Bases de données - Travaux Pratiques

Licence professionnelle

Logiciels libres et propriétaires pour les systèmes, réseaux, et bases de données

David Genest

Université d'Angers

Année universitaire 2008-2009

TP I

Bases de SQL

Bases de SQL (Structured Query Language)

Objectif

Langage simple, non procédural pour la manipulation de données et l'interrogation.

Bases de SQL (Structured Query Language)

Objectif

Langage simple, non procédural pour la manipulation de données et l'interrogation.

Années 1970 : *SEQUEL* basé sur le relationnel.
1980 : SQL

Bases de SQL (Structured Query Language)

Objectif

Langage simple, non procédural pour la manipulation de données et l'interrogation.

Années 1970 : *SEQUEL* basé sur le relationnel.
1980 : SQL

- Langage de définition des données (LDD)
- Langage de manipulation de données (LMD)

Normalisé par l'ANSI (SQL-92)... mais plusieurs variantes dans les SGBD.
SGBD utilisant SQL : MySQL, Oracle, PostgreSQL, DB2, Ingres, Access, SQL Server, ...

Définition (Base)

Une base de données est un ensemble de *tables*.
Une table est formée de *lignes* (tuples) et de *colonnes* (attributs, tous les éléments d'une colonne ont le même type).
Attention : Plusieurs lignes peuvent être égales.

SQL

Définition (Base)

Une base de données est un ensemble de *tables*.
Une table est formée de *lignes* (tuples) et de *colonnes* (attributs, tous les éléments d'une colonne ont le même type).
Attention : Plusieurs lignes peuvent être égales.

Définition (Identificateur)

Doit commencer par une lettre, peut utiliser lettres, chiffres, #, $ ou _.
SQL ne fait pas la différence entre minuscules et majuscules (pour les mots-clefs), mais peut faire une différence pour les identificateurs (MySQL: pour les noms des tables sous Unix, Oracle: non).
Un identificateur ne peut être un mot réservé du langage.

SQL

Définition (Base)

Une base de données est un ensemble de *tables*.
Une table est formée de *lignes* (tuples) et de *colonnes* (attributs, tous les éléments d'une colonne ont le même type).
Attention : Plusieurs lignes peuvent être égales.

Définition (Identificateur)

Doit commencer par une lettre, peut utiliser lettres, chiffres, #, $ ou _.
SQL ne fait pas la différence entre minuscules et majuscules (pour les mots-clefs), mais peut faire une différence pour les identificateurs (MySQL: pour les noms des tables sous Unix, Oracle: non).
Un identificateur ne peut être un mot réservé du langage.

Chaque table et chaque colonne a un nom (identificateur).
Nom d'une colonne : [*nomtable*.]*nomcolonne*

TP I

Bases de SQL

MySQL

Système de gestion de bases de données libre et gratuit.
`http://www.mysql.com`
Surtout utilisé en association avec un serveur Web (mais pas uniquement).
Pas toujours à la "norme SQL" mais évolue rapidement.
Documentation en ligne
`http://dev.mysql.com/doc/refman/5.0/fr/index.html`

Client et serveur

MySQL (comme la plupart des SGBD) est basé sur une architecture client-serveur :

- Le serveur attend les connexions des clients et exécute les commandes reçues.
- Les clients transmettent des commandes.

Client et serveur

MySQL (comme la plupart des SGBD) est basé sur une architecture client-serveur :

- Le serveur attend les connexions des clients et exécute les commandes reçues.
- Les clients transmettent des commandes.

Plusieurs clients MySQL existent : simple interpréteur SQL (`mysql`), interface graphique (`mysqlcc`), administration (`mysql-admin`), accès à partir d'un langage de programmation (Java, PHP, C++, etc.), accès à partir d'autres outils (ODBC), ...

Client MySQL

Quelques options de la commande `mysql`

- `-h, --host=`*name* nom du serveur
- `-u, --user=`*name* nom du compte utilisateur
- `-p, --password[=`*pass*`]` utilisation d'un mot de passe
- `--help` ou `man mysql`

Client MySQL

Quelques options de la commande `mysql`

- `-h, --host=`*name* nom du serveur
- `-u, --user=`*name* nom du compte utilisateur
- `-p, --password[=`*pass*`]` utilisation d'un mot de passe
- `--help` ou `man mysql`

Bases

Toute table fait partie d'une base.

- Création : `create database` *nomdb*
- Choix de la base à utiliser :
  - Commande SQL `use` *nomdb* ou
  - Argument `-D, --database=`*nomdb*

TP I

Bases de SQL

Langage de définition de données

Types

- Numériques
  - tinyint, smallint, integer
  - real, float (float est plus précis)

Langage de définition de données

Types

- Numériques
  - tinyint, smallint, integer
  - real, float (float est plus précis)
  - numeric(*taille, decimales*)
    Exemple : *numeric(6,2)*

Types

Types

- Caractères
  - char(lg) (complétée par espaces) $lg \leq 255$
  - varchar(lg) $lg \leq$ (255 pour MySQL) (2000 pour Oracle)
- Autres
  - *date* (AAAA-MM-JJ)
  - *time* (HH:MM:SS)
  - *timestamp* (AAAAMMJJHHMMSS).
  - *enum* (ex: ENUM ('Lundi', 'Mardi', 'Mercredi'))

Types

Types

- Caractères
  - char(lg) (complétée par espaces) $lg \leq 255$
  - varchar(lg) $lg \leq$ (255 pour MySQL) (2000 pour Oracle)
- Autres
  - *date* (AAAA-MM-JJ)
  - *time* (HH:MM:SS)
  - *timestamp* (AAAAMMJJHHMMSS).
  - *enum* (ex: ENUM ('Lundi', 'Mardi', 'Mercredi'))

NULL

case de la table non renseignée ($\neq 0$, $\neq''$).

Création d'une table

Syntaxe

```
CREATE TABLE <nom_table>
(<nom_colonne_1> <type_colonne_1>
 {, <nom_colonne_i> <type_colonne_i>}*
);
```

Création d'une table

Syntaxe

```
CREATE TABLE <nom_table>
(<nom_colonne_1> <type_colonne_1>
 {, <nom_colonne_i> <type_colonne_i>}*
);
```

Option `NOT NULL` après la déclaration d'une colonne pour interdire les valeurs `NULL`.

Option `DEFAULT valeur`.

Création d'une table

Syntaxe

```
CREATE TABLE <nom_table>
(<nom_colonne_1> <type_colonne_1>
 {, <nom_colonne_i> <type_colonne_i>}*
);
```

Option `NOT NULL` après la déclaration d'une colonne pour interdire les valeurs `NULL`.

Option `DEFAULT valeur`.

Exemple

```
CREATE TABLE Produit
(nom_p char(20) NOT NULL,
 nom_f varchar(200) DEFAULT 'Canson',
 prix numeric(6,2));
```

Contraintes

Expression de contraintes sur les valeurs autorisées dans les tables.
Fixées au moment de la création.

Syntaxe

```
CREATE TABLE <nom_table>
 (<nom_col_i> <type_col_i> <contrainte_col_i>, ...
  <contrainte_table>);
```

Contraintes

Expression de contraintes sur les valeurs autorisées dans les tables.
Fixées au moment de la création.

Syntaxe

```
CREATE TABLE <nom_table>
 (<nom_col_i> <type_col_i> <contrainte_col_i>, ...
  <contrainte_table>);
```

Syntaxe contrainte de colonne

MySQL : `type`

Oracle : `CONSTRAINT [nom] type`

Contraintes

Expression de contraintes sur les valeurs autorisées dans les tables.
Fixées au moment de la création.

Syntaxe

```
CREATE TABLE <nom_table>
 (<nom_col_i> <type_col_i> <contrainte_col_i>, ...
  <contrainte_table>);
```

Syntaxe contrainte de colonne

MySQL : `type`

Oracle : `CONSTRAINT [nom] type`

Syntaxe contrainte de table

`CONSTRAINT [nom] type (arguments, ...)`

Contrainte sur plusieurs colonnes (sur la table) (arguments : nom des colonnes).

Types de contraintes

Clé primaire

`PRIMARY KEY`

Une colonne (ou une concaténation de colonnes) ne peut contenir des valeurs identiques. Valeur `NULL` interdite.

Types de contraintes

Clé primaire

`PRIMARY KEY`

Une colonne (ou une concaténation de colonnes) ne peut contenir des valeurs identiques. Valeur `NULL` interdite.

Exemple

```
CREATE TABLE Fournisseur
(nom_f char(20) PRIMARY KEY, --MySQL
(nom_f char(20) CONSTRAINT fk PRIMARY KEY, --Oracle
 ... );
```

Types de contraintes

Clé primaire

`PRIMARY KEY`

Une colonne (ou une concaténation de colonnes) ne peut contenir des valeurs identiques. Valeur `NULL` interdite.

Exemple

```
CREATE TABLE Fournisseur
(nom_f char(20) PRIMARY KEY, --MySQL
(nom_f char(20) CONSTRAINT fk PRIMARY KEY, --Oracle
 ... );
```

Exemple

```
CREATE TABLE ContenuCommande
(numero numeric(6,0),
 nom_p char(20),
 quantite numeric(3),
 CONSTRAINT ck PRIMARY KEY (numero, nom_p));
```

Types de contraintes

Unicité de valeurs

`UNIQUE`

Une colonne (ou une concaténation de colonnes) ne peut contenir des valeurs identiques. (clé)

Types de contraintes

Unicité de valeurs

`UNIQUE`

Une colonne (ou une concaténation de colonnes) ne peut contenir des valeurs identiques. (clé)

Exemple

```
CREATE TABLE Commandes
(numero numeric(6,0) PRIMARY KEY,
 date_c date,
 nom_c varchar(20),
 CONSTRAINT dc_uni UNIQUE (date_c, nom_c));
```

Suppression d'une table

Syntaxe

```
DROP TABLE nom_table;
```

TP I

Bases de SQL

1. MySQL
2. Langage de définition de données
3. Langage de manipulation de données
   - Insertion de données
   - Sélection de données
   - Modification de données
   - Suppression de données

TP I

Bases de SQL

Langage de manipulation de données
Insertion de données

Syntaxe

```
INSERT INTO <table> [(<col_1> [{, <col_i>}*)]]
VALUES (<val_1> {, <val_i>}*);
```

Langage de manipulation de données
Insertion de données

Syntaxe

```
INSERT INTO <table> [(<col_1> [{, <col_i>}*)]]
VALUES (<val_1> {, <val_i>}*);
```

Si la liste de colonnes n'est pas donnée, colonnes dans l'ordre de la déclaration de la table (donner une valeur par colonne).

Exemple

```
INSERT INTO Fournisseur
VALUES ('Canson', '1 rue de la Paix');
```

Langage de manipulation de données
Insertion de données

Syntaxe

```
INSERT INTO <table> [(<col_1> [{, <col_i>}*)]]
VALUES (<val_1> {, <val_i>}*);
```

Si la liste de colonnes n'est pas donnée, colonnes dans l'ordre de la déclaration de la table (donner une valeur par colonne).

Exemple

```
INSERT INTO Fournisseur
VALUES ('Canson', '1 rue de la Paix');
```

Les colonnes ne figurant pas dans la liste auront la valeur NULL (ou par défaut).

Exemple

```
INSERT INTO Produit (nom_p, prix)
VALUES ('500N', 12);
```

TP I

Bases de SQL

Sélection de données

Syntaxe

```
SELECT [DISTINCT] * | <expr_1> [AS <nom_1>]
                    , <expr_i> [AS <nom_i>]*
FROM <table_1> [<synonyme_1>]
  {, <table_i> [<synonyme_i>]}*
[WHERE <condition>];
```

Sélection de données

Syntaxe

```
SELECT [DISTINCT] * | <expr_1> [AS <nom_1>]
                    , <expr_i> [AS <nom_i>]*
FROM <table_1> [<synonyme_1>]
  {, <table_i> [<synonyme_i>]}*
[WHERE <condition>];
```

1. From (à partir de quelles tables ?)
2. Where (quels tuples ?)
3. Select (comment présenter les résultats ?)

Condition du `where`

- Opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $=$, $!=$
  expr BETWEEN expr1 AND expr2 (bornes incluses)
  expr IS [NOT] NULL

Condition du `where`

- Opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $=$, $!=$
  expr BETWEEN expr1 AND expr2 (bornes incluses)
  expr IS [NOT] NULL
- *expr [NOT] IN (expr1, ..)*

Condition du `where`

- Opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $=$, $!=$
  expr BETWEEN expr1 AND expr2 (bornes incluses)
  expr IS [NOT] NULL
- *expr [NOT] IN (expr1, ..)*
- Opérateurs arithmétiques : $+$, $-$, $*$, $/$

Condition du `where`

- Opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $=$, $!=$
  expr BETWEEN expr1 AND expr2 (bornes incluses)
  expr IS [NOT] NULL
- *expr [NOT] IN (expr1, ..)*
- Opérateurs arithmétiques : $+$, $-$, $*$, $/$
- Opérateurs sur les chaînes : $||$ (concaténation)
  chaine1 [NOT] LIKE chaine2 : Utilisation de jokers dans *chaine2* (_ : un caractère, *%* : 0 ou plusieurs caractères)

Condition du `where`

- Opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $=$, $!=$
  expr BETWEEN expr1 AND expr2 (bornes incluses)
  expr IS [NOT] NULL
- *expr [NOT] IN (expr1, ..)*
- Opérateurs arithmétiques : $+$, $-$, $*$, $/$
- Opérateurs sur les chaînes : $||$ (concaténation)
  chaine1 [NOT] LIKE chaine2 : Utilisation de jokers dans *chaine2* (_ : un caractère, % : 0 ou plusieurs caractères)
- Opérateurs sur les dates : - (nombre de jours entre deux dates)

Condition du `where`

- Opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $=$, $!=$
  expr BETWEEN expr1 AND expr2 (bornes incluses)
  expr IS [NOT] NULL
- *expr [NOT] IN (expr1, ..)*
- Opérateurs arithmétiques : $+$, $-$, $*$, $/$
- Opérateurs sur les chaînes : $||$ (concaténation)
  chaine1 [NOT] LIKE chaine2 : Utilisation de jokers dans *chaine2* (_ : un caractère, *%* : 0 ou plusieurs caractères)
- Opérateurs sur les dates : - (nombre de jours entre deux dates)
- Opérateurs logiques : *AND*, *OR*, *NOT*.

Exemple (SELECT * FROM Produit;)

NOM_P	NOM_F	PRIX
500N	Canson	12
1000C	Canson	25
Cartouche HP500	Pelikan	20

Exemple (SELECT * FROM Produit;)

NOM_P	NOM_F	PRIX
500N	Canson	12
1000C	Canson	25
Cartouche HP500	Pelikan	20

Exemple (SELECT nom_f FROM Produit;)

NOM_F
Canson
Canson
Pelikan

Exemple (SELECT * FROM Produit;)

NOM_P	NOM_F	PRIX
500N	Canson	12
1000C	Canson	25
Cartouche HP500	Pelikan	20

Exemple (SELECT nom_f FROM Produit;)

NOM_F
Canson
Canson
Pelikan

Exemple (SELECT DISTINCT nom_f FROM Produit;)

NOM_F
Canson
Pelikan

Exemple

```
SELECT nom_p, prix+10 AS prixmajore
FROM Produit
WHERE (prix BETWEEN 5 AND 15) AND (nom_f LIKE 'Can%');
```

NOM_P	PRIXMAJORE
500N	22

Exemple

```
SELECT nom_p, prix+10 AS prixmajore
FROM Produit
WHERE (prix BETWEEN 5 AND 15) AND (nom_f LIKE 'Can%');
```

NOM_P	PRIXMAJORE
500N	22

Attention

`expr = NULL` est toujours faux.

"NULL = inconnu, et inconnu est différent d'inconnu".

Exemple

```
SELECT nom_p, prix+10 AS prixmajore
FROM Produit
WHERE (prix BETWEEN 5 AND 15) AND (nom_f LIKE 'Can%');
```

NOM_P	PRIXMAJORE
500N	22

Attention

`expr = NULL` est toujours faux.
"NULL = inconnu, et inconnu est différent d'inconnu".

Exemple

```
SELECT nom_p
FROM Produit
WHERE prix IS NULL;
```

Interrogations : Jointure

`from` peut être utilisé pour calculer le produit cartésien...
et `where` peut être utilisé pour faire des jointures.

Interrogations : Jointure

`from` peut être utilisé pour calculer le produit cartésien...
et `where` peut être utilisé pour faire des jointures.

Exemple

```
SELECT nom_p, Produit.nom_f, adresse
FROM Produit, Fournisseur
WHERE (Produit.nom_f = Fournisseur.nom_f);
```

NOM_P	NOM_F	ADRESSE
500N	Canson	1 rue de la Paix
1000C	Canson	1 rue de la Paix

Interrogations : Jointure

`from` peut être utilisé pour calculer le produit cartésien...
et `where` peut être utilisé pour faire des jointures.

Exemple

```
SELECT nom_p, Produit.nom_f, adresse
FROM Produit, Fournisseur
WHERE (Produit.nom_f = Fournisseur.nom_f);
```

NOM_P	NOM_F	ADRESSE
500N	Canson	1 rue de la Paix
1000C	Canson	1 rue de la Paix

Il est possible d'utiliser des opérateurs autres que $=$
$\longrightarrow$ θ-jointure
Il existe d'autres façons d'exprimer une jointure en SQL.

Interrogations : Jointure

On peut aussi utiliser une syntaxe particulière :

Syntaxe

```
select * from table1 NATURAL JOIN table2;
```

Jointure : Égalité des attributs de même nom

Interrogations : Jointure

On peut aussi utiliser une syntaxe particulière :

Syntaxe

```
select * from table1 NATURAL JOIN table2;
```

Jointure : Égalité des attributs de même nom

```
select * from table1 JOIN table2 USING (col1, col2,
...)
```

Égalité des attributs `col1`, `col2` ...

Interrogations : Jointure

On peut aussi utiliser une syntaxe particulière :

Syntaxe

```
select * from table1 NATURAL JOIN table2;
```

Jointure : Égalité des attributs de même nom

```
select * from table1 JOIN table2 USING (col1, col2,
...)
```

Égalité des attributs `col1`, `col2` ...

```
select * from table1 JOIN table2 ON (<expression>)
```

θ-jointure

TP I

Bases de SQL

Modification de données

Syntaxe

```
UPDATE <table>
SET <col_1> = <expr_1> {, <col_i> = <expr_i>}*
WHERE <condition>;
```

Modification de données

Syntaxe

```
UPDATE <table>
SET <col_1> = <expr_1> {, <col_i> = <expr_i>}*
WHERE <condition>;
```

Exemple

```
UPDATE Produit
SET prix = 14
WHERE nom_p = '500N';
```

Modification de données

Syntaxe

```
UPDATE <table>
SET <col_1> = <expr_1> {, <col_i> = <expr_i>}*
WHERE <condition>;
```

Exemple

```
UPDATE Produit
SET prix = 14
WHERE nom_p = '500N';
```

Exemple

```
UPDATE Produit
SET prix = prix * 2
WHERE nom_f = 'Canson';
```

TP I

Bases de SQL

Suppression de données

Syntaxe

```
DELETE FROM <table>
[WHERE <condition>];
```

Suppression de données

Syntaxe

```
DELETE FROM <table>
[WHERE <condition>];
```

Attention

Si la clause `where` n'est pas donnée, toutes les lignes sont supprimées.

Suppression de données

Syntaxe

```
DELETE FROM <table>
[WHERE <condition>];
```

Attention

Si la clause `where` n'est pas donnée, toutes les lignes sont supprimées.

Exemple

```
DELETE FROM Produit
WHERE nom_f like '_an%';
```

TP II

Modification des tables, Tris, Auto-incrément

TP II

Modification des tables, Tris, Auto-incrément

Modification des tables

La commande `alter table` permet de modifier la structure d'une table existante. Seules certaines modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.

Modification des tables

La commande `alter table` permet de modifier la structure d'une table existante. Seules certaines modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.

Syntaxe

```
alter table <nomtable>
add <col> <type>;
```

Modification des tables

La commande `alter table` permet de modifier la structure d'une table existante. Seules certaines modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.

Syntaxe

```
alter table <nomtable>
add <col> <type>;
```

`NOT NULL` ne peut être utilisé si la table n'est pas vide, car dans ce cas, des valeurs *null* sont utilisées pour les lignes existantes.

alter table : Modification d'une colonne

Syntaxe

```
alter table <nomtable>
modify <col> <type>;
```

col étant le nom d'une colonne existante.

`alter table` : Modification d'une colonne

Syntaxe

```
alter table <nomtable>
modify <col> <type>;
```

`col` étant le nom d'une colonne existante.
On ne peut pas diminuer la taille d'une colonne. Peut être utilisé pour ajouter une option `not null` ou `default`.

`alter table` : Modification d'une colonne

Syntaxe

```
alter table <nomtable>
modify <col> <type>;
```

`col` étant le nom d'une colonne existante.
On ne peut pas diminuer la taille d'une colonne. Peut être utilisé pour ajouter une option `not null` ou `default`.

Exemple

```
alter table vehicule add couleur varchar(15);
alter table sinistre modify numero numeric(12);
```

alter table : Modification d'une colonne

Syntaxe

```
alter table <nomtable>
modify <col> <type>;
```

col étant le nom d'une colonne existante.
On ne peut pas diminuer la taille d'une colonne. Peut être utilisé pour ajouter une option not null ou default.

Exemple

```
alter table vehicule add couleur varchar(15);
alter table sinistre modify numero numeric(12);
```

Plus de renseignements :
http://dev.mysql.com/doc/refman/5.0/fr/alter-table.html

TP II

Modification des tables, Tris, Auto-incrément

Tris

Permet d'ordonner les lignes en fonction des valeurs des colonnes (par défaut dans l'ordre croissant, utiliser le mot-clef `DESC` pour un classement dans l'ordre décroissant).

Syntaxe

```
select ...  from ...  where ...
order by <expr1> [DESC], ...;
```

Tris

Permet d'ordonner les lignes en fonction des valeurs des colonnes (par défaut dans l'ordre croissant, utiliser le mot-clef `DESC` pour un classement dans l'ordre décroissant).

Syntaxe

```
select ...  from ...  where ...
order by <expr1> [DESC], ...;
```

Les `<expr`i`>` sont habituellement des noms de colonnes des tables précisées dans la clause `from`.
Le tri se fait sur la première expression donnée, les lignes ayant même valeur sont ensuite classées selon la seconde expression, etc.

Tris

Après une clause `order by` on utilise parfois une clause
`limit <nblignes>`
pour obtenir les `nblignes` premiers résultats (extension MySQL).

Exemple

```
select * from sinistre
order by montant
limit 1;
```

Tris

Après une clause `order by` on utilise parfois une clause
`limit <nblignes>`
pour obtenir les `nblignes` premiers résultats (extension MySQL).

Exemple

```
select * from sinistre
order by montant
limit 1;
```

`order by` et `limit` peuvent être utilisés avec `select`, `update` et `delete`.

TP II

Modification des tables, Tris, Auto-incrément

Attributs auto incrémentés

`auto_increment` peut être utilisé lors de la déclaration de la table pour un attribut dont des valeurs uniques doivent être générées automatiquement.

Exemple

```
CREATE TABLE Commandes
(numero integer PRIMARY KEY AUTO_INCREMENT,
 date_c date,
 ...);
```

Attributs auto incrémentés

`auto_increment` peut être utilisé lors de la déclaration de la table pour un attribut dont des valeurs uniques doivent être générées automatiquement.

Exemple

```
CREATE TABLE Commandes
(numero integer PRIMARY KEY AUTO_INCREMENT,
 date_c date,
 ...);
```

Habituellement, on ne donne pas de valeur à cet attribut lors d'un `insert`, et on précise donc les noms des autres attributs :

```
insert into Commande(date_c, ...) values
('2007-10-15', ...);
```

Attributs auto incrémentés

`auto_increment` n'est pas utilisable avec un type `numeric` mais uniquement avec les types `int` (=`integer`).

Attributs auto incrémentés

`auto_increment` n'est pas utilisable avec un type `numeric` mais uniquement avec les types `int` (=`integer`).

- Le type `numeric` représente le nombre de façon exacte (représentation sous forme de chaîne de caractère en base 10).

Attributs auto incrémentés

`auto_increment` n'est pas utilisable avec un type `numeric` mais uniquement avec les types `int` (=`integer`).

- Le type `numeric` représente le nombre de façon exacte (représentation sous forme de chaîne de caractère en base 10).
- Les types `tinyint` (1), `smallint` (2), `mediumint` (3), `int` (4), `bigint` (8) représentent les entiers en binaire sous la forme de 1 ou plusieurs octets.

Attributs auto incrémentés

`auto_increment` n'est pas utilisable avec un type `numeric` mais uniquement avec les types `int` (=`integer`).

- Le type `numeric` représente le nombre de façon exacte (représentation sous forme de chaîne de caractère en base 10).
- Les types `tinyint` (1), `smallint` (2), `mediumint` (3), `int` (4), `bigint` (8) représentent les entiers en binaire sous la forme de 1 ou plusieurs octets.
- Les types `float`, `real`, et `double precision` représentent des réels de façon approchée.

Attributs auto incrémentés

La première valeur utilisée est 1.

Attributs auto incrémentés

La première valeur utilisée est 1.
La suppression de lignes (y compris toutes les lignes) d'une table ne réinitialise pas obligatoirement le compteur.

Attributs auto incrémentés

La première valeur utilisée est 1.
La suppression de lignes (y compris toutes les lignes) d'une table ne réinitialise pas obligatoirement le compteur.

```
alter table <nomtable>
auto_increment=<valeurinitiale>;
```

permet de forcer la valeur initiale.

TP II

Modification des tables, Tris, Auto-incrément

Une autre variante de l'opérateur JOIN

Syntaxe

```
select * from table1
LEFT JOIN table2
[ON...| USING...]
```

Une autre variante de l'opérateur JOIN

Syntaxe

```
select * from table1
LEFT JOIN table2
[ON...| USING...]
```

Attention

Il ne s'agit pas de la semi-jointure.

Une autre variante de l'opérateur JOIN

Syntaxe

```
select * from table1
LEFT JOIN table2
[ON...| USING...]
```

Attention

Il ne s'agit pas de la semi-jointure.

Effectue la jointure selon les conditions données, mais conserve dans le résultat tous les tuples de `table1`, même ceux qui n'interviennent pas dans la jointure. Pour ces tuples, des valeurs `NULL` sont utilisées pour les attributs de `table2`.

Une autre variante de l'opérateur JOIN

Syntaxe

```
select * from table1
LEFT JOIN table2
[ON...| USING...]
```

Attention

Il ne s'agit pas de la semi-jointure.

Effectue la jointure selon les conditions données, mais conserve dans le résultat tous les tuples de `table1`, même ceux qui n'interviennent pas dans la jointure. Pour ces tuples, des valeurs `NULL` sont utilisées pour les attributs de `table2`.

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

TP III

Fonctions, Groupes, Droits

1. Fonctions
2. Groupes
3. La clause `HAVING`
4. Contrôle d'accès aux tables
5. `UPDATE` multi-table

TP III

Fonctions, Groupes, Droits

Fonctions

Les fonctions peuvent être utilisées dans des requêtes.

Fonctions

Les fonctions peuvent être utilisées dans des requêtes.
La description des fonctions utilisables dans MySQL est donnée à l'URL :

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

(section 12)

Fonctions
Exemple d'utilisation

Exemple (La fonction `sum(`*`nomcol`*`)`)

```
select sum(salaire) from membre;
```

retourne une relation formée d'une ligne et une colonne contenant la somme des salaires de `membre`.

Fonctions
Quelques fonctions

(Habituellement `<expr>` est le nom d'une colonne.)

- `avg(<expr>)` Moyenne

Fonctions
Quelques fonctions

(Habituellement `<expr>` est le nom d'une colonne.)

- `avg(<expr>)` Moyenne
- `min(<expr>)` `max(<expr>)` Plus petite et plus grande valeur

Fonctions
Quelques fonctions

(Habituellement `<expr>` est le nom d'une colonne.)

- `avg(<expr>)` Moyenne
- `min(<expr>)` `max(<expr>)` Plus petite et plus grande valeur
- `count(*)` Nombre de lignes

Fonctions
Quelques fonctions

(Habituellement `<expr>` est le nom d'une colonne.)

- `avg(<expr>)` Moyenne
- `min(<expr>)` `max(<expr>)` Plus petite et plus grande valeur
- `count(*)` Nombre de lignes
- `count(<expr>)` Nombre de valeurs non nulles

Fonctions
Quelques fonctions

(Habituellement `<expr>` est le nom d'une colonne.)

- `avg(<expr>)` Moyenne
- `min(<expr>)` `max(<expr>)` Plus petite et plus grande valeur
- `count(*)` Nombre de lignes
- `count(<expr>)` Nombre de valeurs non nulles
- `count(distinct <expr>)` Nombre de valeurs non nulles distinctes

TP III

Fonctions, Groupes, Droits

Groupes

Syntaxe

La clause `GROUP BY <expr_1>, <expr_2>, ...`, placée après le `where`, permet de "grouper" en une seule ligne les lignes ayant mêmes valeurs sur les colonnes `expr_1`, `expr_2`, etc

Groupes

Syntaxe

La clause `GROUP BY <expr_1>, <expr_2>, ...`, placée après le `where`, permet de "grouper" en une seule ligne les lignes ayant mêmes valeurs sur les colonnes `expr_1`, `expr_2`, etc

Exemple

L'interrogation suivante retournera 2 lignes (`PRJ1` et `PRJ2`).

```
select projet from affectation group by projet;
```

Groupes

L'intérêt de cette clause est de pouvoir utiliser des fonctions qui sont alors appliquées sur chaque groupe. Ainsi l'interrogation suivante affichera, pour *chaque* projet, le nombre d'employés affectés au projet.

Exemple

```
select projet, count(*)
 from affectation
 group by projet;
```

PROJET	COUNT(*)
PRJ1	5
PRJ2	1

TP III

Fonctions, Groupes, Droits

La clause HAVING

- suit la clause `group by`
- doit être suivie d'une condition portant sur le *groupe* considéré.

Cette condition a la même syntaxe qu'une condition de la clause `where`, mais ne peut porter que sur le groupe (expressions du `group by` ou fonction de groupe).

La clause HAVING

- suit la clause `group by`
- doit être suivie d'une condition portant sur le *groupe* considéré.

Cette condition a la même syntaxe qu'une condition de la clause `where`, mais ne peut porter que sur le groupe (expressions du `group by` ou fonction de groupe).
`where` permet de sélectionner des lignes.
`having` permet de sélectionner des groupes.

La clause HAVING

- suit la clause `group by`
- doit être suivie d'une condition portant sur le *groupe* considéré.

Cette condition a la même syntaxe qu'une condition de la clause `where`, mais ne peut porter que sur le groupe (expressions du `group by` ou fonction de groupe).
`where` permet de sélectionner des lignes.
`having` permet de sélectionner des groupes.

Exemple

```
select projet
from affectation
group by projet
having count(*) > 1;
```

TP III

Fonctions, Groupes, Droits

Contrôle d'accès aux tables

Le *propriétaire* d'une table (l'utilisateur qui l'a créée) a tous les droits sur la table, et les autres utilisateurs n'ont aucun droit.

Contrôle d'accès aux tables

Le *propriétaire* d'une table (l'utilisateur qui l'a créée) a tous les droits sur la table, et les autres utilisateurs n'ont aucun droit.
Le propriétaire peut accorder des droits aux autres utilisateurs sur certaines de ses tables.

TP III

Fonctions, Groupes, Droits

La commande GRANT

Syntaxe

```
GRANT privilège, ...
ON nomtable
TO utilisateur, ...
[WITH GRANT OPTION];
```

La commande GRANT

Syntaxe

```
GRANT privilège, ...
ON nomtable
TO utilisateur, ...
[WITH GRANT OPTION];
```

privilège

select	lecture	insert	insertion de lignes
update	mise à jour	update(*col1*,...)	mise à jour de colonnes
delete	suppression	alter	modification du schéma

et all tous les droits.

La commande GRANT

Syntaxe

```
GRANT privilège, ...
ON nomtable
TO utilisateur, ...
[WITH GRANT OPTION];
```

privilège

select	lecture	insert	insertion de lignes
update	mise à jour	update(*col1*,...)	mise à jour de colonnes
delete	suppression	alter	modification du schéma

et all tous les droits.

(Section 13.5.1.3 du manuel)
http://dev.mysql.com/doc/refman/5.0/fr/grant.html

GRANT

Exemple

L'utilisateur *Util1* donne à *Util2* et *Util3* la possibilité de lire la table *Membre* et de modifier les lignes de cette table.

```
grant select, update on Membre to Util2, Util3;
```

Util2 accèdera à cette table par *nombase*.Membre

GRANT

Exemple

L'utilisateur *Util1* donne à *Util2* et *Util3* la possibilité de lire la table *Membre* et de modifier les lignes de cette table.

```
grant select, update on Membre to Util2, Util3;
```

Util2 accèdera à cette table par *nombase*.Membre

with grant option

L'utilisateur ayant reçu un privilège accompagné de `with grant option` peut retransmettre ce privilège à d'autres utilisateurs.

TP III

Fonctions, Groupes, Droits

La commande REVOKE

Un utilisateur ayant accordé un privilège (propriétaire ou utilisateur ayant ce droit) peut le retirer avec la commande REVOKE.

Syntaxe

```
REVOKE privilège, ...
ON table
FROM utilisateur;
```

Si l'utilisateur à qui on retire un privilège l'avait accordé à d'autres, tous ces privilèges sont retirés.

TP III

Fonctions, Groupes, Droits

UPDATE multi-table (spécifique à MySQL)

Il est parfois nécessaire dans un `update` d'utiliser une autre table (en lecture seule).

Syntaxe

```
UPDATE produits,prixreferences
SET produits.prix=prixreferences.prix
WHERE produits.id=prixpreferences.id;
```

(Section 13.1.10)

TP IV

PostgreSQL, Sous-interrogation

PostgreSQL

PostgreSQL, comme MySQL est un SGBDR libre.
Il offre plus de fonctions que la version actuelle de MySQL :

- Sous-interrogation.
- Transactions.
- Clés étrangères.
- Vues.

`http://www.postgresql.org`
`http://www.postgresqlfr.org`

TP IV

PostgreSQL, Sous-interrogation

Utilisation de PostgreSQL

Basé sur une architecture client-serveur.
Le client "simple" officiel est `psql`.

Syntaxe

```
psql [OPTIONS] NOMBASE
```

Utilisation de PostgreSQL

Basé sur une architecture client-serveur.
Le client "simple" officiel est `psql`.

Syntaxe

`psql [OPTIONS] NOMBASE`

- `-U <nom_utilisateur>` : Choix du nom d'utilisateur pour la connexion.
- `-h <hote_serveur>` : Choix du serveur
- `-W` : Demander un mot de passe.

Quelques commandes de `psql`

- `\?` Aide sur les commandes psql
- `\h` Aide sur les commandes sql
- `\q` Quitter

Quelques commandes de `psql`

- `\?` Aide sur les commandes psql
- `\h` Aide sur les commandes sql
- `\q` Quitter
- `\c <base>` Se reconnecter à une autre base
- `\dt` Afficher les tables de la base courante.
- `\d <table>` Décrire la table

TP IV

PostgreSQL, Sous-interrogation

Sous interrogation

Principe

Utiliser le résultat d'une requête `select` dans une clause `where`.

Sous interrogation

Principe

Utiliser le résultat d'une requête `select` dans une clause `where`.

Fonctionne sous PostgreSQL, Oracle et dans les versions récentes de MySQL.

TP IV

PostgreSQL, Sous-interrogation

Cas où le `select` retourne une ligne et une colonne

Syntaxe

`...where <expr> <op> (select ...).`

op $\in \{=, !=, <, >, <=, >=\}$.

Cas où le `select` retourne une ligne et une colonne

Syntaxe

`...where <expr> <op> (select ...).`

op $\in \{=, !=, <, >, <=, >=\}$.

Attention si le `select` emboîté ne retourne pas *une* ligne et *une* colonne, la requête produit une erreur.

TP IV

PostgreSQL, Sous-interrogation

Cas où le `select` retourne une colonne (et un nombre quelconque de lignes)

Il est possible d'utiliser des opérateurs permettant de comparer une valeur (expression dans le `where`) et un ensemble de valeurs (résultat du `select`).

Syntaxe

`...where <expr> <op> (select ...).`

`op` $\in$ {in, not in, `<opsimple> any`, `<opsimple> all` }.

`opsimple` $\in \{=, !=, <, >, <=, >=\}$

in - not in

in (not in)

La condition est vraie si la valeur de l'expression est (n'est pas) une des valeurs retournées par le `select`.

in - not in

in (not in)

La condition est vraie si la valeur de l'expression est (n'est pas) une des valeurs retournées par le select.

Exemple

```
select *
from sinistre
where conducteur in
   (select numss
    from client
    where nom='Rudoux');
```

`any`, `all`

- opsimple `any` : La condition est vraie si elle est vraie pour *au moins* un élément du résultat du `select` (fausse si le résultat est vide).
- opsimple `all` : La condition est vraie si elle est vraie pour *tous* les éléments du résultat du `select` (vraie si le résultat est vide).

any, all

- opsimple any : La condition est vraie si elle est vraie pour *au moins* un élément du résultat du select (fausse si le résultat est vide).
- opsimple all : La condition est vraie si elle est vraie pour *tous* les éléments du résultat du select (vraie si le résultat est vide).

Exemple

```
select numero
from sinistre
where montant >= all
  (select montant from sinistre);
```

TP IV

PostgreSQL, Sous-interrogation

Cas où le `select` retourne plusieurs colonnes

Syntaxe

La syntaxe est la même que dans le cas d'une sous-interrogation retournant une colonne :

`...where <expr> <op> (select ...)`.

op $\in$ {in, not in, `<opsimple> any`, `<opsimple> all`}

opsimple $\in \{=, !=\}$.

Cas où le `select` retourne plusieurs colonnes

Syntaxe

La syntaxe est la même que dans le cas d'une sous-interrogation retournant une colonne :

`...where <expr> <op> (select ...)`.

op $\in$ {in, not in, `<opsimple> any`, `<opsimple> all`}

`opsimple` $\in \{=, !=\}$.

La différence est dans `<expr>` qui doit ici être une expression formée de plusieurs expressions atomiques : `(<expr1>, ... <exprn>)`

Cas où le `select` retourne plusieurs colonnes

Syntaxe

La syntaxe est la même que dans le cas d'une sous-interrogation retournant une colonne :
`...where <expr> <op> (select ...)`.
op $\in$ {in, not in, `<opsimple> any`, `<opsimple> all`}
opsimple $\in \{=, !=\}$.
La différence est dans `<expr>` qui doit ici être une expression formée de plusieurs expressions atomiques : `(<expr1>, ... <exprn>)`

Exemple

Membres ayant le même salaire et le même supérieur que *Deray*.

```
select nom from membre
where (salaire, superieur) =
  (select salaire, superieur from membre
   where nom = 'Deray');
```

TP IV

PostgreSQL, Sous-interrogation

exists et not exists

- La condition est vraie ssi la sous-interrogation retourne au moins une ligne (`exists`).
  La condition est vraie ssi la sous-interrogation retourne un ensemble vide (`not exists`).

exists et not exists

- La condition est vraie ssi la sous-interrogation retourne au moins une ligne (`exists`).
  La condition est vraie ssi la sous-interrogation retourne un ensemble vide (`not exists`).
- La sous-interrogation peut utiliser des expressions faisant intervenir les tables de l'interrogation.

exists et not exists

- La condition est vraie ssi la sous-interrogation retourne au moins une ligne (`exists`).
  La condition est vraie ssi la sous-interrogation retourne un ensemble vide (`not exists`).
- La sous-interrogation peut utiliser des expressions faisant intervenir les tables de l'interrogation.
- La sous-interrogation est exécutée pour chaque ligne de l'interrogation.

Exemple

Exemple

Noms des projets sur lesquels une personne payée plus de 7000 est affectée.

Exemple

Exemple

Noms des projets sur lesquels une personne payée plus de 7000 est affectée.

```
select nom from projet
where exists
   (select * from membre, affectation
    where (affectation.projet = projet.nom)
      and (affectation.employe = membre.numero)
      and (membre.salaire > 7000));
```

Exemple

Exemple

Noms des projets sur lesquels une personne payée plus de 7000 est affectée.

```
select nom from projet
where exists
   (select * from membre, affectation
    where (affectation.projet = projet.nom)
      and (affectation.employe = membre.numero)
      and (membre.salaire > 7000));
```

Remarquer l'usage de `projet.nom` dans la sous-interrogation.
La sous-interrogation est exécutée pour chaque projet.

Exemple

Exemple

Noms des projets sur lesquels personne n'est affecté.

```
select nom from projet
where not exists
    (select NULL
     from affectation
     where affectation.projet = projet.nom);
```

Remarquer l'utilisation de `null` comme expression indiquant les colonnes utiles de la sous interrogation, ici seule l'existence d'une réponse dans la sous-interrogation est importante, pas le contenu de cette réponse.

TP IV

PostgreSQL, Sous-interrogation

Contraintes

Syntaxe

```
constraint <idcontrainte> <typecontrainte>
<arguments>
```

Contraintes

Syntaxe

```
constraint <idcontrainte> <typecontrainte>
<arguments>
```

Exemple

- Contrainte de colonne

  ```
  nom varchar(15) constraint ckey primary key
  ```

Contraintes

Syntaxe

```
constraint <idcontrainte> <typecontrainte>
<arguments>
```

Exemple

- Contrainte de colonne

  ```
  nom varchar(15) constraint ckey primary key
  ```

- Contrainte de table

  ```
  constraint ckey primary key (cursus, matiere)
  ```

(même syntaxe sous Oracle)

Contraintes check

valables sous MySQL, PostgreSQL, Oracle
check permet d'exprimer des contraintes sur les valeurs des colonnes à l'intérieur d'une ligne.

Exemple

```
create table Membre(
...
salaire numeric(8,2),
constraint salairemin check(salaire >= 500)
);
```

TP IV

PostgreSQL, Sous-interrogation

Transactions

- Certaines modifications laissent la base dans un état incohérent (*supprimer un fournisseur sans supprimer ses produits*).
  La cohérence est retrouvée après d'autres modifications (*supprimer les produits de ce fournisseur*).

Transactions

- Certaines modifications laissent la base dans un état incohérent (*supprimer un fournisseur sans supprimer ses produits*).
  La cohérence est retrouvée après d'autres modifications (*supprimer les produits de ce fournisseur*).
- Une **transaction** est un ensemble de modifications qui conservent la cohérence de la base. (à l'intérieur d'une transaction, l'état peut être incohérent).

Transactions

- Certaines modifications laissent la base dans un état incohérent (*supprimer un fournisseur sans supprimer ses produits*).
  La cohérence est retrouvée après d'autres modifications (*supprimer les produits de ce fournisseur*).
- Une **transaction** est un ensemble de modifications qui conservent la cohérence de la base. (à l'intérieur d'une transaction, l'état peut être incohérent).
- Une transaction doit donc être exécutée "totalement" ou "pas du tout".

Begin transaction, Commit et Rollback

- `Begin transaction` commencer une nouvelle transaction.

Begin transaction, Commit et Rollback

- `Begin transaction` commencer une nouvelle transaction.
- `Commit` permet de valider et terminer la transaction courante.

Begin transaction, Commit et Rollback

- `Begin transaction` commencer une nouvelle transaction.
- `Commit` permet de valider et terminer la transaction courante.
- `Rollback` permet d'annuler la transaction courante (retour à l'état du `Begin transaction`).

Begin transaction, Commit et Rollback

- `Begin transaction` commencer une nouvelle transaction.
- `Commit` permet de valider et terminer la transaction courante.
- `Rollback` permet d'annuler la transaction courante (retour à l'état du `Begin transaction`).

Les commandes de définition de données provoquent une validation automatique de la transaction courante.

TP V

Clé étrangère

TP V

Clé étrangère

Clé étrangère

Dans le TP précédent, aucune vérification n'était faite par le système concernant les valeurs de l'attribut `etudiant` de la table `Note`, alors que seules des valeurs correspondant à des clés de `Etudiant` auraient du être acceptées.

Clé étrangère

Dans le TP précédent, aucune vérification n'était faite par le système concernant les valeurs de l'attribut `etudiant` de la table `Note`, alors que seules des valeurs correspondant à des clés de `Etudiant` auraient du être acceptées.
De même, la suppression d'un `Etudiant` n'entraînait pas la suppression de ses `Notes`. Il est possible de représenter ce type de contrainte de *clé étrangère* en SQL (un ou des attributs d'une table correspondent à une clé d'une autre table).

Clé étrangère : Syntaxe

Syntaxe

```
CONSTRAINT <nom>
 FOREIGN KEY (<col_1>, <col_2>, ...)
 REFERENCES <nomtableref> [(<colr_1>, <colr_2>, ...)]
 [ON UPDATE|DELETE  CASCADE|NO ACTION|SET NULL|SET DEFAULT]
```

Clé étrangère : Syntaxe

Syntaxe

```
CONSTRAINT <nom>
 FOREIGN KEY (<col_1>, <col_2>, ...)
 REFERENCES <nomtableref> [(<colr_1>, <colr_2>, ...)]
 [ON UPDATE|DELETE  CASCADE|NO ACTION|SET NULL|SET DEFAULT]
```

Dans le cas d'une contrainte portant sur une colonne, la ligne `FOREIGN KEY` ne doit pas être donnée : la contrainte porte sur la colonne dans laquelle elle est déclarée.

Clé étrangère

- Une contrainte de clé étrangère signifie que les colonnes sur lesquelles porte la contrainte (les `col_`i ou la colonne de la contrainte) font référence à une clé de `nomtableref` formée par les colonnes `colr_`i.

Clé étrangère

- Une contrainte de clé étrangère signifie que les colonnes sur lesquelles porte la contrainte (les `col_`i ou la colonne de la contrainte) font référence à une clé de `nomtableref` formée par les colonnes `colr_`i.
- Les colonnes `colr_`i doivent être `PRIMARY KEY` ou `UNIQUE`.

Clé étrangère

- Une contrainte de clé étrangère signifie que les colonnes sur lesquelles porte la contrainte (les `col_`i ou la colonne de la contrainte) font référence à une clé de `nomtableref` formée par les colonnes `colr_`i.
- Les colonnes `colr_`i doivent être `PRIMARY KEY` ou `UNIQUE`.
- Dans le cas où la clé primaire de `nomtableref` doit être référencée, il est équivalent d'écrire les colonnes de cette clé primaire ou aucune colonne `colr_`i.

Clé étrangère : Exemple

Exemple

```
create table Etudiant(
 ine numeric(10) constraint ekey primary key, ...
 cursus varchar(15) constraint efk references Cursus);
```

Clé étrangère : Exemple

Exemple

```
create table Etudiant(
 ine numeric(10) constraint ekey primary key, ...
 cursus varchar(15) constraint efk references Cursus);


create table Note(
 etudiant numeric(10), ...
 constraint nkey primary key (etudiant, matiere),
 constraint nfk foreign key (etudiant)
  references Etudiant);
```

Clé étrangère

Choix du comportement en cas de non respect de la contrainte

La contrainte n'est plus respectée dans le cas d'une mise à jour de la table référencée :

- Modification de données (`update` sur la clé référencée)
- Suppression de données (`delete` sur une ligne référencée)

Clé étrangère

Choix du comportement en cas de non respect de la contrainte

La contrainte n'est plus respectée dans le cas d'une mise à jour de la table référencée :

- Modification de données (`update` sur la clé référencée)
- Suppression de données (`delete` sur une ligne référencée)

Par défaut, le SGBD refuse de telles modifications. Mais on peut choisir le comportement du SGBD en cas d'erreur lors d'un `update` ou d'un `delete`

- `on delete ...`
- `on update ...`

Clé étrangère

- `NO ACTION` : Ne fait rien (interdit la mise à jour)

Clé étrangère

- `NO ACTION` : Ne fait rien (interdit la mise à jour)
- `CASCADE` : Suppression on modification en cascade dans la table qui référence la table modifiée.

Clé étrangère

- `NO ACTION` : Ne fait rien (interdit la mise à jour)
- `CASCADE` : Suppression on modification en cascade dans la table qui référence la table modifiée.
- `SET NULL` : Utilisation de la valeur nulle dans la table qui référence la table modifiée.

Clé étrangère

- `NO ACTION` : Ne fait rien (interdit la mise à jour)
- `CASCADE` : Suppression on modification en cascade dans la table qui référence la table modifiée.
- `SET NULL` : Utilisation de la valeur nulle dans la table qui référence la table modifiée.
- `SET DEFAULT` : Utilisation de la valeur par défaut.

Clé étrangère

- `NO ACTION` : Ne fait rien (interdit la mise à jour)
- `CASCADE` : Suppression on modification en cascade dans la table qui référence la table modifiée.
- `SET NULL` : Utilisation de la valeur nulle dans la table qui référence la table modifiée.
- `SET DEFAULT` : Utilisation de la valeur par défaut.

On utilise souvent `on delete cascade`.

TP V

Clé étrangère

TP V

Clé étrangère

Le type boolean

Peut prendre deux valeurs `true` ou `false`

Le type boolean

Peut prendre deux valeurs `true` ou `false`

- `'t' 'true' 'y' 'yes' '1'` = `true`
- `'f' 'false' 'n' 'no' '0'` = `false`

TP V

Clé étrangère

Fonction conditionnelle

```
CASE WHEN condition THEN result
  [WHEN ...]
  [ELSE result]
END
```

Fonction conditionnelle

```
CASE WHEN condition THEN result
  [WHEN ...]
  [ELSE result]
END
```

Retourne le `result` de la `condition` vraie (ou le `result` du `else` si aucune condition n'est vraie).

TP V

Clé étrangère

Fonction de formatage de données

L'affichage par défaut de réels n'est pas toujours satisfaisant. On peut demander le formatage de données (entiers, réels) à l'aide d'une chaine de format avec la fonction `to_char(donnée, format)`.

Fonction de formatage de données

L'affichage par défaut de réels n'est pas toujours satisfaisant. On peut demander le formatage de données (entiers, réels) à l'aide d'une chaine de format avec la fonction `to_char(donnée, format)`.

- `9` : chiffre
- `0` : chiffre (ou 0 pour remplir)
- `.` : séparateur décimales
- ... (voir documentation)

Fonction de formatage de données

L'affichage par défaut de réels n'est pas toujours satisfaisant. On peut demander le formatage de données (entiers, réels) à l'aide d'une chaine de format avec la fonction `to_char(donnée, format)`.

- `9` : chiffre
- `0` : chiffre (ou 0 pour remplir)
- `.` : séparateur décimales
- ... (voir documentation)

Exemple

`to_char(485, 'Valeur 00 0 0')` → `Valeur 04 8 5`

TP VI

Oracle, Vues

TP VI

Oracle, Vues

Oracle

Oracle est un SGBD relationnel propriétaire.

Oracle

Oracle est un SGBD relationnel propriétaire.
Très performant sur de grandes bases de données. . .
. . . mais pas gratuit (de 4000 euros à 30000 euros).
Version gratuite limitée.

Connexion à un serveur Oracle

Au département informatique

Avant de lancer l'interpréteur, il est nécessaire que certaines variables d'environnement soient définies. (sur forge)

```
. oracle_env.sh
```

Connexion à un serveur Oracle

Au département informatique

Avant de lancer l'interpréteur, il est nécessaire que certaines variables d'environnement soient définies. (sur forge)

```
. oracle_env.sh
```

Le client fourni "simple" fourni avec Oracle est `sqlplus`, mais il en existe d'autres comme `yasql` ou `tora` (disponibles sur forge).

```
sqlplus nomdelogin@forge.info-ua
```

sqlplus

- obtenir de l'aide : `help`
- obtenir de l'aide sur les commandes SQL, par exemple : `help select`

sqlplus

- obtenir de l'aide : `help`
- obtenir de l'aide sur les commandes SQL, par exemple : `help select`
- interpréter les commandes stockées dans un fichier : `start`
- sauver les commandes saisies depuis le début de la session : `save`

sqlplus

- obtenir de l'aide : `help`
- obtenir de l'aide sur les commandes SQL, par exemple : `help select`
- interpréter les commandes stockées dans un fichier : `start`
- sauver les commandes saisies depuis le début de la session : `save`
- quitter l'interpréteur : `exit`
- . . . Consulter l'aide en ligne.

sqlplus

- obtenir de l'aide : `help`
- obtenir de l'aide sur les commandes SQL, par exemple : `help select`
- interpréter les commandes stockées dans un fichier : `start`
- sauver les commandes saisies depuis le début de la session : `save`
- quitter l'interpréteur : `exit`
- . . . Consulter l'aide en ligne.

Obtenir la liste des tables de l'utilisateur

```
select table_name from user_tables;
```

TP VI

Oracle, Vues

Le mécanisme des vues

Une *vue* est une "relation dynamique" (calculée dynamiquement, non stockée).

Le mécanisme des vues

Une *vue* est une "relation dynamique" (calculée dynamiquement, non stockée).
Une vue est obtenue par une requête : elle peut montrer une partie de d'une table ou certaines données issues de plusieurs tables.

Le mécanisme des vues

Une *vue* est une "relation dynamique" (calculée dynamiquement, non stockée).
Une vue est obtenue par une requête : elle peut montrer une partie de d'une table ou certaines données issues de plusieurs tables.
Il est possible d'utiliser une vue comme une table (à certaines restrictions près).

TP VI

Oracle, Vues

Création et destruction

Syntaxe (Création)

```
CREATE VIEW nomvue [ (nomcol1, ...) ]
AS SELECT ... ;
```

Création et destruction

Syntaxe (Création)

```
CREATE VIEW nomvue [ (nomcol1, ...) ]
AS SELECT ... ;
```

- *nomcol1, ...* : obligatoire uniquement dans le cas où certaines colonnes du `select` ne sont pas nommées (fonctions).

Création et destruction

Syntaxe (Création)

```
CREATE VIEW nomvue [ (nomcol1, ...) ]
AS SELECT ... ;
```

- *nomcol1, ...* : obligatoire uniquement dans le cas où certaines colonnes du `select` ne sont pas nommées (fonctions).
- La clause `select` suit la syntaxe classique d'une interrogation et peut faire intervenir plusieurs tables.

Création et destruction

Syntaxe (Création)

```
CREATE VIEW nomvue [ (nomcol1, ...) ]
AS SELECT ... ;
```

- *nomcol1, ...* : obligatoire uniquement dans le cas où certaines colonnes du `select` ne sont pas nommées (fonctions).
- La clause `select` suit la syntaxe classique d'une interrogation et peut faire intervenir plusieurs tables.
- La clause `order by` est interdite.

Syntaxe (Destruction)

```
DROP VIEW nomvue;
```

Création

Exemple

```
create view LogicielsSalle100
as select nom
   from Logiciel where numero=100;
```

Création

Exemple

```
create view LogicielsSalle100
as select nom
   from Logiciel where numero=100;
create view TableauNoteMajore
as select nom, prenom, matiere, note,
          note+1 as notemajoree
   from Etudiant, Note
   where Etudiant.ine = Note.etudiant;
```

TP VI

Oracle, Vues

Utilisation

Dans une interrogation

Une vue peut être utilisée dans un `select` comme s'il s'agissait d'une table.

Exemple

```
select nom
from LogicielsSalle100;
```

Utilisation

Dans une interrogation

Une vue peut être utilisée dans un `select` comme s'il s'agissait d'une table.

Exemple

```
select nom
from LogicielsSalle100;
```

Si les données des tables ayant servi à la définition de la vue (`Logiciel`) sont mises à jour, la vue est automatiquement mise à jour.

Utilisation

Dans une mise à jour (`insert`, `update`)

Des mises à jour sont possibles sur une vue (et modifient les tables ayant servi à la définition de la vue) si ces conditions sont vraies :

Utilisation

Dans une mise à jour (`insert`, `update`)

Des mises à jour sont possibles sur une vue (et modifient les tables ayant servi à la définition de la vue) si ces conditions sont vraies :

- La définition de la vue ne doit pas comporter de (θ-)jointure.

Utilisation

Dans une mise à jour (`insert`, `update`)

Des mises à jour sont possibles sur une vue (et modifient les tables ayant servi à la définition de la vue) si ces conditions sont vraies :

- La définition de la vue ne doit pas comporter de (θ-)jointure.
- Les colonnes de la vue doivent être des colonnes des tables (et non des expressions).

Utilisation

Dans une mise à jour (`insert`, `update`)

Des mises à jour sont possibles sur une vue (et modifient les tables ayant servi à la définition de la vue) si ces conditions sont vraies :

- La définition de la vue ne doit pas comporter de (θ-)jointure.
- Les colonnes de la vue doivent être des colonnes des tables (et non des expressions).

Exemple

Il est possible de modifier `LogicielsSalle100` mais pas `TableauNoteMajore`.

Utilisation

Exemple

```
update LogicielsSalle100
set nom='Oracle 9' where nom='Oracle 8';
```

Utilisation

Exemple

```
update LogicielsSalle100
set nom='Oracle 9' where nom='Oracle 8';
insert into LogicielsSalle100 values ('Eclipse');
```

Utilisation

Exemple

```
update LogicielsSalle100
set nom='Oracle 9' where nom='Oracle 8';
insert into LogicielsSalle100 values ('Eclipse');
```

Attention

Si certaines colonnes (des tables ayant servi à la définition) ne sont pas présentes dans la vue, lors de l'insertion, ces colonnes recevront la valeur `null`.

L'insertion ci-dessus ajoute une ligne dans `Logiciel` qui n'est pas dans `LogicielsSalle100`.

Utilisation

La clause `with check option`

. . . interdit les mises à jour (`update`, `insert`) produisant des lignes qui ne sont pas visibles dans la vue.

Utilisation

La clause `with check option`

. . . interdit les mises à jour (`update`, `insert`) produisant des lignes qui ne sont pas visibles dans la vue.

Exemple

```
create view LogicielsSalle100
as select nom
   from Logiciel where numero=100
with check option;
```

L'insertion précédente est refusée dans cette vue.

TP VI

Oracle, Vues

Usages

- Définition d'un schéma externe : partie du monde ("sous-monde") visible par un utilisateur de la base. Favorise l'indépendance données/programmes : le schéma conceptuel peut changer sans que les vues ne changent.

Usages

- Définition d'un schéma externe : partie du monde ("sous-monde") visible par un utilisateur de la base. Favorise l'indépendance données/programmes : le schéma conceptuel peut changer sans que les vues ne changent.
- Définition des droits d'accès sur des vues : permet une expression des droits d'accès plus fine que les droits sur les tables.

Usages

- Définition d'un schéma externe : partie du monde ("sous-monde") visible par un utilisateur de la base. Favorise l'indépendance données/programmes : le schéma conceptuel peut changer sans que les vues ne changent.
- Définition des droits d'accès sur des vues : permet une expression des droits d'accès plus fine que les droits sur les tables.
- Simplification de requêtes : utilisation de vues comme "tables intermédiaires".

TP VI

Oracle, Vues

TP VI

Oracle, Vues

Séquences

Il n'est pas possible de définir des champs auto incrémentés en Oracle. On utilise à la place une séquence.

Séquences

Il n'est pas possible de définir des champs auto incrémentés en Oracle. On utilise à la place une séquence.
Une *séquence* permet de représenter un suite d'entiers, par exemple pour générer des valeurs uniques de clés.

Syntaxe

```
create sequence <nomseq> [start with <val>] [increment by <val>];
```

Séquences

Il n'est pas possible de définir des champs auto incrémentés en Oracle. On utilise à la place une séquence.
Une *séquence* permet de représenter un suite d'entiers, par exemple pour générer des valeurs uniques de clés.

Syntaxe

```
create sequence <nomseq> [start with <val>] [increment by <val>];
```

Une séquence s'utilise à travers deux "propriétés" : `currval` (qui permet d'accéder à la valeur courante) et `nextval` (qui passe à la valeur suivante et permet d'accéder à cette nouvelle valeur).

```
insert into <nomtable>
values (... <nomseq>.nextval ...);
```

Séquences

- La requête suivante permet d'obtenir la valeur courante :
  `select <nomseq>.currval from dual;`

Séquences

- La requête suivante permet d'obtenir la valeur courante :
  `select <nomseq>.currval from dual;`
- Une séquence peut être détruite par
  `drop sequence <nomseq>;`

Séquences

- La requête suivante permet d'obtenir la valeur courante :
  `select <nomseq>.currval from dual;`
- Une séquence peut être détruite par
  `drop sequence <nomseq>;`
- Les séquences de l'utilisateur peuvent être obtenues par
  `select sequence_name from user_sequences;`

TP VI

Oracle, Vues

Fonction conditionnelle

```
decode (expression,
  valeur1 , result1
  [, valeuri , resulti,]
  ...
  [resultdefaut])
```

Fonction conditionnelle

Exemple

```
SELECT nom,
  decode (niveau,
    1,'debutant',
    2,'moyen',
    3,'confirme',
    'inconnu')
FROM inscriptions;
```

TP VI

Oracle, Vues

Opérateurs de tables

Certains opérateurs SQL permettent d'exécuter des opérations ensemblistes entre tables (ou vues, ou résultats de requêtes). Ces opérateurs ne peuvent s'appliquer que sur des tables ayant les mêmes colonnes.

Opérateurs de tables

Certains opérateurs SQL permettent d'exécuter des opérations ensemblistes entre tables (ou vues, ou résultats de requêtes). Ces opérateurs ne peuvent s'appliquer que sur des tables ayant les mêmes colonnes.

- `union` fusionne deux tables.

Exemple

```
select * from Etudiant
 where cursus = 'Licence pro'
union
select * from Etudiant
 where cursus = 'Maitrise info';
```

Opérateurs de tables

Certains opérateurs SQL permettent d'exécuter des opérations ensemblistes entre tables (ou vues, ou résultats de requêtes). Ces opérateurs ne peuvent s'appliquer que sur des tables ayant les mêmes colonnes.

- `union` fusionne deux tables.

Exemple

```
select * from Etudiant
 where cursus = 'Licence pro'
union
select * from Etudiant
 where cursus = 'Maitrise info';
```

- `intersect` calcule l'intersection entre deux tables.

Opérateurs de tables

Certains opérateurs SQL permettent d'exécuter des opérations ensemblistes entre tables (ou vues, ou résultats de requêtes). Ces opérateurs ne peuvent s'appliquer que sur des tables ayant les mêmes colonnes.

- `union` fusionne deux tables.

Exemple

```
select * from Etudiant
 where cursus = 'Licence pro'
union
select * from Etudiant
 where cursus = 'Maitrise info';
```

- `intersect` calcule l'intersection entre deux tables.
- `minus` calcule les lignes présentes dans la première table qui ne sont pas dans la seconde. (`except` avec PostgreSQL)

TP VI

Oracle, Vues

Transactions

Dans Oracle, tout ordre est saisi dans une transaction.

- Il n'y a pas de `begin transaction`.
- Une transaction est démarrée au début d'une session.

Transactions

Dans Oracle, tout ordre est saisi dans une transaction.

- Il n'y a pas de `begin transaction`.
- Une transaction est démarrée au début d'une session.
- Une transaction est validée (`commit`) à la fin d'une session.
- Un `commit` ouvre une nouvelle transaction.

Transactions

Dans Oracle, tout ordre est saisi dans une transaction.

- Il n'y a pas de `begin transaction`.
- Une transaction est démarrée au début d'une session.
- Une transaction est validée (`commit`) à la fin d'une session.
- Un `commit` ouvre une nouvelle transaction.

Les modifications effectuées par une transaction ne sont visibles des autres transactions qu'au moment du `commit`.

TP VI

Oracle, Vues

Contraintes

Une contrainte ne peut pas être modifiée, mais on peut détruire une contrainte et en ajouter une autre à une table.

Contraintes

Une contrainte ne peut pas être modifiée, mais on peut détruire une contrainte et en ajouter une autre à une table.

Syntaxe

```
ALTER TABLE <nomtable> DROP CONSTRAINT
<nomcontrainte>;
ALTER TABLE <nomtable> ADD CONSTRAINT <nomcontrainte>
<defcontrainte>;
```