERE SAL > 3500;       // modification effectuée
```

```
UPDATE emp3000
SET sal=4000            // ça ne passera pas du fait du check
option
WHERE deptno > 10;
```

```
DELETE FROM emp3000
WHERE sal < 3000;      // ça ne passera pas du fait du check
option
```

SQL : CREATION DES DONNEES DE LA BD : DML

1. INSERT Création des tuples : DML

CRUD des tuples : Le DML : Data Manipulation Language

Après avoir créées les tables, c'est-à-dire le schéma de la BD, on va pouvoir y mettre des tuples avec le DML, c'est-à-dire les données à proprement parler.

- INSERT INTO
- UPDATE
- DELETE FROM
- REPLACE : peu utilisé

Création de tuples

```
INSERT INTO NomTable (attribut_1, … , attribut_n )
values (valeur_1, … , valeur_n );
```

Insertion d'un tuple :

```
INSERT INTO dept (ND, nom, ville) values
(10, 'ADMINISTRATION', 'PARIS');
```

ou encore, sans préciser la liste des attributs :

```
INSERT INTO dept values
(20, 'VENTES', 'PARIS');
```

ou encore, en série :

```
INSERT INTO dept (ND, nom, ville) values
(10, 'ADMINISTRATION', 'PARIS'),
(20, 'VENTES', 'PARIS'),
(30, 'ACHATS', 'PARIS');
```

ou encore, en précisant le nom des attributs concernés et avec un calcul à l'intérieur des données :

```
INSERT INTO emp
 (NE, nom, fonction, dateEmb, sal, comm, ND, NEchef) values
 (1, 'DUPOND', 'PDG', to_date ('15-01-94','dd:mm:yy'), 30 000, 0,
10, 1);
```

ou encore, avec un select pour une insertion en série :

On peut insérer dans une table des tuples résultant de l'interrogation de la base de données elle-même.

```
INSERT INTO NomTable   (liste d'attributs )
   SELECT (liste d'attributs ) from …
;
```

Les tuples de la table résultant du select sont créés dans la table de l'insert.

Attention :

La table résultant du select doit avoir le même schéma que celle dans laquelle les tuples sont insérés.

Auto-Incrément

Insertion d'un tuple avec un auto-incrément

Quand on ajoute un tuple avec un auto-incrément, il n'est pas nécessaire de préciser la valeur pour l'attribut.
La syntaxe correspondante est variable selon les SGBD-R

Dernier « id » inséré : last insert id

La notion de « last insert id » est associée à celle d'auto-incrément.
En effet, une fois un tuple inséré avec un auto-incrément, si on veut insérer un autre tuple faisant référence avec une clé étrangère à cet auto-incrément, il faut pouvoir récupérer la valeur insérée avec l'auto-incrément.
Les SGBD-R proposent donc une fonction permettant de récupérer la valeur du dernier « id » inséré (last_insertid() MySQL, nextval ORACLE, @@IDENTITY SQL Server
La syntaxe correspondante est variable selon les SGBD-R.

Syntaxe MySQL de l'auto-incrément

- ***Insertion directe :***

```
INSERT INTO departements (nom, ville)
values ('VENTES', 'PARIS');
```

Dans ce cas, il faut préciser la liste des attributs qu'on veut affecter sans préciser le nom de l'attribut clé, et ne pas mettre de valeur pour l'attribut clé.

- ***En passant une valeur NULL***

En passant une valeur NULL ou '', on fait jouer l'auto-incrément :

```
INSERT INTO departements
values (NULL, 'VENTES', 'PARIS');
```

Ou bien

```
INSERT INTO departements
values ('', 'VENTES', 'PARIS');
```

- ***Gestion des clés étrangères : la fonction last_insert_id()***

La fonction last_insert_id() permet de récupérer le dernier numéro de clé primaire donnée quand on a utilisé un auto-incrément, et uniquement dans ce cas.
C'est utile pour faire un lien sur une clé étrangère :

```
INSERT INTO employes VALUES
(NULL,'SCOTT','ANALYST','1982-12-09',3000,NULL,20,7566);
```

```
INSERT INTO employes VALUES
(NULL,'ADAMS','CLERK','1983-01-12',1100,NULL,20,last_insert_id());
```

Le chef de ADAMS sera SCOTT.
On récupère l'id de SCOTT avec la fonction last_insert_id()
A noter que la fonction « last_insert_id() » est gérée au niveau de chaque client.

2. UPDATE - Modification des tuples

Principes

On veut modifier les données d'un ou plusieurs tuple.
Pour cela, il faut

- Récupérer les tuples qui nous intéressent : c'est l'équivalent d'un SELECT.
- Faire la modification des valeurs : par un SET

Exemple : augmenter tous les Manager de 10%

Récupérer les managers :

```
SELECT *
FROM emp
WHERE job = 'Manager';
```

La modification : UPDATE :

```
UPDATE emp
SET salaire = salaire * 1,1
WHERE job = 'Manager';
```

- Le SELECT disparaît
- On remplace le FROM par UPDATE
- On ajoute le SET avant la clause WHERE

Méthode

1) Commencer par le SELECT des tuples concernés : pour voir d'où on part
2) Faire l'UPDATE
3) Refaire un SELECT des tuples concernés : pour voir à quel résultat on arrive

Exemple

1)

```
SELECT *
FROM emp
WHERE job = 'Manager';
```

2)

```
UPDATE emp
SET salaire = salaire * 1,1
WHERE job = 'Manager';
```

3)

```
SELECT *
FROM emp
WHERE job = 'Manager';
```

Syntaxe générale

UPDATE *NomTable*

SET *attribut_1* = *expression* ou (SELECT …)

WHERE *condition* ;

Remarque

Le SELECT dans le SET doit retourner une valeur unique
Le SELECT dans le SET ne doit pas faire référence à la table de l'update.

Modification de plusieurs attributs à la fois

La syntaxe générale est la suivante :

```
UPDATE NomTable
SET     attribut_1 = expression  ou ( SELECT …),
…
attribut_n = expression_n  ou ( SELECT …)
WHERE condition;
```

Exemple

```
UPDATE emp
SET salaire = salaire * 1,1,
            comm = comm +100
WHERE job = ‘Salesman;
```

Modification de toute la colonne

Attention : S'il n'y a pas de WHERE, c'est tous les tuples qui sont concernés.
Il n'y a pas de possibilité de retour en arrière (le **ROLLBACK** ne sert que dans une transaction). La seule façon de retrouver des tuples détruits par erreur sera de revenir à la dernière sauvegarde ce qui risque d'être pénalisant ou de passer par les fichiers de journaux.

```
UPDATE NomTable
SET attribut = expression ;
```

UPDATE et jointure

Ecrire un UPDATE, c'est récupérer les tuples concernés puis faire la modification.
La récupération peut nécessiter une jointure.

Exemple : augmenter les employés parisiens de 10%.

1)

```
SELECT e.*, d.*
FROM emp e, dept d
WHERE e.ND = d.ND
AND ville = 'PARIS;
```

2)

```
UPDATE emp e, dept d
SET salaire = salaire * 1,1
WHERE e.ND = d.ND
AND ville = 'PARIS;
```

3. DELETE - Suppression des tuples

Principes

On veut supprimer un ou plusieurs tuples.
Pour cela, il faut

- Récupérer les tuples qui nous intéressent : c'est l'équivalent d'un SELECT.
- Faire la suppression des valeurs : DELETE

Exemple : supprimer le département 40

Récupérer le département 40:

```
SELECT *
FROM dept
WHERE nd = 40;
```

La suppression : DELETE:

```
DELETE
FROM dept
WHERE nd = 40;
```

- Le SELECT est remplacé par un DELETE
- Il n'y a pas d'attributs projeté

Méthode

1) Commencer par le SELECT des tuples concernés : pour voir d'où on part
2) Faire le DELETE
3) Refaire un SELECT des tuples concernés : pour voir à quel résultat on arrive

Exemple

1)

```
SELECT *
FROM dept
WHERE nd = 40;
```

2)

```
DELETE
FROM dept
WHERE nd = 40;
```

3)

```
SELECT *
FROM dept
WHERE nd = 40;
```

Syntaxe générale

```
DELETE FROM NomTable
WHERE condition ;
```

Suppression de tous les tuples

Attention : S'il n'y a pas de WHERE, c'est tous les tuples qui sont supprimés : la table est vidée !!!

Il n'y a pas de possibilité de retour en arrière (le **ROLLBACK** ne sert que dans une transaction). La seule façon de retrouver des tuples détruits par erreur sera de revenir à la dernière sauvegarde ce qui risque d'être pénalisant ou de passer par les fichiers de journaux.

```
DELETE
FROM dept ;
```

DELETE et jointure

Ecrire un DELETE, c'est récupérer les tuples concernés puis faire la modification.
La récupération peut nécessiter une jointure.
Dans ce cas, **on précise sur la ligne du DELETE les tables concernées par le DELETE**

Exemple 1 : supprimer tous les employés qui travaillent à DALLAS

1)

```
SELECT e.ne, e.nom, d.nd, d.ville
FROM emp e, dept d
WHERE e.ND = d.ND
AND ville = 'DALLAS;
```

2)

```
DELETE e
FROM emp e, dept d
WHERE e.ND = d.ND
AND ville = 'DALLAS;
```

Ca va boguer : on a un problème de clés étrangère.
On peut régler le problème en arrêtant le contrôle des clés étrangères : à pratiquer avec prudence !

2 bis)

```
SET FOREIGN_KEY_CHECKS = OFF;
DELETE e
FROM emp e, dept d
WHERE e.ND = d.ND
AND ville = 'DALLAS;
SET FOREIGN_KEY_CHECKS = ON;
```

Exemple 2 : supprimer tous les employés qui travaillent à CHICAGO et le ou les départements à CHICAGO – version SQL 2

1)

```
SELECT e.ne, e.nom, d.nd, d.ville
FROM emp e
JOIN dept d ON e.ND = d.ND
WHERE ville = ‘CHICAGO;
```

2)

```
DELETE e, d
FROM emp e
JOIN dept d ON e.ND = d.ND
WHERE ville = ‘CHICAGO;
```

2 bis)

```
SET FOREIGN_KEY_CHECKS = OFF;
DELETE e, d
FROM emp e
JOIN dept d ON e.ND = d.ND
WHERE ville = ‘CHICAGO;
SET FOREIGN_KEY_CHECKS = ON;
```

4. REPLACE : Remplacement des tuples (pas standard)

Principe

Remplacer un tuple est équivalent à le supprimer (DELETE) et le créer à nouveau (INSERT INTO).
C'est une manipulation complexe et son usage est limité.

Syntaxe

La syntaxe du REPLACE mixe celle du DELETE (présence du FROM) et de l'UPDATE (présence du SET), mais sans clause WHERE.
A noter qu'il y aura en général plusieurs attribut settés.

```
REPLACE
FROM NomTable
SET      attribut_1 = expression  ou (SELECT …),
…
attribut_n = expression_n  ou (SELECT …) ;
```

4 cas de figure

Si la clé primaire est dans le SET

1. Si la valeur de la clé primaire du SET correspond à un tuple qui existe déjà, alors c'est comme s'il y avait un **DELETE** du tuple puis un **INSERT** d'un nouveau tuple avec les valeurs précisées dans le SET. Les attributs non spécifiés dans le SET passent à NULL.
2. Si la valeur de la clé primaire du SET n'existe pas déjà, alors il y création d'un nouveau tuple, **INSERT**, avec les valeurs précisées dans le SET. Les attributs non spécifiés dans le SET sont à NULL.

Si la clé primaire n'est pas dans le SET

3. Si la clé primaire n'est pas dans le SET et qu'il y a un auto-incrément, alors il y création d'un nouveau tuple, **INSERT**, avec les valeurs précisées dans le SET. Les attributs non spécifiés dans le SET sont à NULL.
4. Si la clé primaire n'est pas dans le SET et qu'il n'y a pas d'auto-incrément, alors la commande génère une **ERREUR**.

5. Import /Export / Sauvegarde : Interactions avec le SE

Présentation

- Le SGBD fonctionne de telle façon qu'il permet de s'abstraire quasiment complètement du système d'exploitation.
- Toutefois, il y a quelques éléments d'interactions avec le SE.
- Concernant le DDL et le DML les interactions sont :
    - La sauvegarde de la base par création d'un code SQL permettant de recréer la base
    - L'importation de données à partir de fichier
    - L'exportation de données dans des fichiers

Import-Export de données

- Les SGBD-R permettent toujours exporter des données au format « Excel » ou équivalent (CSV) et réciproquement d'importer des données à partir d'un format « Excel » ou équivalent (CSV).
- La syntaxe est plus ou moins spécifique selon les SGBD.
- On trouvera des exemples dans le chapitre sur les détails syntaxiques par SGBD.

Exemple MySQL : LOAD DATA INFILE

La commande LOAD DATA ne permet pas de créer la table. Elle ne permet que de charger des données à partir d'une table existante. On l'abordera avec le chapitre sur l'INSERT.

Exemple MySQL : SELECT … INTO OUFILE

La commande SELECT associée à un « INTO OUTFILE » permet d'exporter les tuples d'une table dans un fichier.

Sauvegardes : « dump » de la BD

- Sauvegarde de la BD = « dump » de la BD.

- Les SGBD-R offre toujours des solutions de sauvegarde de la BD : sauvegarde de la structure (le DDL) et/ou sauvegarde des données (le DML).

- Ces solutions prennent deux formes :
  - Soit elles consistent à faire des **copies des fichiers de la BD**. On peut par exemple copier le datadir en entier. Mais c'est une solution à éviter. Un administrateur utilisera les outils proposés par le serveur pour gérer les sauvegardes.
  - Soit elles consistent à enregistrer les commandes SQL qui permettent la restauration d'une BD ou d'une partie de la BD. La **commande « mysqldump » de MySQL** est de ce type.

- Les outils de sauvegarde sont différents selon les SGBD.

Importation de fichier Excel avec phpMyAdmin

Présentation

- Avec phpMyADMIN, on peut importer des données issues de fichiers Excel et ainsi créer une table et mettre des tuples dedans.
- Il faut commencer par enregistrer les données Excel au format CSV.
- Ensuite on peut importer le fichier via phpMyAdmin : onglet « Importer »
- Il faut faire attention à bien paramétrer les séparateurs d'attribut et la présence d'une première ligne de description des attributs

Résultats

- L'importation va créer une table et insérer des tuples.
- Les attributs n'auront ni type ni contraintes d'intégrité.
- Pour ajouter le type et les contraintes d'intégrité, il faudra faire des ALTER TABLE ou faire les modifications directement avec phpMyAdmin.

COMPLEMENTS SPECIFIQUES

MySQL : calculette mysql.exe

Connexion

```
C : > mysql –uroot -p
```

Les fonctions de la calculette

http://dev.mysql.com/doc/refman/5.0/fr/entering-queries.html

Select database();

Select user() ;

Select version() ;

Select found_rows( ), row_count( )

La commande SHOW

http://dev.mysql.com/doc/refman/5.0/fr/show.html

help show

Show databases, Show tables, Show create table, Show grants, Show variables

Show table status from nomBD ; // synthèse sur les tables de la BD

Select table_schema, table_name, table_row from tables where table_schema='maBD';

Présentation des résultats : limit et \G

Limit 5 ; // les 5 premiers
Limit 5, 10 // 10 tuples à partir du 5+1ème
\G // affichage en mode page.

MySQL : WorkBench

Installation de MySQL Workbench

MySQL Workbenc est un environnement de développement graphique proposé par MySQL.

Aller sur le site : http://dev.mysql.com/download

http://dev.mysql.com/downloads/gui-tools/8.0.html
Télécharger la version Windows (x86) : c'est un .msi (autrement dit, un installateur).

Installation

Lancer le fichier .msi téléchargé.
Sur cette page : http://bliaudet.free.fr/article.php3?id_article=307, on trouve quelques explications de post-installation concernant Workbench dans la page et aussi dans le document « 02_Installation_de_MySQL.pdf » qui se trouve en bas de page, et qu'on trouve directement ici : http://bliaudet.free.fr/IMG/pdf/02_Installation_de_MySQL.pdf

MySQL : les types

documetation

http://dev.mysql.com/doc/refman/8.0/en/column-types.html

Les types numériques

http://dev.mysql.com/doc/refman/8.0/en/numeric-type-overview.html

BOOL, BOOLEAN, BIT

INT (M), INTEGER (M), TINYINT (M), SMALLINT (M), MEDIUMINT (M), BIGINT (M)

FLOAT(precision), FLOAT (M,D), DOUBLE (M,D), DOUBLE PRECISION (M,D), REAL (M,D)

DECIMAL (M,D), DEC (M,D)

avec M : taille de l'affichage, D: nombre de décimal, M>=D, si M=D alors –1 < n < 1.

Les types date

http://dev.mysql.com/doc/refman/8.0/en/date-and-time-type-overview.html

DATE, DATETIME, TIME

TIMESTAMP : pour gérer automatiquement la date de création ou de mise à jour.

YEAR(2|4)

Type chaîne

http://dev.mysql.com/doc/refman/8.0/en/string-type-overview.html

CHAR(M), NCHAR(M), CHARACTER : taille fixe, 255 max.

VARCHAR(M), NATIONAL VARCHAR(M) : taille variable, 2000 max.

BLOB, TEXT, TINYBLOB, TINYTEXT, MEDIUMBLOB, MEDIUMTEXT, LONGBLOB, LONGTEXT : taille dynamique, jusqu'à 4GO pour les plus grands.

Type collection : enum et set

ENUM('value1','value2',...) : une valeur parmi une liste. Les valeurs proposées seront les seules possibles. L'enum se comporte donc comme un « check ».

SET('value1','value2',...) : zéro, une ou plusieurs valeurs parmi une liste (violation de la première forme normale !).

MySQL : import, export, sauvegarde

mysqldump

http://dev.mysql.com/doc/refman/5.0/fr/mysqldump.html
mysqldump est une commande qui permet de produire le code SQL permettant de recréer entièrement la BD.

Trois usages de mysqldump

- ***Sauvegarder des BD***

```
C:\ mysqldump [options] --databases DB1 [DB2 DB3...]
```

- ***Sauvegarder toutes les BD***

```
C:\ mysqldump [options] --all-databases
```

Affichage à l'écran ou redirection dans un fichier

```
C:\ mysqldump -uroot -p nomBD
```

La commande affiche le code SQL de la BD nomBD.

```
C:\ mysqldump -uroot -p nomBD > nomFichier
```

La commande écrit le code SQL de la BD nomBD dans le fichier nomFichier.

Usage courant de MYSQLDUMP pour sauvegarder et recharger une BD

Commande usuelle de sauvegarde d'une BD :

```
C:\ mysqldump --opt nomBD > backupFile.sql
```

L'option --opt inclut l'option --quick et l'option --lock-tables, entre autres.

Pour recharger le fichier de sauvegarde :

```
C:\ mysql nomBD < backupFile.sql
```

Usage courant de MYSQLDUMP pour sauvegarder toutes les BDs

Commande usuelle de sauvegarde de toutes les BD :

```
C:\ mysqldump --opt --all-databases > allBD.sql
```

Pour recharger le fichier de sauvegarde de toutes les BD :

```
C:\ mysql < allBDql
```

Usage spécial de MYSQLDUMP : ne sauvegarder que le schéma de la BD

-- no-data

Exporter les données : SELECT … INTO OUTFILE

A partir de mysql

Production d'un fichier « csv » exploitable sous Excel.

- ***Chemin absolu***

```
mysql> SELECT * INTO OUTFILE 'c:/dept.csv'
       fields terminated by ','
       FROM dept;
```

- ***Chemin relatif : DATADIR***

```
mysql> SELECT * INTO OUTFILE './dept.csv'
       fields terminated by ','
       FROM dept;
```

- ***Chemin relatif : DATADIR / BDused***

```
mysql> SELECT * INTO OUTFILE 'dept.csv'
       fields terminated by ','
       FROM dept;
```

A partir du SE

```
C: \ > mysql dbname –u user –p >output.tab
```

On passe en mode batch, l'affichage est redirigé dans le fichier output.tab

- ***Ou bien***

```
C: \ > mysql dbname –u user –p –pager=cat>output.tab
```

Remarque : output.tab est écrasé au début de l'opération, mais toutes les commandes de la même session sont executées à la suite

- ***Autre usage :***

```
C: \ > mysql dbname –u user –p < fichier.sql > output.tab
```

Importer les données : LOAD DATA INFILE

http://dev.mysql.com/doc/refman/8.0/en/load-data.html

On peut importer des données en précisant le format de séparation des champs.
Par exemple: les fichiers “csv” produit par Excel utilisent la virgule comme séparateur.

Attention :

La table correspondant au fichier importé doit avoir le même schéma que celle dans laquelle les tuples sont insérés.

Chemin absolu

```
mysql> LOAD DATA INFILE 'c:/dept.csv' INTO TABLE dept
       fields terminated by ',';
```

Chemin relatif : DATADIR

```
mysql> LOAD DATA INFILE './dept.csv' INTO TABLE dept
       fields terminated by ',';
```

Chemin relatif : DATADIR / BDused

```
mysql> LOAD DATA INFILE 'dept.csv' INTO TABLE dept
       fields terminated by ',';
```

Chemin relatif : DATADIR / BD au choix

```
mysql> LOAD DATA INFILE './empdept/dept.csv' INTO TABLE dept
       fields terminated by ',';
```

Avec les options REPLACE ou IGNORE, quand une donnée rentre avec une clé primaire existant déjà, elle remplacera l’ancienne ou elle sera ignorée. Sinon, l’opération s’arrête en erreur.

```
LOAD DATA INFILE 'nomFichier' REPLACE INTO TABLE nomTableOUTFILE
```

MySQL : ENGINE - Moteur MyISAM et moteur InnoDB

Notion de « moteur »

MySQL offre plusieurs « moteurs » pour gérer les tables. Les deux principaux sont : MyISAM et InnoDB.
Un moteur peut être vu comme un SGBD particulier pour gérer les tables.

Moteur d'une table

Par défaut

Quand on fait un CREATE TABLE, le moteur associé est celui par défaut.

Explicitement

A la création de la table, on peut préciser le moteur :

```
CREATE TABLE NomTable  (
       …
) ENGINE InnoDB;
```

On peut écrire ENGINE ou TYPE.

Moteur par défaut

default_storage_engine : moteur par défaut

La variable d'environnement « default_storage_engine » définit le moteur par défaut.

```
mysql> show variables like '%engine%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| default_storage_engine   | InnoDB |
+--------------------------+--------+
```

Choix d'un moteur

Le moteur MyISAM est le moteur d'origine. Il est très permissif : il ne gère pas les contraintes d'intégrité référentielle ni les transactions.
Le moteur InnoDB gère l'intégrité référentielle et les transactions. On n'utilisera toujours ce moteur pendant ce cours.
Par défaut, il vaut mieux choisir le moteur InnoDB qui prend en compte la norme SQL.
Pour améliorer les performances, on peut revenir, si nécessaire au moteur MyISAM.
Etant donné que les performances sont, dans un premier temps, une question secondaire, et que par contre, on veut absolument apprendre à gérer finement les contraintes de clés étrangères puis les transactions :

On utilisera toujours le moteur InnoDB !!!

Passage d'un moteur à l'autre

On peut modifier le moteur d'une table :

Passage en MyISAM

```
ALTER TABLE emp ENGINE MyISAM ;
```

Pour passer en d'InnoDB à MyISAM, il ne doit pas y avoir de clé étrangère. Il faut donc commencer par supprimer les contraintes nommées de clé étrangère.

Passage en InnoDB

```
ALTER TABLE emp ENGINE InnoDB ;
```

Le passage de MyISAM a InnoDB est possible directement.

Modification du moteur par défaut

Via le fichier de configuration

On modifie la variable 'default_storage_engine' dans le fichier de configuration (my.ini sur PC).
En redémarrant le serveur, c'est cette nouvelle valeur qui sera prise en compte par tous les clients.

Par modification dynamique des variables du serveur

```
Set @@table_type ='myisam';
```

Est équivalent à :

```
Set @@local.table_type ='myisam';
```

Est équivalent à :

```
Set @@session.table_type ='myisam';
```

Est équivalent à :

```
Set local table_type ='myisam';
```

La modification est prise en compte par le client qui a lancé la commande.
En remplaçant « local » par « global » :

```
Set global table_type ='myisam';
```

Est équivalent à:

```
Set @@global.table_type ='myisam';
```

La modification sera prise en compte par tout nouveau client : elle n'est donc pas prise en compte par le client qui a envoyé la commande.

MySQL : SHOW CREATE TABLE -> voir le code pris en compte par le serveur

SHOW CREATE TABLE

La commande **show create table** *nomTable* permet de montrer le code pris en comte par le serveur pour la création d'une table.

Exemple 1 : mysql> show create table dept;

```
| emp   |
CREATE TABLE `dept` (
  `ND` int(11) NOT NULL auto_increment,
  `NOM` varchar(14) default NULL,
  `VILLE` varchar(13) default NULL,
  PRIMARY KEY  (`ND`)
) ENGINE=InnoDB AUTO_INCREMENT=41 DEFAULT CHARSET=latin1
```

- ***Remarques***
  1. le n° du prochain ID automatique sera 41.
  2. InnoDB est un choix de SGBD. C'est celui par défaut.
  3. Le code est exploitable directement pour créer une table.

Exemple 2 : mysql> show create table emp;

```
| emp   |
CREATE TABLE `emp` (
  `NE` int(11) NOT NULL auto_increment,
  `NOM` varchar(10) default NULL,
  `JOB` varchar(9) default NULL,
  `DATEMB` date default NULL,
  `SAL` float(7,2) default NULL,
  `COMM` float(7,2) default NULL,
  `ND` int(11) NOT NULL,
  `NEchef` int(11) default NULL,
  PRIMARY KEY  (`NE`),
  KEY `ND` (`ND`),
  KEY `NEchef` (`NEchef`),
  CONSTRAINT `emp_ibfk_1` FOREIGN KEY (`ND`) REFERENCES `dept`
(`ND`),
  CONSTRAINT `emp_ibfk_2` FOREIGN KEY (`NEchef`) REFERENCES `emp`
(`NE`)
) ENGINE=InnoDB AUTO_INCREMENT=7944 DEFAULT CHARSET=latin1
```

- ***Remarques***
  1. Le n° du prochain ID automatique sera 7944.
  2. Les clés étrangères sont déclarées en plusieurs étapes : avec KEY et avec CONSTRAINT
  3. Le code est exploitable directement pour créer une table.

MySQL : Dictionnaire

2 bases de données pré-installées, entre autres :

- information_schema
- mysql

Dictionnaire des données

information_schema et mysql sont des bases systèmes qui correspondent à ce qu'on appelle le dictionnaire des données.
Ces deux bases contiennent des méta-données : données sur les données.
Dans la BD mysql, on trouve particulièrement la table « users » qui contient les utilisateurs enregistrés dans la BD.

MySQL : commandes de la calculette

Show databases : lister toutes les databases auxquelles on a accès.
Create database *nomDataBase* : créer une database.
Drop database *nomDataBase* : supprimer une database.
Drop database if exists *nomDataBase* : supprimer une database si elle existe : évite le message d'erreur.
Use *nomDataBase* : utiliser une database.
Select database : affiche le nom de la database utilisée.
Show tables : lister les tables de la database en cours.
Show tables from nom_bd : lister les tables de la database nom_bd.
Desc *nomTable* : lister les attributs d'une table
Show index from *nomTable* : lister les index d'une table
Show create table *nomTable* : afficher le code de création d'une table

MySQL : ISO-ASCII – UTF 8 - Questions de caractères

Généralités sur la problématique des caractères

Présentation

Il s'agit de coder les caractères alphanumériques : les chiffres (0,1,...,9), les lettres minuscules et majuscules (a,...z, A,...,Z), les signes de ponctuation (. , ; / = * - …), des symboles usuels (%, @, $, &, etc.), des caractères spéciaux (passage à la ligne, delete, bip sonore, etc.)
Cet ensemble de caractères contient : 10 chiffres, 52 lettres, une 20aine de signe de ponctuation ou de symboles usuels, ce qui fait déjà plus de 80 caractères.
Avec 6 bits, on peut coder $2^6 = 64$ caractères : c'est insuffisant. Avec 7 bits, on peut coder $2^7 = 128$ caractères. C'est suffisant.
Il existe finalement trois codes : le code ASCII, ASCII étendu ou ISO et l'UNICODE.

- Le code ASCII codifie 128 caractères sur 7 bits.
- Les codes ASCII étendus ou codes ISO 8859 permettent de codifier 256 caractères sur 8 bits. Il en existe de nombreux, ce qui est source de problèmes avec les accents particulièrement.
- L'UNICODE permet de normaliser tous les caractères possibles, actuellement 128 172. Il est indépendant de la représentation physique (le nombre d'octets).

Le code ASCII

- ***Présentation***

Le code ASCII est un code standard qui définit la valeur du jeu courant de caractères sur 7 bits.
Sur un octet, il restera le bit de poids fort.
Les caractères sont donc codés de 0 à 127 :

- 0 à 31 : des caractères spéciaux
- 32 à 47 : des symboles et de la ponctuation : (espace, ! , « , #, etc.)
- 48 à 57 : les chiffres dans l'ordre : (0, … , 9)
- 58 à 64 : des symboles et de la ponctuation : (: , ; , < , etc.)
- 65 à 90 : les alphabets majuscules (A,...,Z)
- 97 à 122 : les alphabets minuscule (a,...z)
- 123 à 127 : des symboles et de la ponctuation : ([, \ , etc.)

- ***Avantages***

Il permet aux machines de communiquer entre elles, quels que soient les processeurs et quels que soit les système d'exploitation.

- ***Défaut***

American Standard Code for Information Interchange" signifie en français "Code américain normalisé pour l'échange d'information".
C'est un code basé sur l'alphabet américain : alphabet latin sans accents.

➢ ***Table ASCII***

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Les codes ASCII étendus : codes ISO 8859

➢ ***Présentation***

L'objectif est d'élargir le code ASCII pour permettre par exemple l'introduction d'accents.
Le 8ème bit de l'octet du caractère va permettre cela. C'est ce qu'on appelle le code ASCII étendu.
C'est le code ISO 8859 qui normalise ça.
ISO signifie International Organization for Standardization (le sigle ne correspond pas !)
Comme il y a beaucoup de système d'accentuation dans les langues latines, et qu'il y a aussi des langues non latines, le code ISO 8859 se décline en :
ISO 8859-1 (ou latin-1 ou européen occidental) : pour la plupart des langues européennes.
ISO 8859-2 (ou latin-2 ou européen central) : pour le le croate, le tchec, le hongrois, etc.
…
ISO 8859-6 : pour l'arabe
…
ISO 8859-15 : révision du 8859-1
…
https://fr.wikipedia.org/wiki/ISO/CEI_8859

➢ ***Avantage***

Intègre le standard ASCII.
Permet de standardiser les accents.

- ***Défaut***

Il y a plusieurs standards ! Et des révisions dans les standards.

L'UNICODE et UTF8

- ***UNICODE***

http://www.tuteurs.ens.fr/faq/utf8.html#unicode
Le Standard Unicode est produit par une organisation à but non lucratif (le Consortium Unicode) ayant pour objectif d'attribuer un numéro à tout caractère utilisé dans une langue humaine.
Un caractère Unicode est un caractère défini dans le Standard Unicode. On y fait référence par son numéro écrit en hexadécimal précédé de «U+». Par exemple, la lettre latine « a » correspond à U+0061.
Unicode couvre actuellement 128 172 caractères.
http://www.unicode.org/charts/

- ***UTF 8***

L'UTF8 est le système d'encodage qui transforme une code Unicode en octets : de 1 à 4 octets.
L'UTF8 est compatible avec les caractères ASCII. Particulièrement, les caractères de 32 à 127 seront codés sur 1 octet.

- ***Avantage***

Un seul encodage pour toutes les langues : c'est LE système universel.

- ***Inconvénients***

La taille des caractères n'est pas fixe, ce qui complique certains calculs.

- ***Usages***

Généralisé ! Dans les sites web, sur les terminaux d'ordinateur, etc.

Aparté technique

L'encodage des terminaux windows fonctionne avec des « page de code ». La page de code n°850 est celle d'Europe occidentale. On peut voir le n° avec la commande chcp.
https://fr.wikipedia.org/wiki/Page_de_code
Sous linux, la commande xxd permet de voir l'encodage d'un texte : xxd-entrée-texte-entrée-ctrl-d. Sont affichés les caractères du texte en hexadécimal avec un 0a à la fin pour le line feed. Si les caractères accentués sont sur 2 octets, c'est que l'encodage est en UTF8.

MySQL : Jeu de caractères -> character set

Définition d'un jeu de caractères

Ensemble de caractères (lettres, ponctuation, chiffres, etc.) auxquels sont associés un numéro de code.
Exemple : code ASCII. « A »=65, « , » = 44. 128 caractères sont codés de façon standard.
Select ASCII ('A');
Select CHAR(65);

Codages ISO (origine ASCII)

Principe : 1 octet = 8 bit = 256 codages possibles.
Les jeux de caractères normalisés ISO 8859 sont codés sur 1 octet.
Le ISO 8859-1 est le « latin-1 » ou « occidental » ou « western »
Le ISO 8859-6 fournit les caractères arabes.

Codages Unicode : UTF 8

Principe : un jeu de caractères pour toutes les langues : plus que 256 caractères.
Les jeux de caractères Unicode utilisent plusieurs octets.
UTF-8 utilise des caractères de taille variable, de 1 à 4 octets.
Attention, l'UTF 8 gère mal les accents.

Police de caractère : character font

La police est une notion différente du jeu de caractères. Elle ne concerne que l'apparence graphique.

Character set dans Mysql

- ***Variables d'environnement***

Show character set; // jeux de caractères disponibles.
Select * from information_schema.character_sets; // état du dictionnaire des données
Show variables like "%char%"; // état des variables d'environnement
Set @@character_set_client="latin1"; // modification des variables

- ***Signification des variables***

character_set_client : jeu de caractère utilisé en saisie
character_set_connection : jeu de caractères utilisé pour la communication client serveur
character_set_results : jeu de caractères utilisé pour l'affichage des résultats
character_set_database : jeu de caractères de la BD en cours d'utilisation
character_set_server : jeu de caractère du serveur

- ***My.ini***

Show variable like "datadir" ; // répertoire du fichier my.ini
[mysql]
default-character-set=latin1
[mysqld]
default-character-set=latin1

MySQL : Collation

Définition d'une collation

Une collation est liée à un jeu de caractères.
Elle donne l'ordre de classement des caractères et l'équivalence éventuelle entre caractère (« e » équivalent à « E », « é », « è »)

Type de collation en fonction du classement

_bin : dans l'ordre, majuscules d'abord, minuscules ensuite, accentuées après, etc.
_cs : « case sensitive »
_ci : « case insensitive »

Collation par défaut

latin1_swedish_ci : collation par défaut de latin1, bien adaptée au français.
latin1_german_ci : collation sensée (?) être mieux adaptée au français.
Autres collations : latin1_bin, latin1_general_cs, latin1_general_ci, utf8_bin, utf8_general_ci, utf8_unicode_ci.

Collation dans Mysql

Show collation like « latin1% » ; // collations associées au latin1.
Select * from information_schema.collations; // état du dictionnaire des données
Show variables like "%coll%"; // état des variables d'environnement
Set @@collation_database="latin1_bin";

- ***My.ini***

[mysql]
default-collation=latin1_german_1
[mysqld]
default-collation=latin1_german1_ci

MySQL : 5 niveaux de définition -> Serveur, Client, Base, Table, Colonne

Serveur et client

my.ini
show variable like …
select @@global..character_set_database // niveau serveur
select @@local..character_set_database // niveau client

Base

CREATE DATABASE nomDatabase CHARSET latin1 COLLATE latin1_german1_ci;
Show create database nomDataBase;
Show variable like “character_set_database”; ⇔ Select @@character_set_database;
Show variables like “collation_database”; ⇔ Select @@collation_database;
ALTER DATABASE nomDataBase CHARSET latin1 COLLATE latin1_bin;
// attention: l’alter database modifie les variable @@character_set_database et @@collation_database. Le contraire n’est pas vrai.

Table

CREATE TABLE nomTable { … } CHARSET latin1 COLLATE latin1_bin; // valeurs par défaut
ALTER TABLE nomTable CHARSET latin1 COLLATE latin1_bin; // valeurs par défaut

Colonnes

CREATE TABLE nomTable { …, att1 varchar(50) CHARSET latin1 COLLATE latin1_bin, …}; // valeur par défaut de la colonne
ALTER TABLE nomTable ADD COLUMN attX varchar(50) CHARSET latin1 COLLATE latin1_bin; // valeur par défaut de la colonne
SHOW FULL COLUMNS FROM nomTable ;
SELECT DISTINCT CHARSET(attribute) FROM nomTable;

MySQL : Conversion -> CONVERT

La conversion pose un problème quand un caractère du jeu de départ n’a pas d’équivalent dans le jeu d’arrivée.
Une colonne est convertie quand on change son jeu ou sa collation.
Pour convertir une table il faut utiliser la fonction CONVERT

Conversion niveau colonne

ALTER TABLE nomTable MODIFY attX varchar(50) CHARSET latin1 COLLATE latin1_bin; // conversion de la colonne

Conversion niveau table

ALTER TABLE nomTable CONVERT TO CHARSET latin1 COLLATE latin1_bin ; // conversion de toutes les colonnes et des valeurs par défaut de la table.

Conversion dans les SELECT

- ***Forcer la sensibilité à la casse : binary, collate***

SELECT * from table WHERE BINARY attribut =”TOTO”;
SELECT * from table WHERE attribute COLLATE latin1_bin=”TOTO”;

Les “toto” en minuscules ne rentreront pas dans le resultats.

- ***Changer de jeu de caractères***

SELECT … WHERE CONVERT(attribut USING utf8)

- ***Changer de jeu de caractères et de collation***

SELECT … WHERE CONVERT(attribut USING utf8) collate utf8_Unicode_ci

Usages

UNICODE : niveau serveur (jeu de caractères le plus large : exemple : utf8)
ISO niveau client (jeu de caractères plus restreint et mieux adapté : exemple : latin1)
En latin1, la collation par défaut est : latin1_swedish_ci. Il semblerait que la collation : latin1_german1_ci soit plus adapté au français.
En théorie, l’utf8 n’est sensible ni à la casse, ni aux accents. Pour pouvoir regrouper des mots distingués uniquement par leurs accents, on peut passer l’attribut en utf8, soit au niveau du select, soit au niveau de l’attribut de la table. En pratique, c’est moins convaincant…

MySQL : Consultation des données sur le disque

Selon le moteur choisi, MySQL enregistre différemment les données sur le disque.

Moteur MyISAM

Répertoire de stockage : le « datadir »

La variable « datadir » précise quel est le répertoire de stockage des données.
La commande :

```
Mysql > show variables like 'datadir';
```

ou

```
Mysql > Show variable like '%dir%';
```

Permet de consulter la valeur du « datadir » ;

Organisation des données

Chaque BD est rangée dans le DATADIR et dans un répertoire du nom de la BD.
Quand on supprime la BD (drop database), c'est tout le répertoire qui est supprimé.

Chaque table d'une BD MyISAM est stockée sur disque dans trois fichiers.
Les fichiers portent le nom de la table et ont une extension qui spécifie le type de fichier.
Il faut éviter de manipuler directement ces fichiers.

Fichier de description de la BD : db.opt

La commande :

```
Create database nomBase ;
```

Crée le répertoire « *nomBase* » dans le datadir et un fichier « db.opt » dans ce répertoire.
Ce fichier contient les caractéristiques de la base. En effet, le create database peut être paramétré.

Fichier de description des tables : nomTable.FRM

La commande :

```
Create table nomTable (…);
```

Crée dans un fichier .frm qui décrit la structure de la table.
La commande crée aussi deux fichiers pour les données : un fichier « .myd » pour les tuples et un fichiers « .myi » pour les index.
Au départ, ces fichiers sont vides.

Fichiers de données et d'index : nomTable.MYD et nomTable.MYI

La commande

```
Insert into table nomTable values (…);
```

Remplit les fichiers MYD et MYI correspondants à la table.

Moteur InnoDB

Répertoire de stockage : « InnoDB_data_home_dir » ou « datadir »

La commande :

```
Mysql> Show variable like 'InnoDB_data_home_dir';
```

ou

```
Mysql> Show variable like '%dir%';
```

Permet de consulter la valeur du « InnoDB_data_home_dir ».
Si la valeur n'est pas renseignée, le répertoire de stockage correspond à la valeur du DATADIR.
Le répertoire de InnoDB_data_home_dir correspond au répertoire des données et des index.

Fichier de description de la BD et des tables : db.opt et nomTable.frm

Ce sont les même qu'avec le moteur MyISAM

Fichiers de données et d'index : ibdata1

La commande :

```
Insert into table nomTable values (…);
```

Met à jour le fichier « ibdata1 » du répertoire « InnoDB_data_home_dir ».

Fichier de log : ib_logfile0

Un fichier de log est aussi mis à jour.

PostgreSQL

Fonctionnement de la calculette

Organisation logique arborescente

Une database contient un ou plusieurs schémas qui contiennent une ou plusieurs tables et autres objets d'une base de données : des vues, des procédures stockées, des triggers, des index, des sequences, etc.

Databases

Schémas

Table et autres objets

Organisation logique des fichiers

Une table est enregistrée dans un « tablespace ».
Un « tablespace » contient plusieurs tables.
Les tables d'un même schéma peuvent être enregistrées dans des « tablespace » différents. Il en est de même pour tous les objets de la BD.

Organisation physique des fichiers

Un fichier correspond à un « tablespace » et un seul.
Un « tablespace » peut être réparti sur plusieurs fichiers.

Niveau Database

Afficher les database

```
Postgres=# \l
```

Connexion à une database

```
Postgres=# \c nomDatabase -- commande psql
```

Create database

```
CREATE DATABASE nomDatabase();
```

Drop database

```
DROP DATABASE nomDatabase();
```

Niveau Schéma

Afficher les schémas de la database

```
Postgres=# \dn
```

Selection d'un ou plusieurs schémas

```
Postgres=# set search_path = nomSchema
```

Afficher le ou les schémas sélectionnés

```
Postgres=# show search_path
```

Create schema

```
CREATE SHEMA nomSchema;
```

Drop schema

```
DROP SHEMA nomSchema;
```

Niveau table et objets de la BD

Afficher les tables des schémas

```
Postgres=# \d
```

Afficher les attributs d'une table

```
Postgres=# \dn nomTable
```

Les types

Les numériques

A chaque type, correspond un autre nom avec le nombre d'octets
smallint, int2
integer, int, int4
bigint, int8
serial, serial4
bigserial, bigserial8
real, float4
double precision, float8
numeric[(p,s)], decimal[(p,s)], avec p, nombre total de chiffres, s, nombre de décimales

Les caractères

char [(n)], character[(n)] : longueur fixe
varchar[(n)], character varying[(n)] : longueur variable
text : longueur illimitée

Booléen

boolean, bool

Date

date : de –beaucoup avant JC à + beaucoup après JC !
time : jusqu'à la microseconde
timestamp : date et heure, jusqu'à la microseconde.
interval : interval de temps, entre – et + 178 millions d'année, précision jusqu'à la microseconde.

Format des dates

La variable « datastyle »

show datestyle : SQL, DMY
« Datestyle » prend deux valeurs parmi (ISO, SQL, POSTGRES, GERMAN) et (DMY, MDY)

Mise à jour de la variable « datastyle »

SET datestyle TO valeur

Conversion d'un texte en date

La conversion de texte en date et de date en texte est automatique.

Import-export

Exportation format Excel

COPY nomtable TO 'C:/…/maTable.csv' WITH CSV;
Ou
COPY (select …) TO 'C:/…/maTable.csv' WITH CSV;

Importation format Excel

COPY nomtable FROM 'C:/…/maTable.csv' WITH CSV;
La table doit déjà exister.

http://www.postgresql.org/docs/8.3/static/sql-copy.html

Exemple de code PostgreSQL

```
\!cls

drop table if exists etudiants cascade;
drop table if exists examens cascade;
drop table if exists groupes cascade;
drop table if exists epreuves cascade;
drop table if exists evaluer cascade;
drop table if exists participer cascade;

------------------------------------------------------------
CREATE TABLE etudiants (
  NET        int PRIMARY KEY,
  nom        varchar(50),
  prenom     varchar(50),
  email      varchar(100)
);

------------------------------------------------------------
CREATE TABLE examens (
  NEX              int PRIMARY KEY,
  matiere    varchar(50) default '',
  professeur       varchar(50),
  session    varchar(12)
             check (session in('juin','septembre')),
  annee            numeric(4)
             check (annee>2000 and annee<2050),
  niveau           int2,
  type             varchar(12) check (type in('ecrit','oral')),
  duree            float
);
```

```
-------------------------------------------------------------
CREATE TABLE groupes (
  NGR        int PRIMARY KEY,
  promo      int,
  annee      int2,
  groupe     varchar(10),
  UNIQUE (Promo, annee, groupe)
);

-------------------------------------------------------------
CREATE TABLE epreuves (
  NEP        int PRIMARY KEY,
  salle           int2,
  dateHeure       date,
  NEX             int not NULL,
             FOREIGN KEY (NEX) REFERENCES examens (NEX),
  NGR             int not NULL,
             FOREIGN KEY (NGR) REFERENCES groupes (NGR),
  UNIQUE (NGR, dateHeure)
);

-------------------------------------------------------------
-- Table evaluer
CREATE TABLE evaluer(
  NET        int,FOREIGN KEY (NET) REFERENCES etudiants (NET),
  NEP        int,FOREIGN KEY (NEP) REFERENCES epreuves (NEP),
  note       int not NULL,
  PRIMARY KEY     (NET,NEP)
);

-------------------------------------------------------------
CREATE TABLE participer (
  NGR        int,
  NET        int,
  CONSTRAINT participer_ngr_fk FOREIGN KEY (NGR) REFERENCES
groupes (NGR),
  CONSTRAINT participer_net_pk FOREIGN KEY (NET) REFERENCES
etudiants (NET),
  PRIMARY KEY    (NGR,NET)
);


-------------------------------------------------------------
-- Tuples etudiants

INSERT INTO etudiants VALUES
(1,'Dupont','Jean','jeandupont@yahoo.fr'),(2,'Durand','Marie','ma
ried@hotmail.com'),
(3,'Dupont','Pierre','dupontp@caramail.com'),(4,'David','Marc','m
arcdavid@yahoo.com'),
(5,'Dupuis','Vanessa','vaness632@caramail.com'),(6,'Carlier','Ste
phane','steph@carlier.com'),
(7,'Merlot','Stephane','merlot@yahoo.fr'),(8,'Chenu','Caroline','
carochenu@laposte.net'),
(9,'Michelin','Baptiste','bap@mymail.com'),(10,'Nerval','Marie-
Ange','man@email.com'),
```

```
(11,'Janset','Nicole','nicolejanset@yahoo.fr'),(12,'Brulard','Nic
olas','nico@brulard.com'),
(13,'Jordan','Jacques','jak@myemail.com'),(14,'Chamblard','Philip
pe',NULL),
(15,'d''Estienne','Paul','paulot@today.com'),(16,'Decroix','Johan
n','johann2x@yahoo.com'),
(17,'Lutece','Stephanie',NULL),(18,'Tourette','Carmen','carmenita
@yahoo.fr'),
(19,'Leonin','Jean-
Sebastien','jsl@mydomain.com'),(20,'Vassal','Helene','helene_vass
al@hotmail.com'),
(21,'Dumont','Xavier',NULL),(22,'Williot','Claire','claire_w@cara
mail.com'),(23,'Cremant','Gregory','greg@cremant.fr'),
(24,'Mauroit','Sandra',NULL),(25,'Guyard','Bertrand','tranber18@c
aramail.com'),
(26,'Dupret','Jean-
Francois','jfdupret@hotmail.com'),(27,'Allard','Severine','sevall
ard@hotmail.com'),
(28,'Chetty','Stanislas','stan@chetty.com'),(29,'Laury','Etienne'
,'etiennelaury@yahoo.com'),
(30,'Mironton','Claude','claudemironton@yahoo.fr'),(31,'Ghilain',
'Sophie',NULL),
(32,'Genty','Aurore','auroregenty@laposte.net'),(33,'Thibault','J
eremy','jerem45@yahoo.fr'),
(34,'Lorrain','Michel','mlorrain@aol.com'),(35,'Parclos','Celine'
,'celineparclos@hotmail.com'),
(36,'Cadet','Alexandre',NULL);

-------------------------------------------------------------
-- Tuples examens

INSERT INTO examens VALUES
(1,'mathematiques','Martin','juin',2006,1, 'ecrit',3),
(2,'mathematiques','Martin','septembre',2006,1,'oral',1),
(3,'anglais','Jones','juin',2006,1,'ecrit',2),
(4,'anglais','Jones','septembre',2006,1,'oral',0.5),
(5,'culture generale','Dupuis','juin',2006,1,'ecrit',1),
(6,'culture generale','Dupuis','septembre',2006,1,'oral',0.5),
(7,'mathematiques','Martin','juin',2006,2,'ecrit',1),
(8,'mathematiques','Martin','septembre',2006,2,'ecrit',1),
(9,'mathematiques','Martin','juin',2006,3,'ecrit',1),
(10,'mathematiques','Martin','septembre',2006,3,'ecrit',1),
(11,'anglais','Jones','juin',2006,2,'ecrit',1),
(12,'anglais','Jones','septembre',2006,2,'ecrit',1),
(13,'anglais','Jones','juin',2006,3,'ecrit',1),
(14,'anglais','Jones','septembre',2008,3,'ecrit',1);


-------------------------------------------------------------
-- Tuples groupes

INSERT INTO groupes VALUES
(1, 2011, 1, 'A'),
(2, 2011, 1, 'B'),
(3,2010, 2, 'SIGL'),
(4,2010, 2, 'SRT'),
(5, 2009, 3,'SIGL'),
(6, 2009, 3,'SRT'),
```

```
(7,2010, 1, 'A'),
(8,2010, 1, 'B'),
(9, 2009, 1,'A'),
(10, 2009, 1,'B'),
(11,2009, 2, 'SIGL'),
(12 ,2009, 2, 'SRT');

-------------------------------------------------------------
-- Tuples epreuves

INSERT INTO epreuves VALUES
(1,301, '2006-10-17 17:30:00',1,1),
(2,302, '2006-10-17 17:30:00',1,2),
(3,302, '2006-10-21 9:00:00',2,1),
(4,302, '2006-10-21 14:00:00',2,2),
(5,101, '2006-11-10 9:00:00',3,1),
(6,101, '2006-11-10 14:00:00',3,2),
(7,106, '2006-11-20 9:00:00',4,1),
(8,107, '2006-11-21 9:00:00',4,2),
(9,302, '2007-01-12 9:00:00',6,1),
(10,301, '2007-01-12 9:00:00',6,2),
(11,302, '2007-01-25 15:00:00',5,1),
(12,303, '2007-01-25 15:00:00',5,2);

-------------------------------------------------------------
-- Tuples evaluer

INSERT INTO evaluer VALUES
(1,1,14),(1,5,11),(1,7,7),(1,9,12),(1,11,13),(2,1,6),
(2,3,10),(2,5,14),(2,7,12),(2,11,15),(3,1,17),(3,5,5),
(3,7,7),(3,9,9),(3,11,10),(4,1,8),(4,3,13),(4,5,9),(4,7,10),
(4,9,12),(4,11,12),(5,1,10),(5,5,12),(5,7,13),(5,11,18),
(6,1,6),(6,3,10),(6,5,8),(6,7,11),(6,9,10),(6,11,10),
(7,4,13),(7,6,9),(7,8,11),(7,12,14),(8,2,13),(8,6,13),
(8,8,13),(8,12,13),(9,2,6),(9,4,8),(9,6,4),(9,8,5),
(9,10,5),(9,12,7),(10,2,4),(10,4,9),(10,6,10),(10,8,11),
(10,12,11),(11,2,14),(11,6,4),(11,8,8),(11,10,5),(11,12,9),
(12,2,9),(12,4,11),(12,6,3),(12,10,7),(12,12,10),
(13,2,11),(13,6,11),(13,8,12),(13,12,16);

-------------------------------------------------------------
-- Tuples participer

INSERT INTO participer VALUES
(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),
(2,7),(2,8),(2,9),(2,10),(2,11),(2,12),(2,13),
(3,14),(3,15),(3,16),(3,17),(3,18),(3,19),(3,20),
(4,21),(4,22),(4,23),(4,24),(4,25),(4,26),(4,27),
(7,14),(8,15),(7,16),(8,17),(7,18),(8,19),(7,20),
(8,21),(7,22),(8,23),(7,24),(8,25),(7,26),(8,27),(7,28),
(5,30),(5,31),(5,32),(5,33),(5,34),(5,35),(5,36),
(11,29),(11,30),(11,31),(11,32),(11,33),(11,34),(11,35),
(11,36),(9,29),(9,30),(9,31),(9,32),(10,33),(10,34),(10,35),
(10,36);
```

ORACLE : calculette SQL*PLUS

Principales commandes

- **help index** (liste des commandes sqlplus)
- **help *nomCommande*** (documentation de la commande « nomCommande »)
- **select * from cat ;** (pour lister les tables de la base)
- **show user** ; (pour connaître l'utilisateur connecté)
- **connect *moi* / *mdp*** (pour connecter l'utilisateur « moi » avec son mot de passe « mdp »
- **host *nomDeCommande*** (pour exécuter une commande du système d'exploitation)
- **start *nomFichier* ;** (pour exécuter un fichier de commandes).
- **@ *nomFichier* ;** (équivalent à start)
- **show all** (pour voir les variables d'environnement)
- **set *nomVar valeur*** : (pour donner la valeur « valeur » à la variable d'environnement « nomVar »)

Formats d'affichage

linesize et pagesize

linesize et pagesize sont des variable d'environnement SQLPLUS :

- ***Consultation***

```
SQL> show linesize
SQL> show pagesize
```

- ***Modification***

```
SQL> set linesize 100   // Taille d'une ligne de résultats
SQL> set pagesize 200   // Taille d'une page de résultats
```

Ces commandes peuvent être placées dans le fichier login.sql

Taille des colonnes

```
SQL> column mgr format 999 // nombre sur 3 chiffres
SQL> column job format a4 // chaine sur 4 caractères
SQL> column job trunc // job tronqué à la taille max
```

Gestion des accents

Sous windows

Choisir un répertoire et passer la commande :

```
C:/monRepertoire> set NLS_LANG=FRENCH_FRANCE.WE8PC850
```

Dans ce répertoire, lancer SQLPLUS : les accents sont pris en compte.

```
C:/monRepertoire> sqlplus
```

Sous linux

```
SQL> alter session set nls_language=French;
SQL> alter session set nls_territory=France;
```

Connexion

Sous SE

```
C:> sqlplus
C:> sqlplus /nolog     -- pas d'affichage des commentaires
C:> sqlplus nomUser
C:> sqlplus nomUser/password
C:> sqlplus @nomFic    -- le fichier doit commencer par 1
connexion
C:> sqlplus /nolog @nomFic
```

Sous SQL

```
SQL> connect
SQL> connect / as sysdba – connexion comme sysdba
SQL> connect nomUser
SQL> connect nomUser/password
SQL> disconnect              // disconnect et exit valident
SQL> exit         // la transaction : commit
```

Afficher le nom de l'utilisateur connecté

```
SQL> show user
```

ou

```
SQL> select user from dual; //dual est une pseudo-table
```

login.sql

Le fichier login.sql s'exécute automatiquement au démarrage de slqplus.
Le fichier doit se trouver dans le répertoire de lancement de sqlplus.
Ce fichier va permettre de paramétrer l'environnement de travail : pagesize, linesize, etc.

Variables d'environnement

Lister toutes les variables

```
SQL> show all     // Lister toutes les variables
```

Principales variables d'environnement sqlplus

- user
- linesize, pagesize, column
- echo, termout, feedback, heading, trimspool
- autocommit

Afficher une variable

```
SQL> show nomVariable  // Affiche la valeur de la var.
```

```
SQL> show user
SQL> select user from dual; //dual est une pseudo-table
```

Modifie une variable

```
SQL> set linesize 80
```

Scripts

Exécuter un script

```
SQL> @nomFichier //exécute le fichier, .sql par défaut
SQL> start nomFichier  //équivalent à @
```

```
PAUSE : pour arrêter l'exécution d'un script
```

Commentaires

```
REM         ligne de commentaire dans un script
--                ligne de commentaire dans un script
/*                texte de commentaires dans un script
 */
```

Gestion de fichier

```
SQL> save nomFich [create, replace, append]
   //enregistre le buffer dans un fichier
SQL> get fichier
   // met le contenu du ficher dans un buffer
```

Gestion de l'affichage des résultats d'un script

```
SQL> spool fic.txt     // Copie l'affichage à l'ecran
                  // dans 'fic.lst'
SQL> spool off    // Stoppe la copie dans 'fic.lst'

SQL> set echo on    // affiche la commande en cours

SQL> set feedback on // affiche le nb lignes résultat
SQL> set heading on  // affiche l'entête des colonnes
SQL> set trimspool on  // supprime les blancs de fin
                          de ligne

SQL> set termout OFF   // supprime tout affichage
```

Gestion des transactions

```
SQL> show autocommit         // OFF par défaut
SQL> set autocommit {ON | OFF | IMMEDIATE} //ON ⇔ IMMEDIATE
```

ORACLE : Exemple de code Oracle

Les employés et les départements, version 1

```
DROP TABLE EMP;
DROP TABLE DEPT;

CREATE TABLE DEPT (
  DEPTNO              NUMBER(2) NOT NULL,
  DNAME               CHAR(14),
  LOC                 CHAR(13),
```

```
  CONSTRAINT DEPT_PRIMARY_KEY PRIMARY KEY (DEPTNO)
);

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

CREATE TABLE EMP (
  EMPNO               NUMBER(4) NOT NULL,
  ENAME               CHAR(10),
  JOB                 CHAR(9),
  HIREDATE            DATE,
  SAL                 NUMBER(7,2),
  COMM                NUMBER(7,2),
  DEPTNO              NUMBER(2) NOT NULL,
  MGR                 NUMBER(4),
  CONSTRAINT EMP_SELF_KEY FOREIGN KEY (MGR) REFERENCES EMP
(EMPNO),
  CONSTRAINT EMP_FOREIGN_KEY FOREIGN KEY (DEPTNO) REFERENCES DEPT
(DEPTNO),
  CONSTRAINT EMP_PRIMARY_KEY PRIMARY KEY (EMPNO)
);

INSERT INTO EMP VALUES (7839,'KING','PRESIDENT','17-NOV-
81',5000,NULL,10,NULL);
INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER','1-MAI-
81',2850,NULL,30,7839);
INSERT INTO EMP VALUES (7782,'CLARK','MANAGER','9-JUN-
81',2450,NULL,10,7839);
INSERT INTO EMP VALUES (7566,'JONES','MANAGER','2-AVR-
81',2975,NULL,20,7839);
INSERT INTO EMP VALUES (7654,'MARTIN','SALESMAN','28-SEP-
81',1250,1400,30,7698);
INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN','20-FEBV-
81',1600,300,30,7698);
INSERT INTO EMP VALUES (7844,'TURNER','SALESMAN','8-SEP-
81',1500,0,30,7698);
INSERT INTO EMP VALUES (7900,'JAMES','CLERK','3-DEC-
81',950,NULL,30,7698);
INSERT INTO EMP VALUES (7521,'WARD','SALESMAN','22-FEV-
81',1250,500,30,7698);
INSERT INTO EMP VALUES (7902,'FORD','ANALYST','3-DEC-
81',3000,NULL,20,7566);
INSERT INTO EMP VALUES (7369,'SMITH','CLERK','17-DEC-
80',800,NULL,20,7902);
INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST','09-DEC-
82',3000,NULL,20,7566);
INSERT INTO EMP VALUES (7876,'ADAMS','CLERK','12-JAN-
83',1100,NULL,20,7788);
INSERT INTO EMP VALUES (7934,'MILLER','CLERK','23-JAN-
82',1300,NULL,10,7782);

COMMIT;
```

- ***Remarques***

- On commence par supprimer les tables : ça supprime les tuples en même temps. A noter qu'on commence par la table des employés pour des questions d'intégrité référentielle.
- On crée les tables : on commence par la table des départements, pour des questions d'intégrité référentielle.
- On crée les tuples : on commence par les tuples départements, pour des questions d'intégrité référentielle.

Les employés et les départements, version 2

Code créé automatiquement par le logiciel power designer à partir d'un modèle conceptuel des données :

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 10g
*/
/* Created on:     23/09/2006 21:23:09
*/
/*==============================================================*/
alter table EMP
   drop constraint FK_EMP_A_POUR_SU_EMP;
alter table EMP
   drop constraint FK_EMP_TRAVAILLE_DEPT;

drop index A_POUR_SUPHIE_FK;
drop index TRAVAILLE_DANS_FK;

drop table DEPT;
drop table EMP;

/*==============================================================*/
/* Table: DEPT
*/
/*==============================================================*/
create table DEPT  (
   DEPTNO               INTEGER                         not null,
   DNAME                VARCHAR2(20)                    not null,
   LOC                  VARCHAR2(20),
   constraint PK_DEPT primary key (DEPTNO)
);

/*==============================================================*/
/* Table: EMP
*/
/*==============================================================*/
create table EMP  (
   EMPNO                INTEGER                         not null,
   DEPTNO               INTEGER                         not null,
   MGR                  INTEGER,
   ENAME                VARCHAR2(20)                    not null,
   JOB                  VARCHAR2(20),
   HIREDATE             DATE,
   SAL                  NUMBER,
   COMM                 NUMBER,
   constraint PK_EMP primary key (EMPNO)
);

/*==============================================================*/
/* Index: TRAVAILLE_DANS_FK
*/
/*==============================================================*/
create index TRAVAILLE_DANS_FK on EMP (
   DEPTNO ASC
);

/*==============================================================*/
```

```
/* Index: A_POUR_SUPHIE_FK
*/
/*==============================================================*/
create index A_POUR_SUPHIE_FK on EMP (
   MGR ASC
);

/*==============================================================*/
/* Clés étrangères
*/
/*==============================================================*/
alter table EMP
   add constraint FK_EMP_A_POUR_SU_EMP foreign key (MGR)
      references EMP (EMPNO);

alter table EMP
   add constraint FK_EMP_TRAVAILLE_DEPT foreign key (DEPTNO)
      references DEPT (DEPTNO);
```

- ***Remarques***
  - On commence par supprimer les clés étrangères puis les index, puis les tables.
  - Ensuite on crée les tables par ordre alphabétique.
  - Ensuite on crée les index.
  - Enfin on crée les clés étrangères.