INSTITUT D'ENSEIGNEMENT TECHNIQUE
DE MECANIQUE ET D'ELECTRICITE
MARGUERITE MASSART

BASES
DE
DONNEES

Professeur : BANZIRA Pascal

1ère PARTIE :

Introduction à

MySQL / PhPMyAdmin

Introduction

- MySQL dérive directement de SQL (Structured Query Language)
- L'outil phpMyAdmin est développé en PHP et offre une interface pour l'administration des base de données
- phpMyAdmin est téléchargeable ici : http://phpmyadmin.sourceforge.net
- cet outil permet de :
  - créer de nouvelles bases
  - créer/modifier/supprimer des tables
  - afficher/ajouter/modifier/supprimer des tuples dans des tables
  - effectuer des sauvegardes de la structure et/ou des donnés
  - effectuer des requêtes
  - gérer les privileges des utilisateurs

Liens intéressants (MySQL)

- La rérérence MySQL (anglais): `http://www.mysql.org`
- Le manuel MySQL : `http://dev.mysql.com/doc/refman/5.7/en`
- Des cours et articles intéressants :
  - `http://www.developpez.com`
  - `http://cyberzoide.developpez.com/php4/mysql/`

Types des attributs MySQL

- Nombre entier signé ou non
- Nombre à virgule
- Chaîne de caractères
- Date et heure
- Enumeration
- Ensemble

Entiers

nom	borne inférieure	borne supérieure
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT*	-2147483648	2147483647
INT* UNSIGNED	0	4294967295
BIGINT	-9223372036854775808	9223372036854775807
BIGINT UNSIGNED	0	18446744073709551615

(*) : **INTEGER** est un synonyme de **INT**.

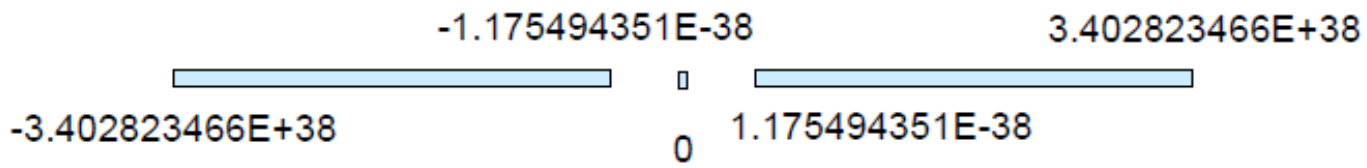
UNSIGNED permet d'avoir un type non signé.

ZEROFILL : remplissage des zéros non significatifs.

Flottants

Les flottants – dits aussi nombres réels – sont des nombres à virgule. Contrairement aux entiers, leur domaine n'est pas continu du fait de l'impossibilité de les représenter avec une précision absolue.

Exemple du type **FLOAT** :



nom	domaine négatif : borne inférieure borne supérieure	Domaine positif : borne inférieure borne supérieure
FLOAT	-3.402823466E+38 -1.175494351E-38	1.175494351E-38 3.402823466E+38
DOUBLE*	-1.7976931348623157E+308 -2.2250738585072014E-308	2.2250738585072014E-308 1.7976931348623157E+308

(*) : **REAL** est un synonyme de **DOUBLE**.

Chaînes

nom	longueur
CHAR(M)	Chaîne de taille fixée à M, où 1<M<255, complétée avec des espaces si nécessaire.
CHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
VARCHAR(M)	Chaîne de taille variable, de taille maximum M, où 1<M<255, complété avec des espaces si nécessaire.
VARCHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
TINYTEXT	Longueur maximale de 255 caractères.
TEXT	Longueur maximale de 65535 caractères.
MEDIUMTEXT	Longueur maximale de 16777215 caractères.
LONGTEXT	Longueur maximale de 4294967295 caractères.
DECIMAL(M,D)*	Simule un nombre flottant de D chiffres après la virgule et de M chiffres au maximum. Chaque chiffre ainsi que la virgule et le signe moins (pas le plus) occupe un caractère.

(*) : **NUMERIC** est un synonyme de **DECIMAL**.

Dates et heures

nom	description
DATE	Date au format anglophone AAAA-MM-JJ.
DATETIME	Date et heure au format anglophone AAAA-MM-JJ HH:MM:SS.
TIMESTAMP	Affiche la date et l'heure sans séparateur : AAAAMMJJHHMMSS.
TIMESTAMP(M)	Idem mais M vaut un entier pair entre 2 et 14. Affiche les M premiers caractères de **TIMESTAMP**.
TIME	Heure au format HH:MM:SS.
YEAR	Année au format AAAA.

nom	description
TIMESTAMP(2)	AA
TIMESTAMP(4)	AAMM
TIMESTAMP(6)	AAMMJJ
TIMESTAMP(8)	AAAAMMJJ
...	...

Si attribut de type TIMESTAMP

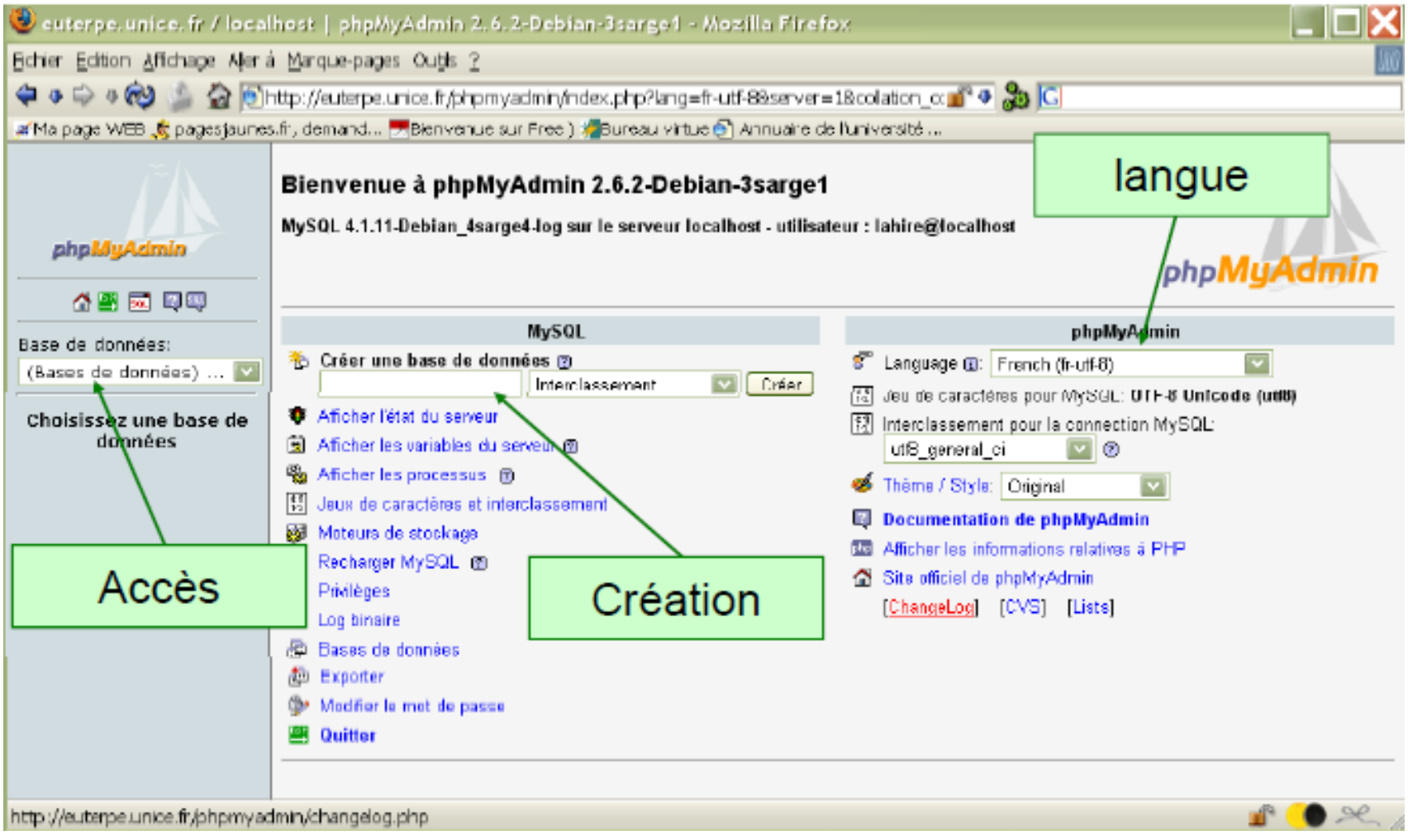
= *NULL* :

- prend la date et heure de l'insertion.
- suit le format indiqué

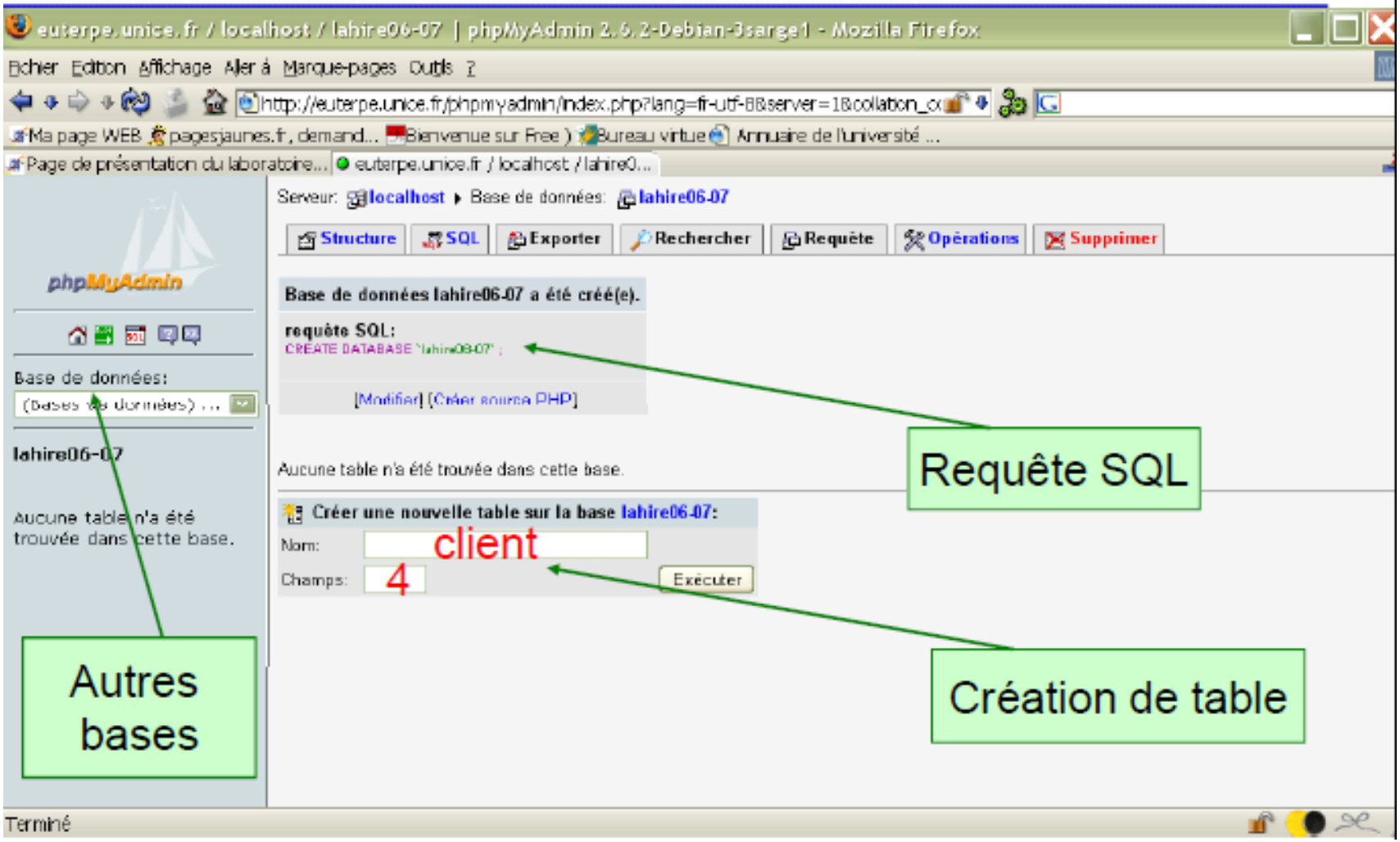
Interface graphique (phpMyAdmin)

- développé en PHP
- téléchargeable : `http://phpmyadmin.sourceforge.net`
- `http://euterpe.unice.fr/phpmyadmin`
- version installée (5.x)

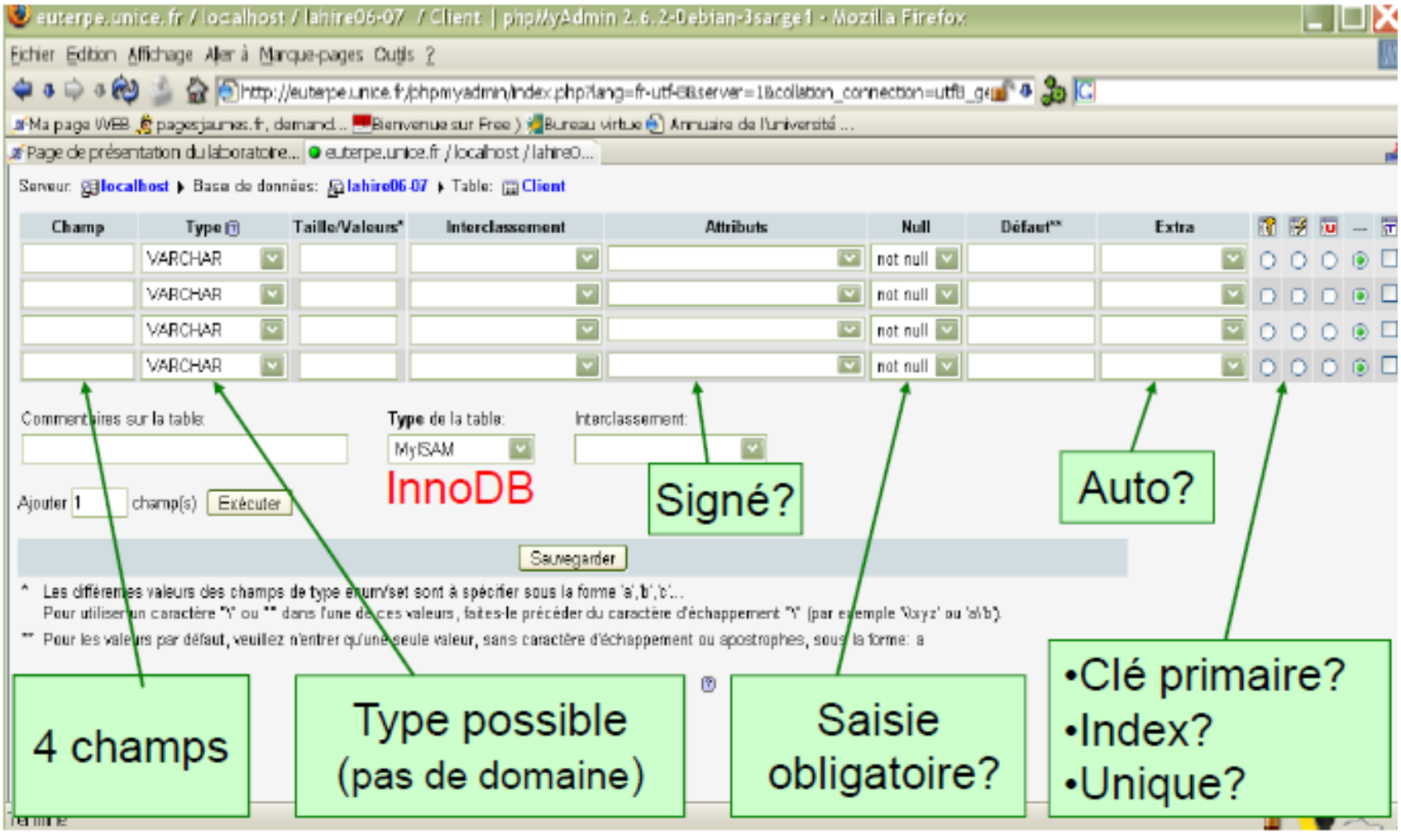
Interface graphique (phpMyAdmin) - 1



Interface graphique (phpMyAdmin) - 2



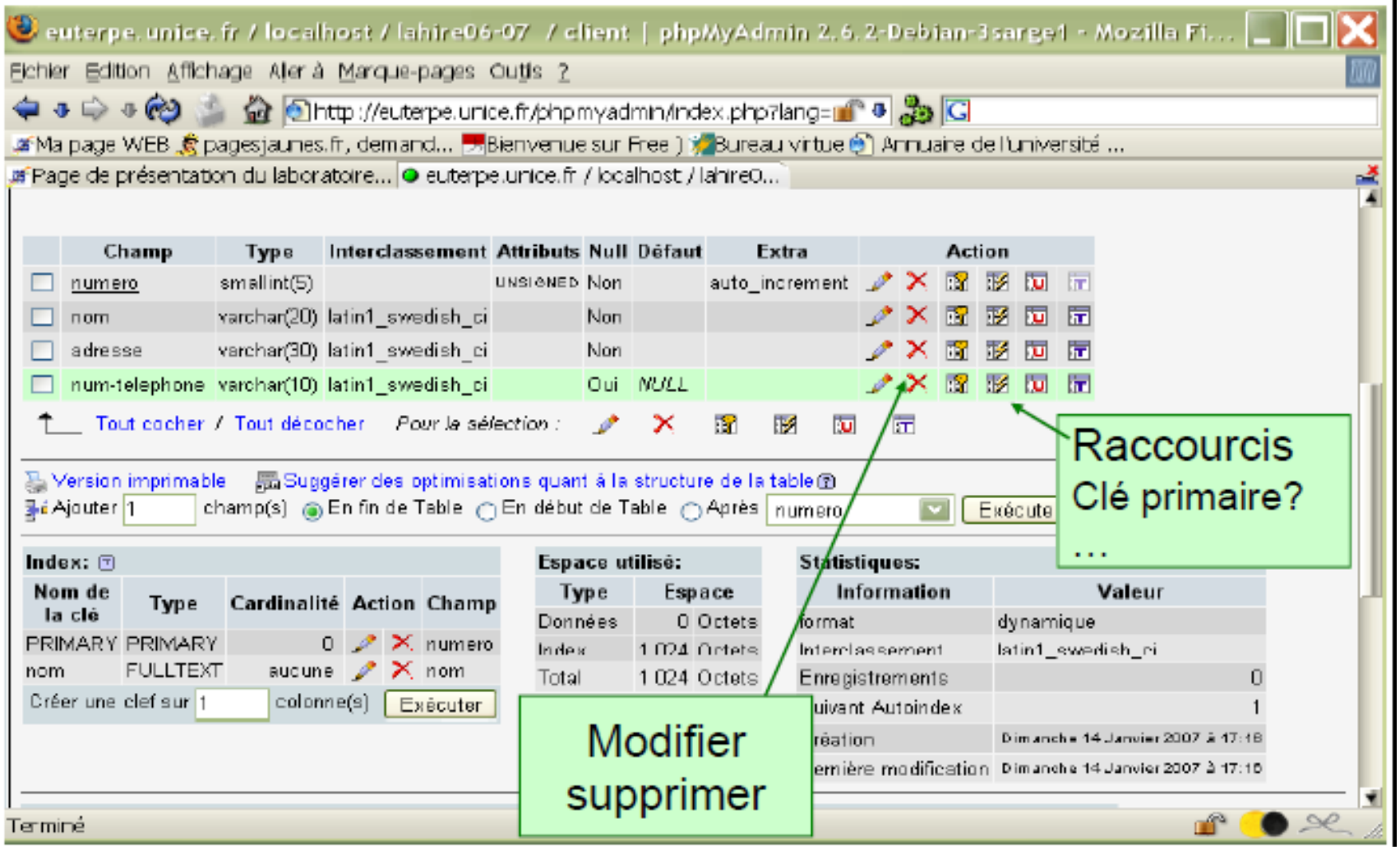
Interface graphique (phpMyAdmin) - 3



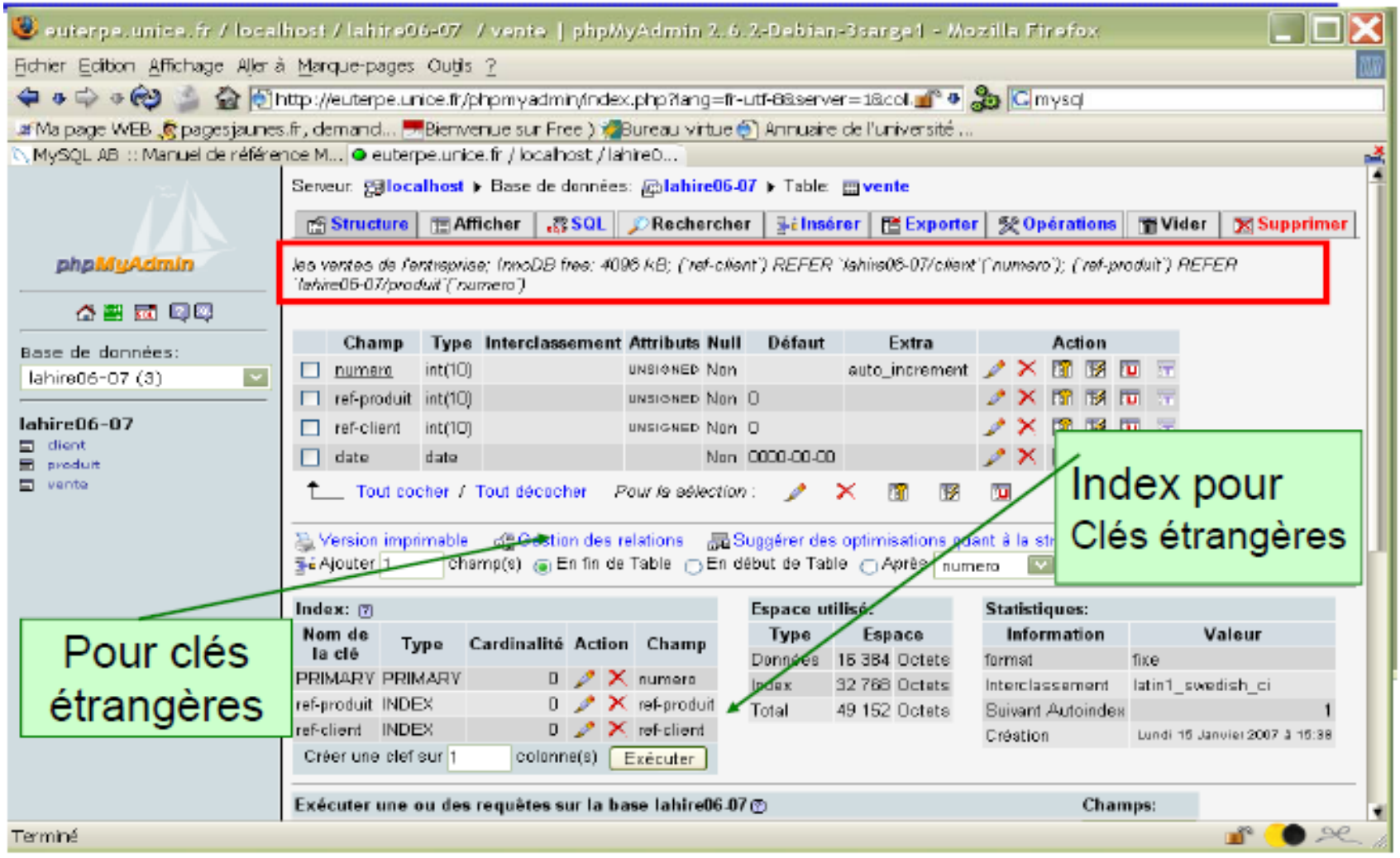
Quelques mots sur InnoDB

- Moteur de tables (licence GNU GLP)
- gestionnaire de tables transactionnelles (verrouillage de lignes)
- maximisation de performances (grans volumes de données)
- support clé étrangère
- stockage des tables et index : espace de tables (un ou plusieurs fichiers). MyISAM : un fichier par table

Interface graphique (phpMyAdmin) - 4



Interface graphique (phpMyAdmin) – 4 bis

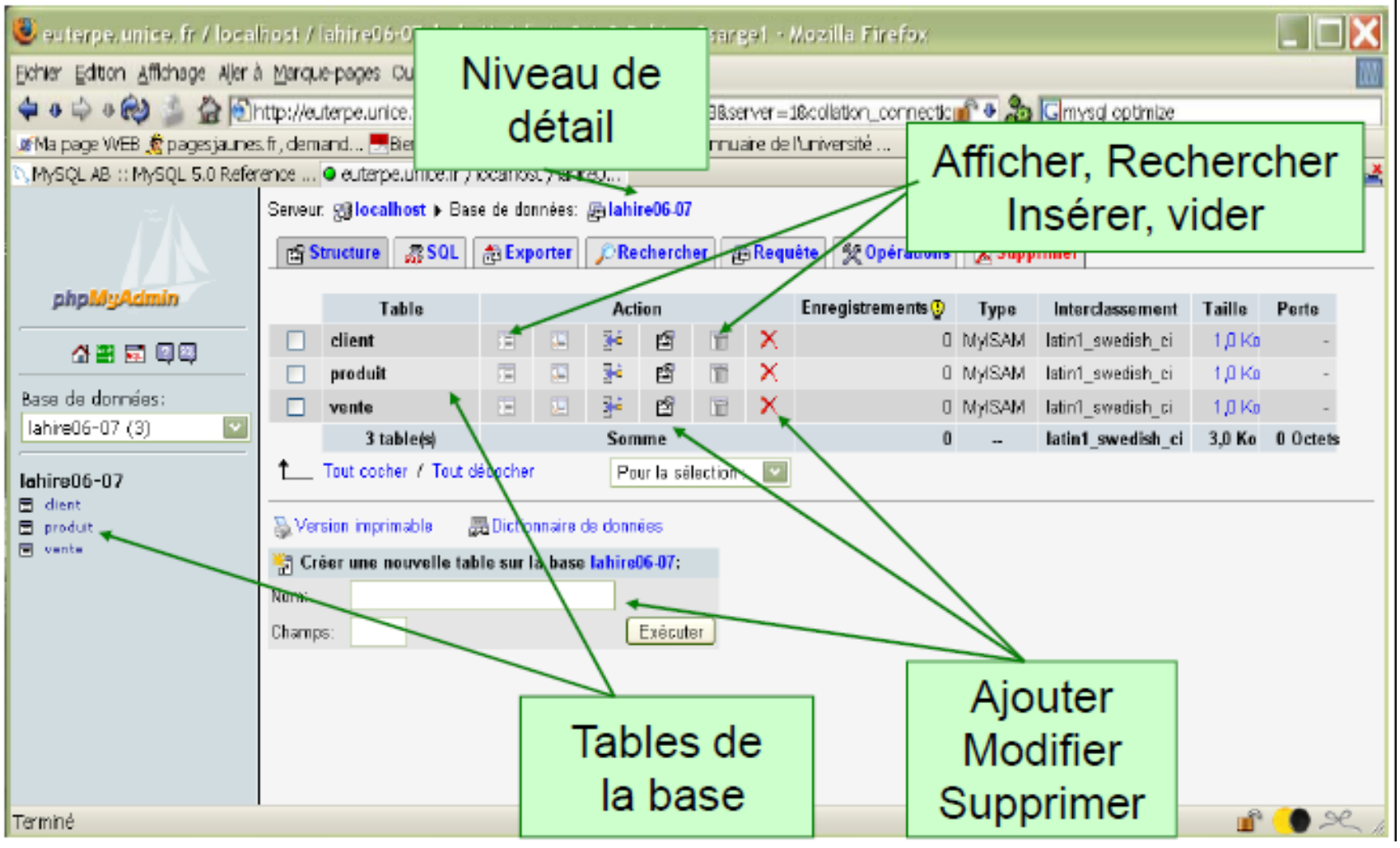


Intégrité référentielle

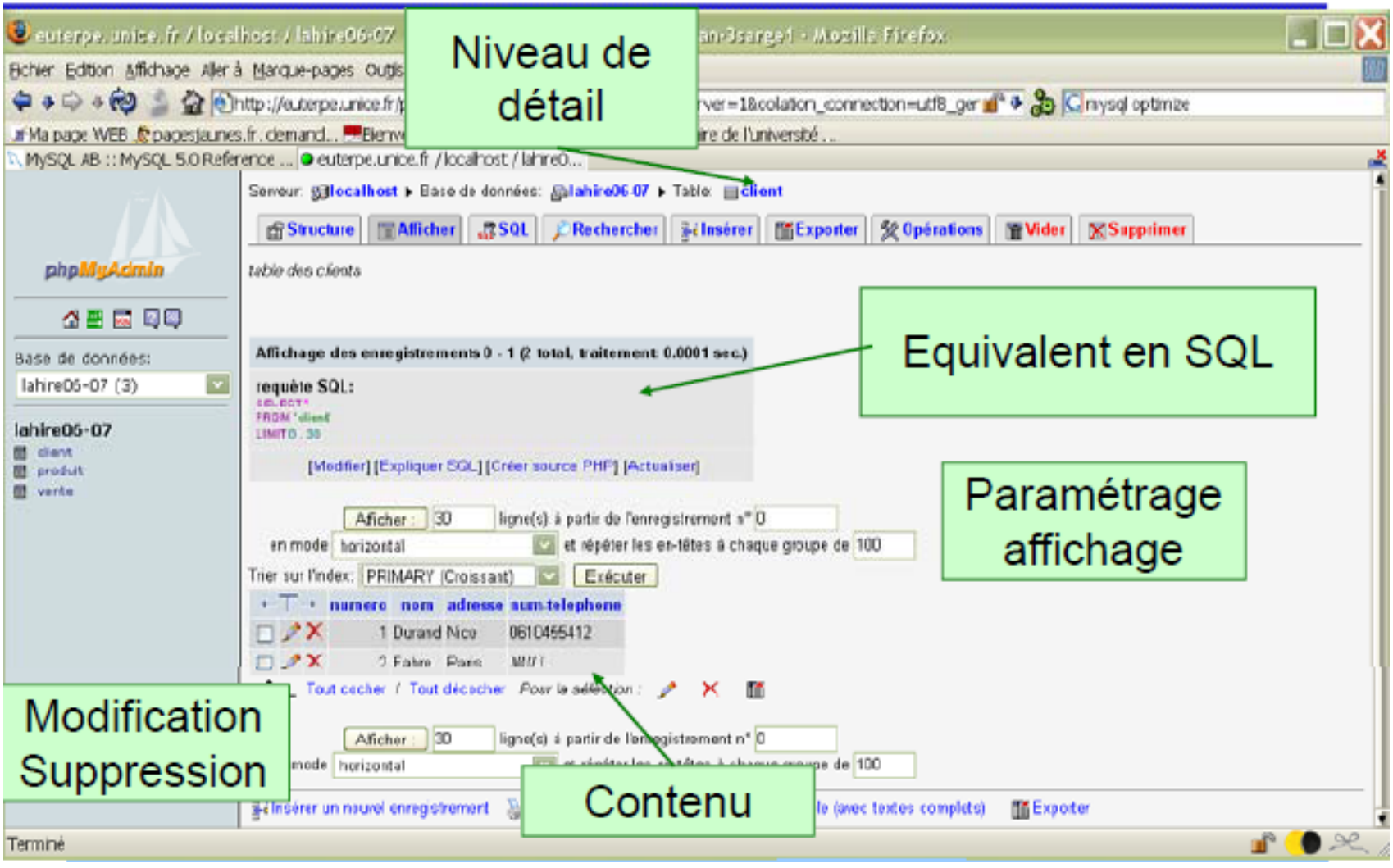
ON UPDATE et ON DELETE
Si l'utilisateur tente de supprimer une ligne d'une table parente, alors qu'une ou plusieurs lignes existent dans une table enfant correspondente à la première, il y a plusieurs possibilités :

- CASCADE efface la ligne de la table parente et supprime automatiquement les lignes correspondents dans la table enfante
- SET NULL supprime la ligne de la table parente et met la (ou les) valeur(s) de la clé étrangère à NULL
- SET DEFAULT supprime la ligne de la table parente et met tous les composants de la clé étrangère à leur valeur par défault dans la table enfante
- NO ACTION rejette l'opération de suppression dans la table parente

Interface graphique (phpMyAdmin) - 5

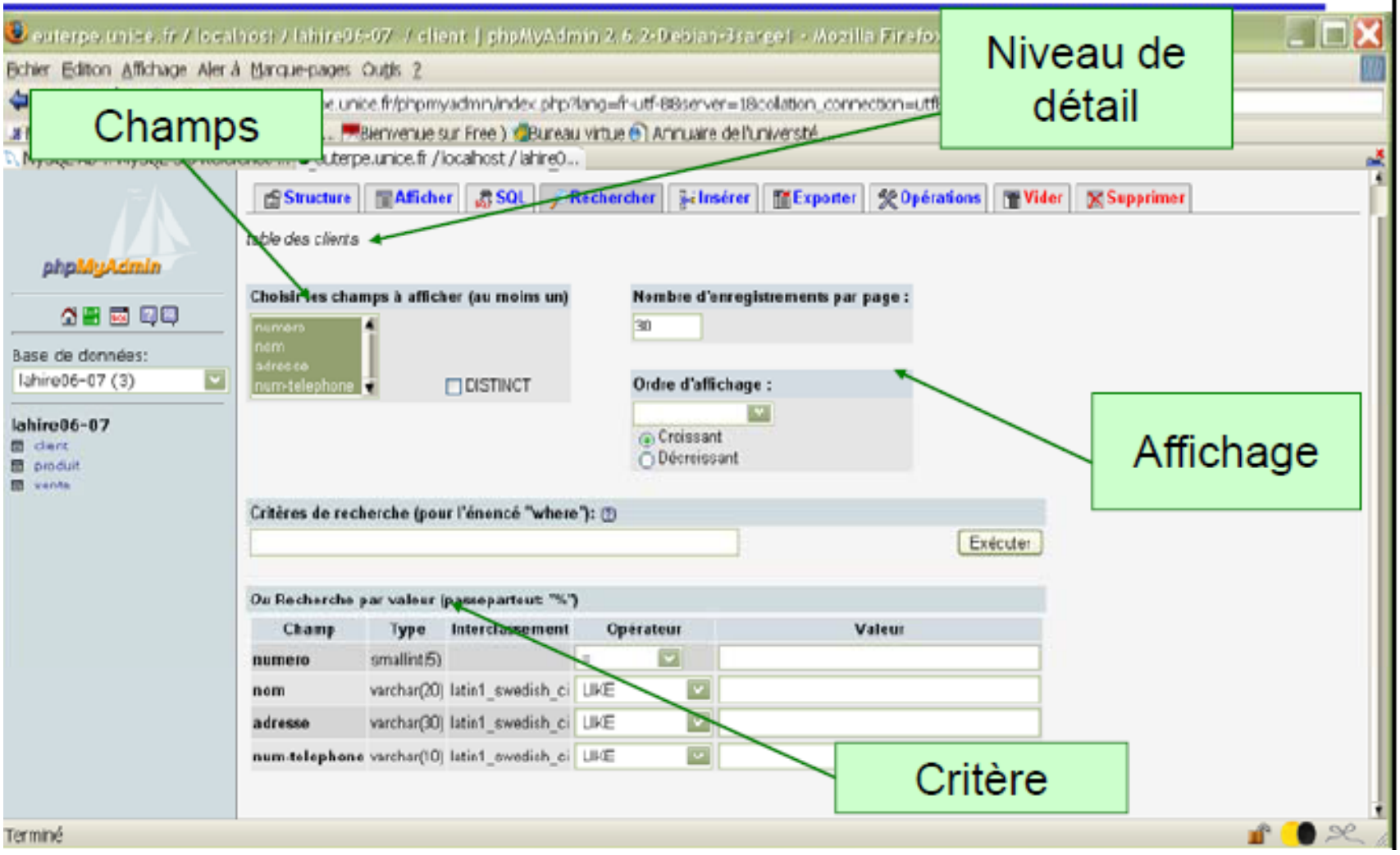


Interface graphique (phpMyAdmin) - 6

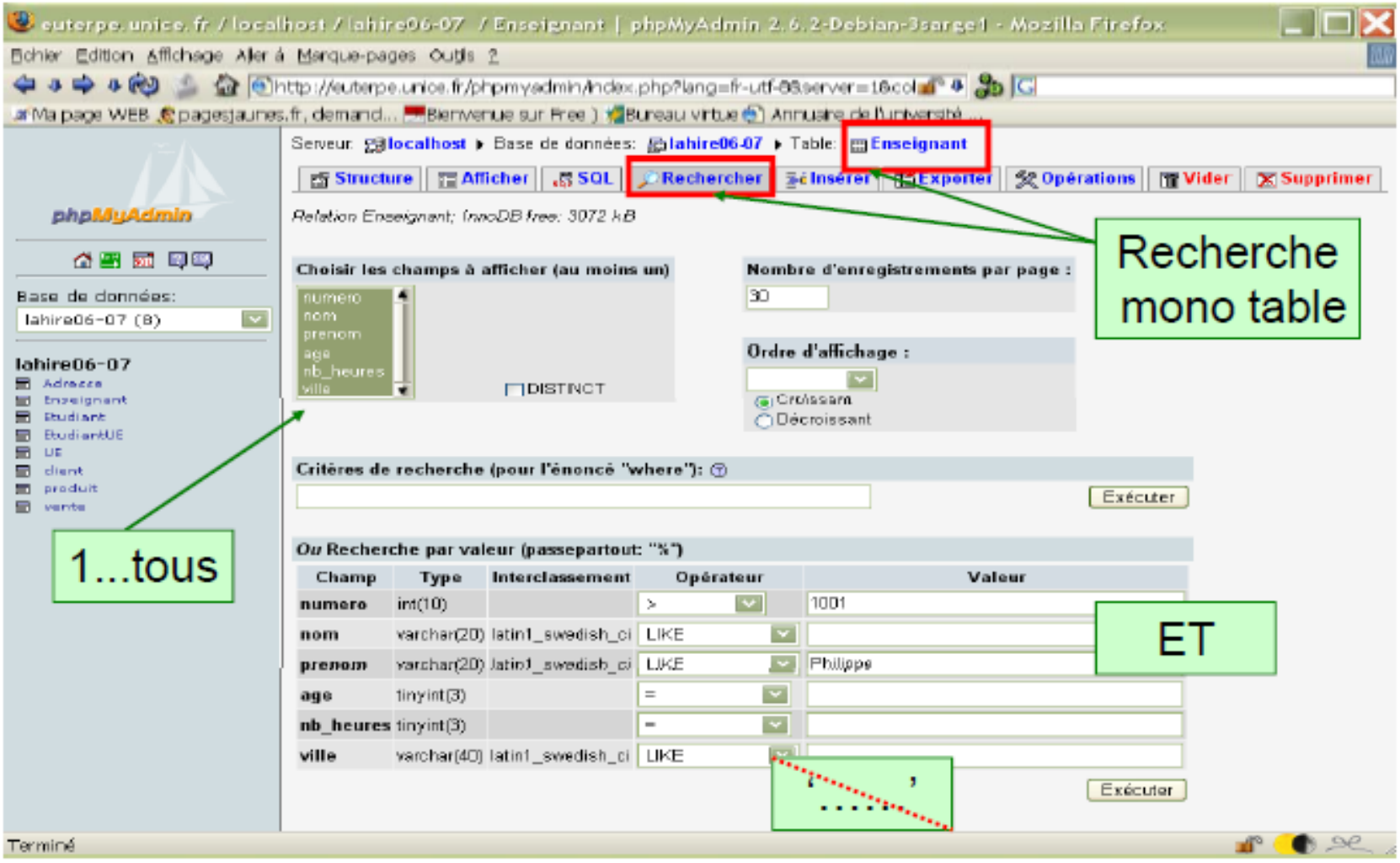


Interface graphique (phpMyAdmin) - 7

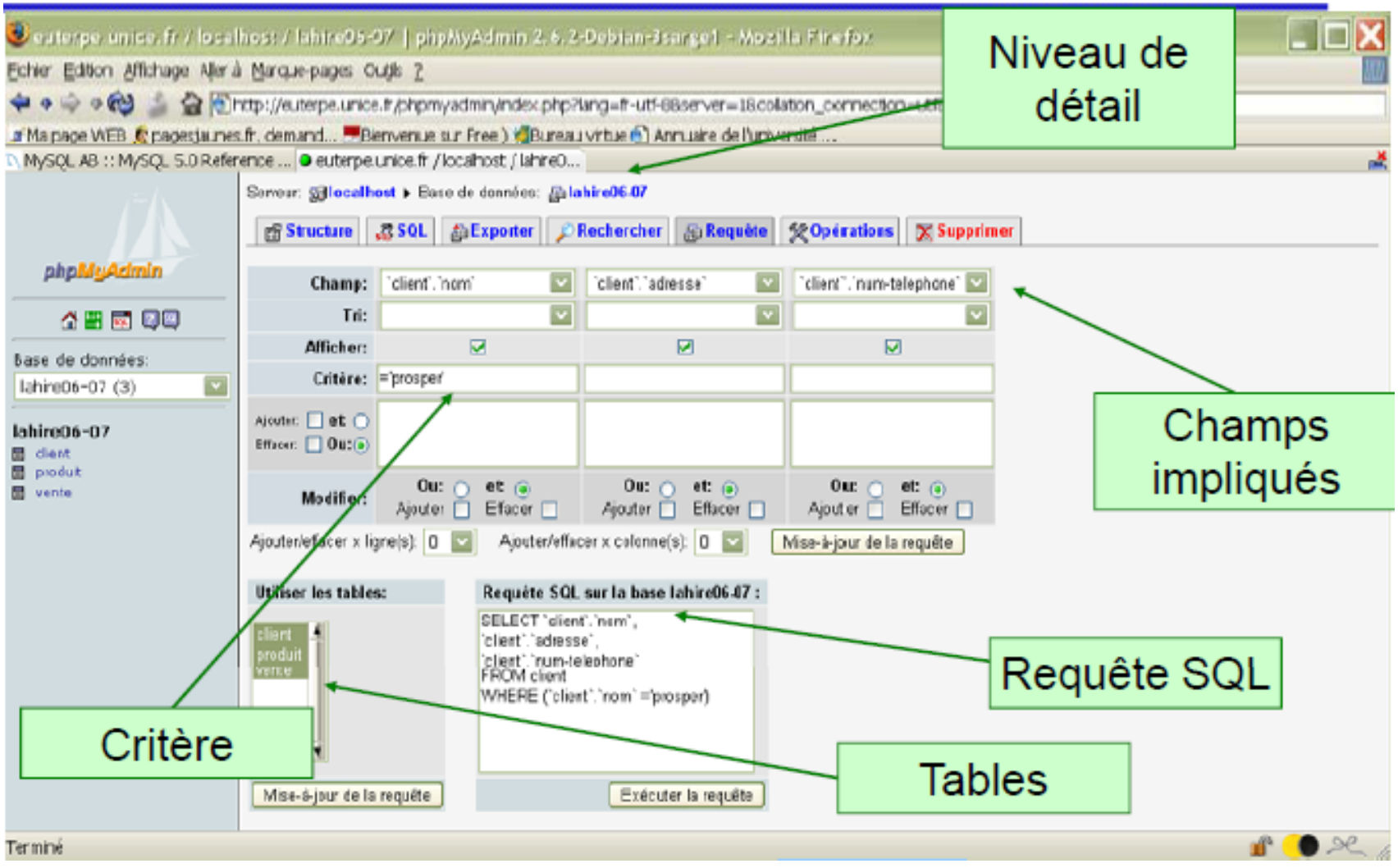
Interface graphique (phpMyAdmin) - 8



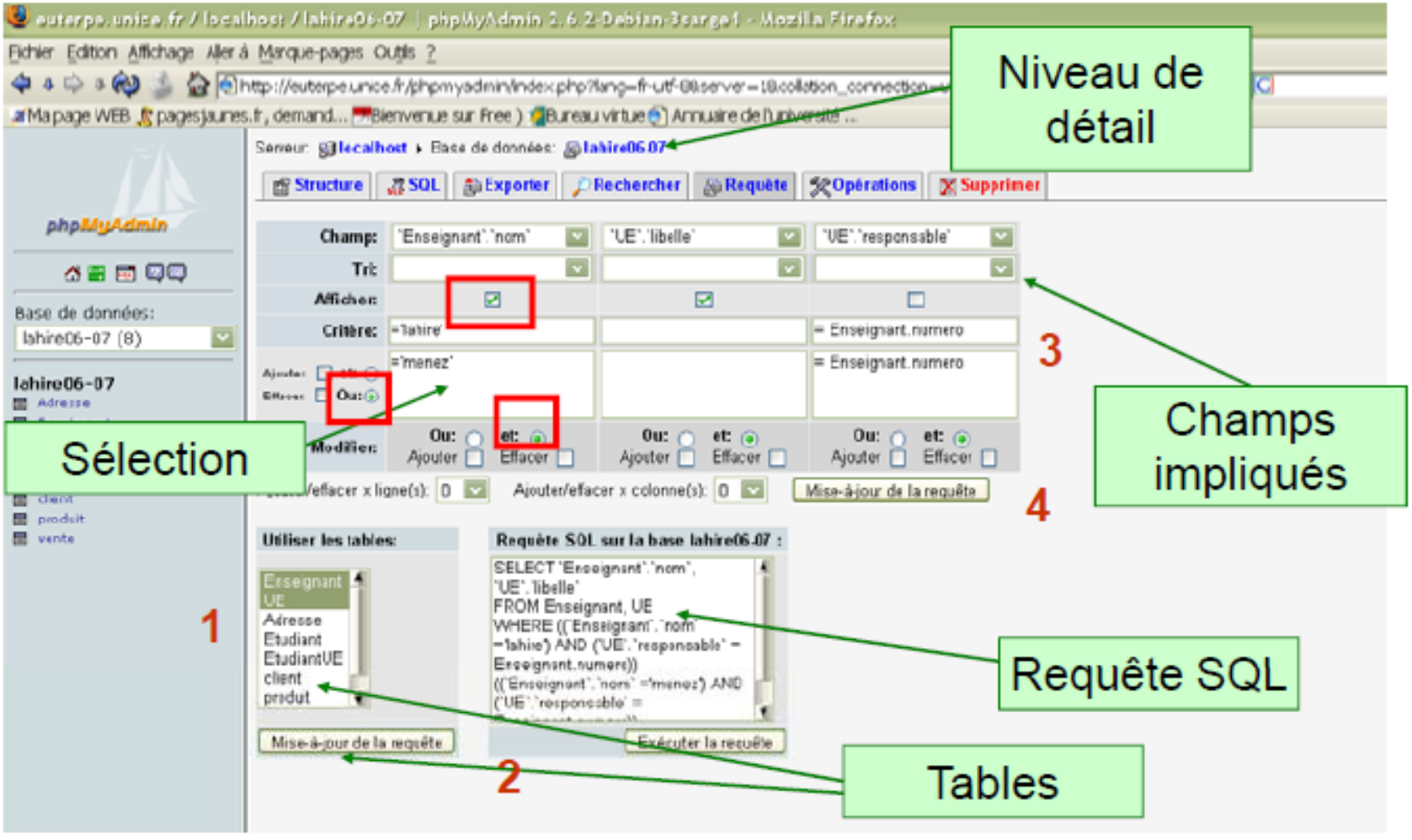
Interface graphique (phpMyAdmin) – 8b



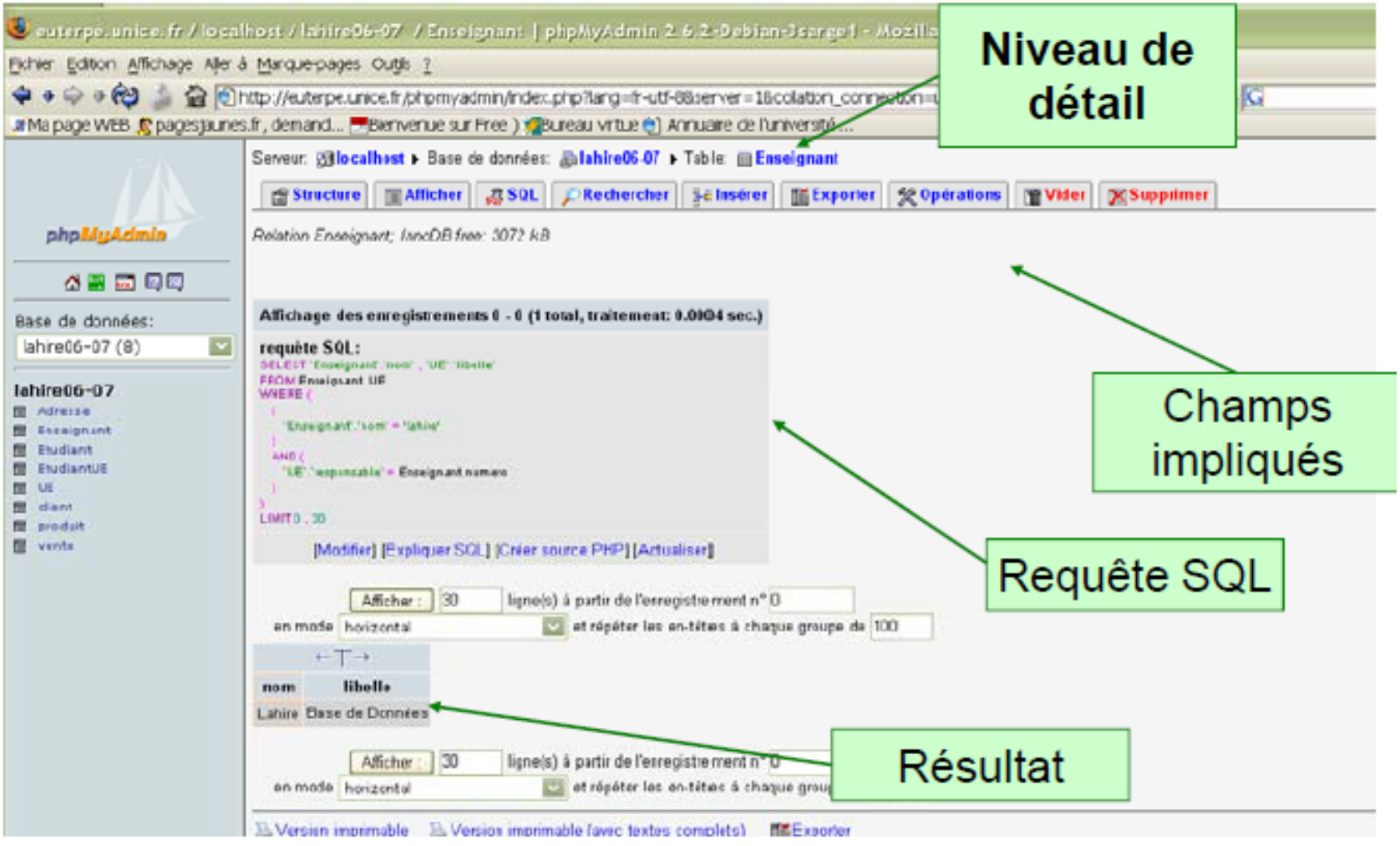
Interface graphique (phpMyAdmin) - 9



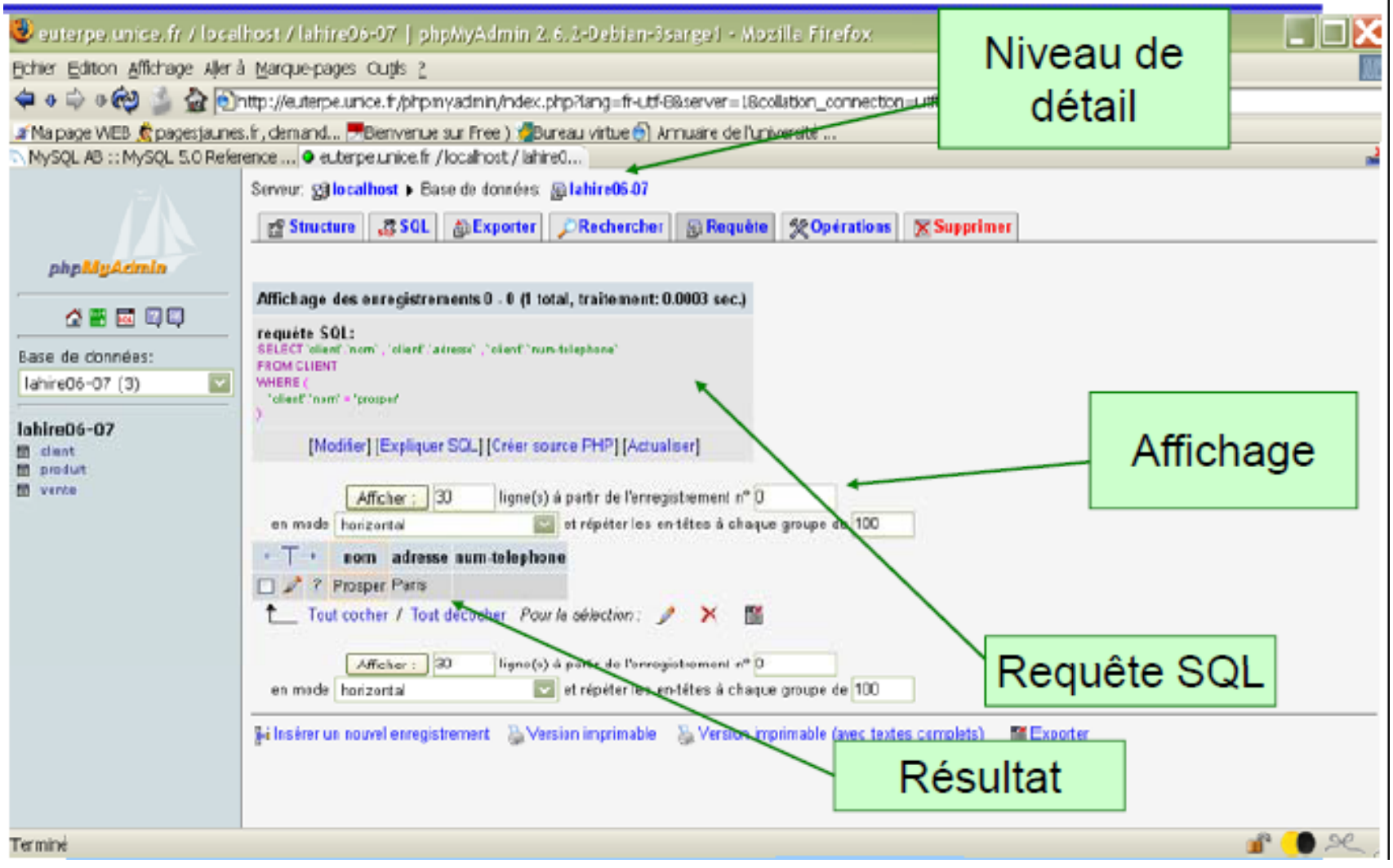
Interface graphique (phpMyAdmin) - 10



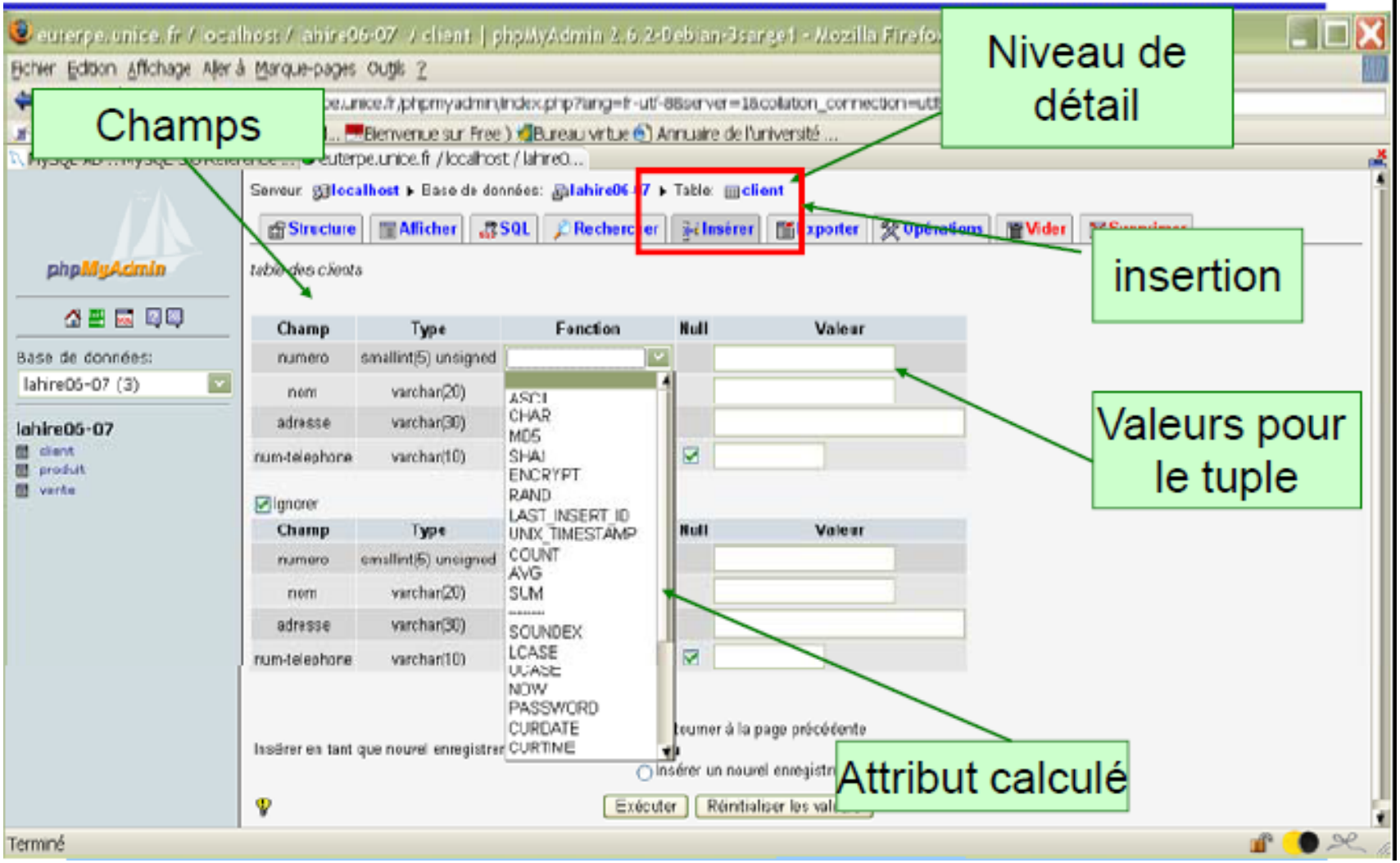
Interface graphique (phpMyAdmin) - 11



Interface graphique (phpMyAdmin) - 12



Interface graphique (phpMyAdmin) - 13



Résumé sur le modèle relationnel (1)

- Le système de gestion de base de données relationnelle est actuellement le logiciel de traitement de données le plus fréquemment utilisé
- Une **relation** mathématique est un sous-ensemble du produit cartésien de deux ensembles ou plus. En termes de base de données, une relation est n'importe quel sous-ensemble du produit cartésien des domaines des attributs
- Les relations sont représentées de manièere physique par des **tables**, dont les lignes correspondent aux tuples individiels et les colonnes aux attributs.

Résumé sur le modèle relationnel (2)

- Les propriétés d'une base de données sont les suivantes :
  - chaque cellule contient exactement une valeur atomique
  - les noms d'attributs sont distincts les uns des autres
  - l'ordre des attributs est immatériel
  - l'ordre des tuples est immatériel
  - il n'existe pas de tuples en double
- Dans un tuple, un **nul** représente une valeur d'un attribut, inconnue à l'heure actuelle ou qui ne s'applique pas à ce tuple

Résumé sur le modèle relationnel (3)

- Une **clé candidate** est un ensemble minimum d'attributs qu'identifie les tuples d'une relation de façon unique
- Une **clé primaire** est la clé candidate choisie pour servir à l'identification de tuples
- Une **clé étrangère** est un ensemble d'attributs au sein d'une relation qui constitue une clé candidate d'une autre relation
- l'**intégrité d'entité** établit que, dans une relation de base, aucun attribut qui fait partie de la clé primaire ne peut être nul
- l'**intégrité référentielle** établit que les valeurs d'une clé étrangère doivent correspondre à une valeur d'une clé candidate d'un tuple dans la relation de référence de la clé candidate ou être complètement nulles

2ème PARTIE :

Le langage SQL

Introduction

Objectifs de SQL

- Créer la structure de la base de données et de ses table
- Exécuter les tâches de base de la gestion des données, telle que l'insertion, la modification et la suppression de données des tables
- Effectuer des requêtes simples ou complexes

Langage orienté transformation

Origine

Structured **Q**uery **L**anguage

- caractéristiques des langages déclaratifs
- origine : IBM, System R, milieu des années 70
- implémenté dans de nombreux SGBD

Plusieurs versions :

- SQL1 initial : ANSI* —1986
- SQL1 avec intégrité référentielle, ANSI —1989
- SQL2 ANSI —1992
- SQL3 ANSI — 1999 incorpore la notion d'objet

** ANSI = American National Standard Institute*

Caractéristiques de SQL

- Fonctionnalités :
  - Définition des objets de la base de données (LDD)
  - Manipulation de données (LMD)
  - Contrôle des accès aux données
  - Gestion de transactions

- Utilisé par : DBA, développeurs, quelques utilisateurs

Parallèle avec mySQL

Principales Instructions

- Définitions (LDD)

  CREATE, DROP, ALTER

- Mises à jour (LMD)

  INSERT, UPDATE , DELETE

- Interrogations (LMD)

  SELECT

- Contrôle d'accés aux données

  GRANT, REVOKE

- Gestion de transactions

  COMMIT, ROLLBACK

Consultation des données

- Hypothèse:
  - un schéma de base de données est créé
  - Une base de données a été remplie
- La création a été faite par une interface QBE (mySQL)
- On expérimente la consultation avec SQL
- On expérimentera la création du schéma et le remplissage de la base avec SQL plus tard

➡ **Langage algébrique en SQL**

➡ **Requêtes mono et multi table(s)**

Exemple pour les requêtes

CLIENT

numéro	nom	adresse	numéro_téléphone
101	Durand	NICE	0493456743
108	Fabre	PARIS	NULL
110	Prosper	PARIS	NULL
125	Antonin	MARSEILLE	NULL

PRODUIT

référence	marque	Prix HT
153	BMW	1000
589	PEUGEOT	1800
158	TOYOTA	1500

VENTE

numéro	référence_produit	numéro_client	date
00102	153	101	12/10/04
00809	589	108	20/01/05
11005	158	108	15/03/05
12005	589	125	30/03/05

Format des requêtes

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY

- FROM spécifie la table ou les tables à utiliser
- WHERE filtre les lignes selon une condition donnée
- GROUP BY forme des groupes de lignes de même valeur de colonne
- HAVING filtre les groupes sujets à une certaine condition
- SELECT spécifie les colonnes qui doivent apparaître dans les résultats
- ORDER BY spécifie l'ordre d'apparition des données dans le résultat

Requêtes simples (SELECT-FROM)

- Afficher le nom et l'adresse des clients

SELECT *nom* , *adresse*

FROM *Client* ;

- Afficher toutes les informations des clients

SELECT *

FROM *Client* ;

Elimination des doublons

- Afficher l'adresse de tous les clients

```
SELECT adresse
FROM  Client ;
```

```
SELECT ALL adresse
FROM Client;
```

- Afficher toutes les adresses existantes (sans doublons)

```
SELECT DISTINCT adresse
FROM  Client ;
```

Par défaut

Projection

Relationnel : On doit pouvoir distinguer chaque tuple !

SQL : le résultat peut ne pas être une relation

Sélection de colonne (clause WHERE)

Les conditions fondamentales de recherche

- *comparaison*
  (salaire$>$10000, ville=’Paris’)
- *étendue* ou *intervalle*
  (salaire BETWEEN 20000 and 30000)
- *appartenance à un ensemble*
  (couleur IN (’red’, ’vert’))
- *correspondence à un masque*
  (adresse LIKE ’%Montréal%’)
- *nul*
  (adresse IS NULL)

Opérateur Sélection

- Quels sont les clients dont l'adresse est *Paris*

```
SELECT *
FROM Client
WHERE adresse = 'Paris';
```

- Quels sont les produits dont le prix TTC est supérieur à *1000*

```
SELECT *
FROM Produit
WHERE prix_HT + prix_HT * 0.195 > 1000;
```

Opérations possibles (mySQL)

- Booléennes
  and, or, xor, =, !=, <, >, <=, >=
- Arithmétiques
  +, -, *, /
  +, - *opérateurs unaires*
- Fonctions numériques
  abs, log, cos, sin, mod, power ...
- Arithmétiques sur date
  +, -
- Fonctions sur chaînes
  length, concat, ...

Un grand nombre

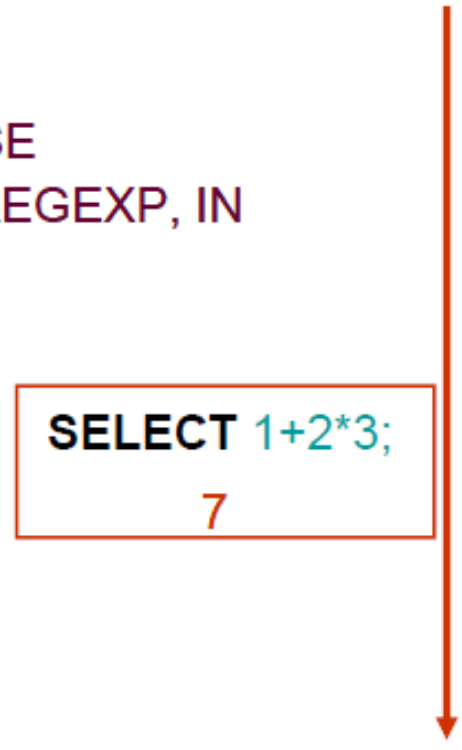
Dans Select: attribut calculé (résultat)

Dans Where: participer à la sélection

Précédence des opérateurs

- :=
- ||, OR, XOR
- &&, AND
- BETWEEN, CASE, WHEN, THEN, ELSE
- =, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
- |
- &
- <<, >>
- -, +
- *, /, DIV, %, MOD
- ^
- - (unary minus), ~ (unary bit inversion)
- !, NOT
- BINARY, COLLATE

+ faible



+ forte

```
SELECT 1+2*3;
7
```

Utilisation des opérateurs

- **SELECT** ABS (-32); 32
- **SELECT** FLOOR (1.23); 1
- **SELECT** MOD (234, 10); 4
- **SELECT 253** % 7; 1
- **SELECT** ROUND (1.298, 1); 1,3
- **SELECT** ROUND (1.298, 0); 1
- **SELECT** SIGN (234) SIGN (-32) SIGN (0); 1 / -1 / 0
- **SELECT** 3 / 5; 0,60

Bien sur on peut utiliser des attributs

Peut se trouver dans Where

WHERE 3 / 5 < 1;

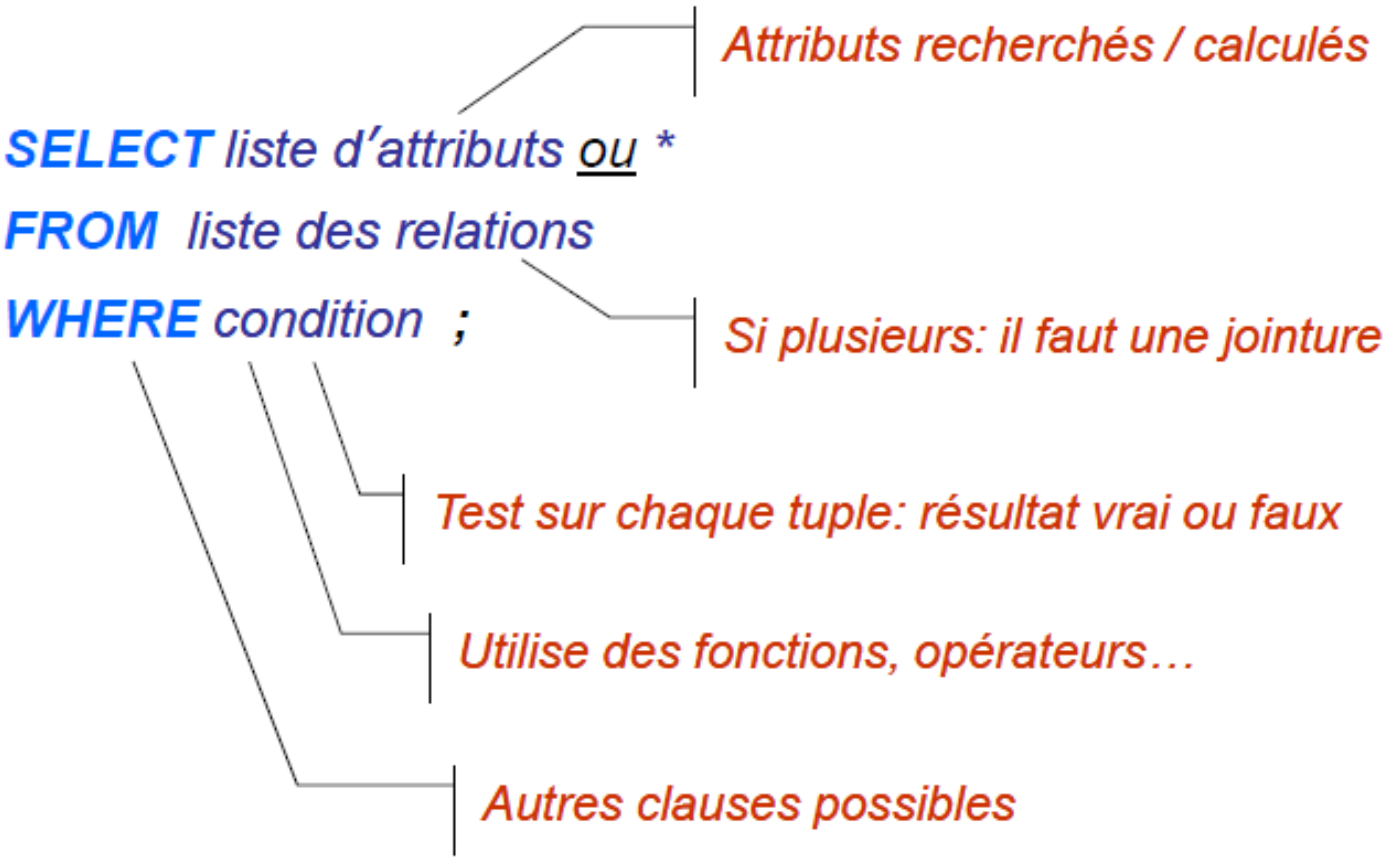
Utilisation des opérateurs (Chaînes)

- **SELECT** CONCAT ('*My*', '*S*', '*QL*'); ‘MySQL’
- **SELECT** CHAR_LENGTH (‘*MySQL*’); 5
- **SELECT** LOCATE ('bar', 'foobarbar'); 4
- **SELECT** LOCATE ('bar', 'foobarbar', 5); 7
- **SELECT** INSERT ('Quadratic', 3, 4, 'What'); 'QuWhattic'
- **SELECT** LOWER (‘*MySQL*’); ‘mysql’
- **SELECT** SUBSTRING ('Quadratically',5,6); ‘ratica'
- **SELECT** 'David!' LIKE ‘David_'; 1
- **SELECT** 'David!' LIKE ‘%D%v%'; 1
- **SELECT** STRCMP (S1, S2); -1,0,1

_ et %

0 (false), **1** (true)

Combinaison *Sélection + projection*



Requêtes simples (1)

- *Y a-t-il des produits dont le nom est « XBOX »*

```
SELECT *
FROM  Produit
WHERE nom = 'XBOX' ;
```

- *Quels sont les ventes réalisés il y a plus de 30 jours?*

```
SELECT *
FROM  Vente
WHERE CURRENT_DATE () > 30 + date ;
```

- *Quels sont les ventes faites après le 1er janvier 2007?*

```
SELECT *
FROM  Vente
WHERE date > DATE ('2007-01-01') ;
```

Requêtes simples (2)

- *Quels sont les ventes dont le montant HT est entre 1000 et 3000 euros et dont le client n'est pas le numéro 101?*

```
SELECT *
FROM  Vente
WHERE  (prix_ht between 1000 and 3000) and
       (numero != 101);
```

- *Quels sont les clients dont le nom est soit Prosper, soit Durand, soit Anthonin ?*

```
SELECT *
FROM  Client
WHERE  nom in ('Prosper', 'Durand', 'Anthonin') ;
```

Requêtes simples (3)

- *Quels sont les clients dont le nom commence par 'P'*

```
SELECT *
FROM  Employe
WHERE  nom LIKE 'P%' ;
```

- *Quels sont les clients dont le nom commence par 'P' et a un 'S' comme 4ème lettre*

```
SELECT *
FROM  Client
WHERE  nom LIKE  'P__S%' ;
```

Requêtes et valeurs nulles

• Quels sont les ventes dont la date de réalisation est inconnue?

```
SELECT *
FROM  Vente
WHERE  date is null ;
```

Ou : IS NOT NULL

Attention :

WHERE date = NULL ⇨ *Whatever* comparé avec NULL
→ Ni vrai ni faux

COUNT, MIN, SUM ⇨ *Ignore les valeurs* NULL
→ sauf COUNT (*)

Toute opération appliquée à NULL donne pour résultat NULL

Les autres clauses (tri)

```
SELECT attribut1 ..
FROM
WHERE expression
ORDER BY attribut1 [ASC] [DESC]…
```

Affichage!

- Donner le numero, le prix HT et la marque des produits selon l'ordre décroissant des marques et l'ordre croissant des prix HT

```
SELECT marque, prix_ht, numero
FROM  Produit
ORDER BY marque DESC, prix_ht ASC;
```

Ordre d'affichage!

Mais aussi:

```
ORDER BY 1 DESC, 2 ASC;
```

Requêtes multi- tables (Opérateur *Jointure*)

Deux points de vue:

- exécuter des **boucles imbriquées** (une table par boucle)
  appliquer la clause WHERE dans les boucles
- calculer le **produit cartésien** (une nouvelle table)
  Appliquer la clause WHERE sur chaque ligne

Donner le nom d'un produit et le montant de la vente

```
SELECT Produit.nom, Vente.prix_HT
FROM  Produit , Vente
WHERE Vente.reference_produit = Produit.numero ;
```

Critères de jointure

Il faut joindre les tables (jointure)

Définition de la jointure

SELECT *Client.nom, Vente.prix_ht*
FROM *Client, Vente ;*

→ tous les tuples (*nom*, *prix_ht*)
→ nom est un nom de client et prix_ht est un prix de vente

Pas de critère de jointure → Produit cartésien

Intérêt?

Mots-clés pour exprimer le critère de sélection

Jointure (compléments)

- *Donner la marque des produits dont le prix HT est supérieur à celui d'une BMW*

Renommage

```
SELECT Produit.marque
FROM Produit  Produit_reference, Produit
WHERE Produit_reference.marque = 'BMW' AND
        Produit.prix_HT > Produit_reference.prix_HT ;
```

Ou *de même prix HT*:

```
SELECT Produit.marque
FROM  Produit  Produit_reference, Produit
WHERE Produit_reference.marque = 'BMW' AND
        Produit.prix_ht = Produit_reference.prix_ht AND
        Produit.marque != Produit_reference.marque ;
```

Quelques jointures (compléments)

Que deviennent les tuples non sélectionnés de R1 ou R2?

- dans la jointure (interne ***INNER*** **JOIN**) les tuples qui ne peuvent pas être joints sont éliminés du résultat
- dans la jointure externe (**OUTER JOIN**) les tuples qui ne peuvent pas être joints sont conservés dans le résultat

Pour les tuples de la relation de gauche (R1) et / ou de droite (R2)

Défaut

R1 **FULL OUTER JOIN** R2 : *Remplit R1.* et R2.**

R1 **LEFT OUTER JOIN** R2 : *Remplit R1.**

R1 **RIGHT OUTER JOIN** R2 : *Remplit R2.**

avec NULL si nécessaire.

Exercice (1)

Relations :

- Journal (code-j, titre, prix, type, périodicité)
- Dépôt (no-dépôt, nom-dépôt, adresse)
- Livraison (no-dépôt, code-j, date-liv, quantité-livrée)

Requêtes : donner…

• le prix des journaux livrés le 15/01/07 ?
• tous le nom des hebdomadaires reçus par le dépôt de Paris..
• les titre des journaux livrés à Nice.
• le nom des dépôts qui reçoivent des hebdomadaires dont la quantité livrée excède 100.

Exercice (2)

```
SELECT *
FROM LIVRAISON RIGHT JOIN (DEPOT, JOURNAL)
ON (LIVRAISON.no-depot = DEPOT.no-depot AND
    LIVRAISON.code-j = JOURNAL.code-j)
```

Toutes les lignes de DEPOT et JOURNAL seront présentes
Avec éventuellement rien pour la partie livraison

```
SELECT *
FROM LIVRAISON, DEPOT, JOURNAL
WHERE LIVRAISON.no-depot = DEPOT.no-depot AND
      LIVRAISON.code-j = JOURNAL.code-j)
```

Seulement les lignes de DEPOT et JOURNAL qui correspondent à une livraison

Opérateur *Union*

Afficher la liste des numéros d'employé des responsables de département et des directeurs

Département

numéro-département	...	responsable

Employé

numéro-employé	...	fonction

```
SELECT responsable
FROM Département
UNION
SELECT numéro-employé
FROM Employé
WHERE fonction = 'Directeur';
```

Opérateurs *Intersection, Différence*

Afficher les numéros d'employé des responsables de département qui sont aussi des directeurs

```
SELECT responsable
FROM Département
INTERSECT
SELECT numéro-employé
FROM Employé
WHERE fonction = 'Directeur';
```

```
SELECT responsable
FROM Département
EXCEPT
SELECT numéro-employé
FROM Employé
WHERE fonction = 'Directeur';
```

Afficher les numéros d'employé des responsables de département Sauf ceux qui sont aussi des directeurs

Fonctions d'agrégat

- SUM (*nom d'attribut*)
- COUNT(*)
- COUNT(DISTINCT *nom d'attribut*)
- MAX (*nom d'attribut*)
- MIN (*nom d'attribut*)
- AVG (*nom d'attribut*)
- AVG (DISTINCT *nom d'attribut*)

Dans les clauses :
- SELECT
- HAVING TO

Clause Group by

SELECT *attributs recherchés*
FROM *liste des relations*
WHERE *condition*
GROUP BY *attributs de regroupement*
[**HAVING** *condition sur le groupe*];

```
SELECT COUNT (*)
FROM  Produit
GROUP BY marque
```

```
SELECT AVG (prix_ht), nom
FROM  Produit
GROUP BY marque
```

```
SELECT COUNT (*)
FROM  Produit
WHERE marque <> 'BMW'
GROUP BY marque
HAVING AVG (prix_ht) > 10.5
```

Utilisation des fonctions statistiques

- *Donner la moyenne des prix HT et les prix min. et max.*

SELECT AVG *(prix_ht)*, MIN *(prix_ht)*, MAX *(prix_ht)*
FROM *Produit*

- *Même chose mais par marque*

SELECT AVG *(prix_ht)*, MIN *(prix_ht)*, MAX *(prix_ht)*
FROM *Produit*
GROUP BY *marque*

- Même chose mais par marque si la moyenne > 10,5

SELECT AVG *(prix_ht)*, MIN *(prix_ht)*, MAX *(prix_ht)*
FROM *Produit*
GROUP BY *marque* **HAVING** AVG *(prix_ht) > 10.5*

Requêtes imbriquées (1)

SELECT *liste d'attributs recherchés*

FROM *liste_des_relations*

WHERE ***bloc SFW dans la condition***

Intérêt: indiquer qu'un attribut doit prendre ses valeurs dans une liste de valeurs définies par un autre bloc *SWF*

- Itérations imbriquées
- autre façon de faire certaines formes de jointure
- Sous-requêtes indépendantes ou pas

Requêtes imbriquées (2)

Principaux connecteurs: = != > < IN EXISTS ANY ALL

EXISTS *R* : retourne TRUE si *R* n'est pas vide, FALSE sinon

t **IN** *R* : retourne TRUE si t appartient à *R*, FALSE sinon

valeur *comp* **ANY** *R* : retourne TRUE si la comparaison avec **au moins un** des tuples de **R** renvoie TRUE

valeur *comp* **ALL** *R* : retourne TRUE si la comparaison avec **tous** les tuples de **R** renvoie TRUE

Sous-requêtes indépendantes

Quelles sont les références produit dont le prix HT est supérieur au prix HT moyen des produits ?

```
SELECT référence
FROM  Produit
WHERE Produit.prix_ht  > (SELECT AVG(P.prix_ht)
                          FROM Produit P);
```

Le bloc SFW imbriqué peut être évalué séparément du bloc principal

Sous-requêtes indépendantes (1)

Quels sont les marques de produits vendus le 1/1/2007 ?

```
SELECT marque
FROM  Produit, Vente
WHERE Vente.ref-produit = Produit.référence
          AND date = 1/1/2007;
```

ou

```
SELECT Produit.marque
FROM  Produit
WHERE Produit.référence IN
                    (SELECT Vente.ref-produit
                     FROM Vente
                     WHERE date = 1/1/2007);
```

Jointure ou requête imbriquée

Sous-requêtes indépendantes (2)

Quelles sont les marques de produits qui n'ont pas le prix HT le plus élevé ?

SELECT marque
FROM *Produit*
WHERE Produit.prix-ht < **ANY**
(**SELECT** *P*.prix-ht
FROM *Produit P;*)

< SELECT max (P.prix_ht) …

Quelles sont les marques de produits qui ont le prix HT le plus élevé ?

SELECT marque
FROM *Produit*
WHERE *Produit*.prix_ht **>= ALL**
(**SELECT** *P*.prix_ht **FROM** *Produit P;*)

> SELECT max (P.prix_ht) …

Sous-requêtes corrélées (1)

Quels produits dont le prix HT est supérieur au prix HT moyen des produits de la même marque ?

```
SELECT Produit.référence
FROM   Produit
WHERE Produit.prix-ht > (SELECT AVG(P.prix-ht)
                         FROM Produit P
                         WHERE Produit.marque = P.marque);)
```

Le bloc *SFW* principal et le bloc *SFW* imbriqué doivent être évalués simultanément

Sous-requêtes corrélées (2)

Quels sont les marques dont aucun produit n'a été vendu le 1/01/2007 ?

```
SELECT Produit.marque
FROM   Produit
WHERE NOT EXISTS
        (SELECT Vente.ref-produit
         FROM Vente, Produit P
          WHERE Vente.ref-produit = P.référence and
                P.marque = Produit.marque and
                date = 1/01/2007;)
```

Exercices

Donner le nom des employés qui ont le même salaire que Dupont

Employé

numéro-employé	nom	...	salaire

```
SELECT Y.nom
FROM  Employé  X , Employé  Y
WHERE
        X.nom='Dupont'
        and Y.salaire = X.salaire
        and X.nom != Y.nom;
```

Remplacer la jointure par une requête imbriquée

Manipulation des structures de données

- Schéma logique / Base de données

  → CREATE/DROP SCHEMA/DATABASE ...

- Schéma de ***tables*** *(relations)* et de leur contenu

  → CREATE/ALTER/DROP TABLE ...

- Définition des ***contraintes*** qui assurent des contrôles sur l'intégrité des données

- Mais encore :
  - les index
  - les utilsateurs et les privilèges

Création/Suppression d'un schéma

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] *db_name*

DROP DATABASE [IF EXISTS] *db_name*

Pas de message d'erreur si la BD existe!

- Dans la norme SQL: création de schéma
- MySQL: *schema* = *database*
- Il faut avoir les droits de le faire (administrateur?)
- Création d'un compte s'il n'existe pas
- Association des droits

CREATE DATABASE *lahire06-07*

GRANT ALL PRIVILEGES
ON *lahire06-07.**
TO '*lahire06-07*'@localhost
IDENTIFIED BY '*lahire06-07*';

DROP DATABASE *lahire06-07*

Attention !

Types de la norme ANSI

INTEGER	Entiers relatifs	4 octets
SMALLINT	Entiers relatifs	2 octets
BIGINT	Entiers relatifs	8 octets
FLOAT	Flottants	4 octets
DOUBLE	Flottants	8 octets
REAL	Flottants	4 ou 8 octets
NUMERIC(N,D)/ DECIMAL(N,D)	Décimaux à précision fixe	N octets

CHAR(M)	Chaînes de longueur fixe	M octets
VARCHAR(M)	Chaînes de longueur variable	au plus M octets
BIT VARYING	Chaînes d'octets	longueur de la chaîne
DATE	date(hour, mois, an)	4 octets
TIME	heure(h, mn, sh)	4 octets
DATETIME	date et heure	8 octets
YEAR	année	2 octets

Création et gestion des relations

- Structures de données (relations, attributs, tuples)

relation	→	**table**
attribut	→	**column**
tuple	→	**row**

- Instructions sur les relations

CREATE TABLE	*créer une relation*
DROP TABLE	*supprimer une relation*
ALTER TABLE	*modifier la structure d'une relation*

Création de table/relation (1)

```
CREATE TABLE Produit (
référence CHAR(4),
    nom VARCHAR(20),
    marque VARCHAR(15),
    constructeur        CHAR(20)
    );
```

```
CREATE TABLE Client (
numéro INT(10),
    nom VARCHAR(20),
    adresse VARCHAR(30),
    téléphone CHAR(10)
    );
```

```
CREATE TABLE Vente (
numéro INT(10),
    ref-produit INT(10),
    ref-client INT(10),
    date DATE
    );
```

- *Compléter la description*
- *Voir avec les types MySQL*

Création de table (valeurs nulles, défaut)

- NOT NULL : l'attribut correspondant doit *toujours* avoir une valeur

  nom **VARCHAR(20) NOT NULL**

- DEFAULT : définir la valeur d' un attribut par défaut

  adresse **VARCHAR (30) DEFAULT** 'Inconnue'

```
CREATE  TABLE  Produit (
référence CHAR(4) NOT NULL,
     nom VARCHAR(20) NOT NULL,
     marque VARCHAR(15),
     constructeur        CHAR(20) DEFAULT 'Renault'
     );
```

Création de table (clé primaire)

Identification des tuples

PRIMARY KEY (*att*)

PRIMARY KEY (*att1*, ..., *attn*)

→ mono attribut ou multi attributs

→ Attributs figurant dans une clé : déclaration NOT NULL

```
CREATE  TABLE  Produit (
référence CHAR(4) NOT NULL,
     nom VARCHAR(20) NOT NULL,
     marque VARCHAR(15),
     constructeur        CHAR(20) DEFAULT 'Renault',
     PRIMARY KEY (référence)
     );
```

Création de table (unicité)

Unicité des valeurs dans une colonne

UNIQUE (nom, prenom)

→ Principalement pour les clés secondaires

```
CREATE  TABLE  Produit (
référence CHAR(4) NOT NULL,
     nom VARCHAR(20) NOT NULL,
     marque VARCHAR(15),
     constructeur        CHAR(20),
     PRIMARY KEY (référence),
     UNIQUE (nom)
     );
```

NOT NULL non spécifié

UNIQUE ne s'applique pas aux valeurs nulles

Création de table (clé étrangère)

FOREIGN KEY (att) REFERENCES ...
FOREIGN KEY (att1, ..., attn) REFERENCES ...

FOREIGN KEY(ref-produit) **REFERENCES** *Produit*
→ ref-produit référence la clé primaire de la table **Produit**

```
CREATE TABLE Vente (
numéro INT(10) NOT NULL,
    ref-produit INT(10) NOT NULL,
    ref-client INT(10) NOT NULL,
    date DATE,
    PRIMARY KEY (numéro),
    FOREIGN KEY (ref-produit) REFERENCES Produit,
    FOREIGN KEY (ref-client) REFERENCES Client
    );
```

Gestion de l'intégrité référentielle

Vérification, si insertion, suppression, mise à jour …

Les valeurs prises par la clé étrangère correspondent-elles à la clé?

Action par défaut : **rejet** de l'opération

Autres actions si celle par défaut ne convient pas

- **ON UPDATE** *(en cas de mise à jour)*
- **ON DELETE** *(en cas de suppression)*

Objectif: faire respecter la contrainte

- **SET NULL** clé étrangère mise à NULL
- **CASCADE** application de la même opération
- **SET DEFAULT** valeur par défaut pour la clé étrangère

Intégrité référentielle (exemple)

```
CREATE  TABLE  Vente (
numéro INT(10) NOT NULL,
    ref-produit INT(10) NOT NULL,
    ref-client INT(10) NOT NULL,
    date DATE,
    PRIMARY KEY (numéro),
    FOREIGN KEY (ref-produit) REFERENCES Produit
    ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (ref-client) REFERENCES Client
    ON DELETE SET NULL ON UPDATE CASCADE
    );
```

- Si suppression dans Produit : ref-produit = NULL
- Si mise à jour dans Produit : ref-produit = mise à jour

Mise à jour d'un schéma de BD

- Suppression d'une relation

  DROP TABLE *nom de relation ;*

- Modification d'une table

  ALTER TABLE *nom de_relation* *ACTION* *description*

  ADD, MODIFY, ALTER, DROP, RENAME

  description commande

Mais aussi :

ALTER TABLE Vente TYPE = *innodb*

Modification d'une table (structure)

```
ALTER TABLE Produit MODIFY nom VARCHAR(25);

ALTER TABLE Produit ADD quantité-stock VARCHAR(20);

ALTER TABLE Client ALTER adresse SET DEFAULT 'NICE';

ALTER TABLE Client DROP adresse;

ALTER TABLE Client
   ADD CONSTRAINT CT_UN UNIQUE (nom) ;

ALTER TABLE Client
   ADD CONSTRAINT CT_PR PRIMARY KEY (numéro) ;

ALTER TABLE Vente ADD CONSTRAINT CT_ET
   FOREIGN KEY (ref-client) REFERENCES Client  ;
```

Mises à jour des données

- **INSERT INTO**

pour insérer des tuple

- **DELETE FROM**

pour supprimer des tuples

- **UPDATE**

pour mettre à jour des tuples

Insertion de tuples (1)

Insertion avec désignation explicite des colonnes

```
INSERT INTO Produit (nom, marque, constructeur)
VALUES ('mercedes 300 SL', 'mercedes benz',
        'daimler');
```

Insertion dans l'ordre des colonnes

```
INSERT INTO Produit
VALUES (DEFAULT, 'mercedes 300 SL', 'mercedes
        benz', 'daimler');
```

Insertion de tuples (2)

Insertion à partir d'une autre table

INSERT INTO *Produit* (nom, marque, constructeur)
SELECT P.nom, P.marque, p.constructeur
FROM *OldProduit P*
WHERE *P*.marque = '*Peugeot*'

➡ **Evolution du schéma relationnel**

Suppression de tuples

Suppression de tous les tuples

```
DELETE FROM Produit
```

ou

```
DELETE FROM Produit WHERE 1 > 0
```

Suppression conditionnelle

```
DELETE FROM Produit
WHERE marque = 'peugeot'
```

```
DELETE FROM VENTE WHERE ref-produit IN
      (SELECT  référence
      FROM Produit
      WHERE Produit.marque = 'peugeot') ;
```

Modification des tuples

```
UPDATE Produit
SET nom = 'mercedes 300 SLK'
WHERE numero =3;
```

Modification d'un tuple

```
UPDATE Produit
SET prix-ht = prix-ht * 0,9
WHERE référence NOT IN
      (SELECT ref-produit
       FROM VENTE
       WHERE quantité > 10);
```

Modification d'une collection de tuple

Compléments: Définition de vues

Une vue est une table virtuelle dérivée de tables de base :

- On stocke seulement la définition de vue
- Son contenu est généré dynamiquement

```
CREATE VIEW ProduitPhare
AS
SELECT *
FROM Vente V, Produit P
WHERE
  V.ref-produit = P.référence
  AND V.quantité > 100 ;
```

```
CREATE VIEW VenteInfo
    (référence, date, marque)
AS
SELECT P.référence, V.date,
           P.marque
FROM  Vente V, Produit P
WHERE P.référence =V.ref-produit;
```

Evolution du schéma relationnel

Requêtes fréquentes

DROP VIEW VenteInfo

Compléments: Création de contraintes

CREATE ASSERTION
CHECK *pour décrire des contraintes générales*

CREATE ASSERTION *QuantiteVendue*
CHECK NOT EXISTS (
    SELECT *V*.quantité
    FROM *Vente V*, *Produit P*
    WHERE
      V.ref-produit = *P*.référence AND
      P.quantité-vendue < *V*.quantité);

On suppose que la table Produit contient la colonne quantité-vendue

Compléments: Création d'index

CREATE INDEX

Un index offre un chemin d'accès aux lignes d'une table

CREATE INDEX index1 ON Client(nom,prenom);

un index est systématiquement défini sur la clé primaire de chaque table

pour chaque clause UNIQUE utilisée dans la création de la table, un index permet de vérifier rapidement, au moment d'une insertion, que la clé n'existe pas déjà

Compléments: Gestion des privilèges

SHOW DATABASES ;
SHOW PRIVILEGES ;

GRANT *privilèges* ON … TO …

REVOKE *privilèges* ON … FROM …

ALL [PRIVILEGES]	Tous les droits sauf WITH GRANT OPTION
ALTER	Autorise l'utilisation de ALTER TABLE
CREATE	Autorise l'utilisation de CREATE TABLE
DELETE	Autorise l'utilisation de DELETE
DROP	Autorise l'utilisation de DROP TABLE
…	…