

DUT MMI – IUT de Marne-la-Vallée
12/03/2019
M2203 – Bases de données

Cours 3 & 4

Le langage SQL

Sources

- Cours de Tony Grandame à l'IUT de Marne-la-Vallée en 2010-2011

- Cours de Mathieu Mangeot, IUT de Savoie

<http://jibiki.univ-savoie.fr/~mangeot/Cours/BasesDeDonnees.pdf>

- Cours de Fabrice Meuzeret, IUT de Troyes

<http://195.83.128.55/~fmeuzeret/vrac/>

- Livre de Laurent Audibert : *Bases de données - de la modélisation au SQL*

Version partielle sur :

<http://laurent-audibert.developpez.com/Cours-BD/html/index.php>

Plan du cours 3 – Le langage SQL

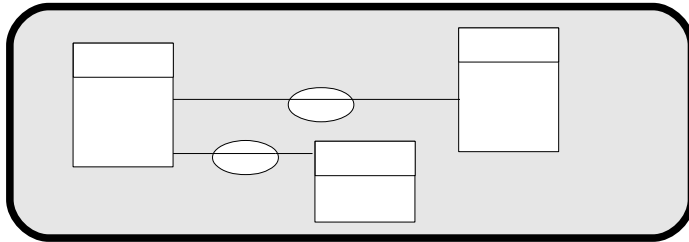
- Résumé des épisodes précédents
- Introduction au langage SQL
- Langage de définition des données
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions

Plan

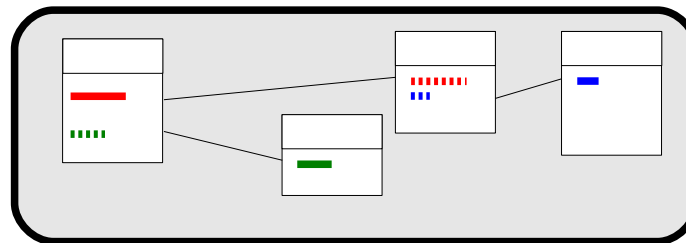
- Résumé des épisodes précédents
- Introduction au langage SQL
- Langage de définition des données
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions

Modèle physique des données

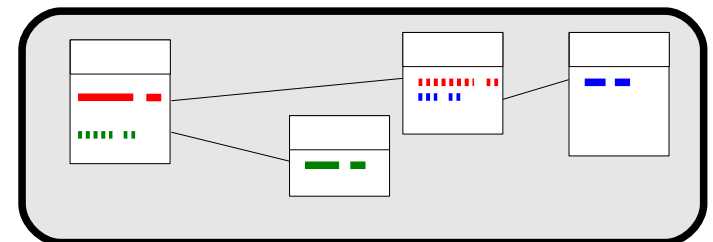
Modèle entité-association
(modèle conceptuel des données)



Modèle logique des données



Modèle physique des données



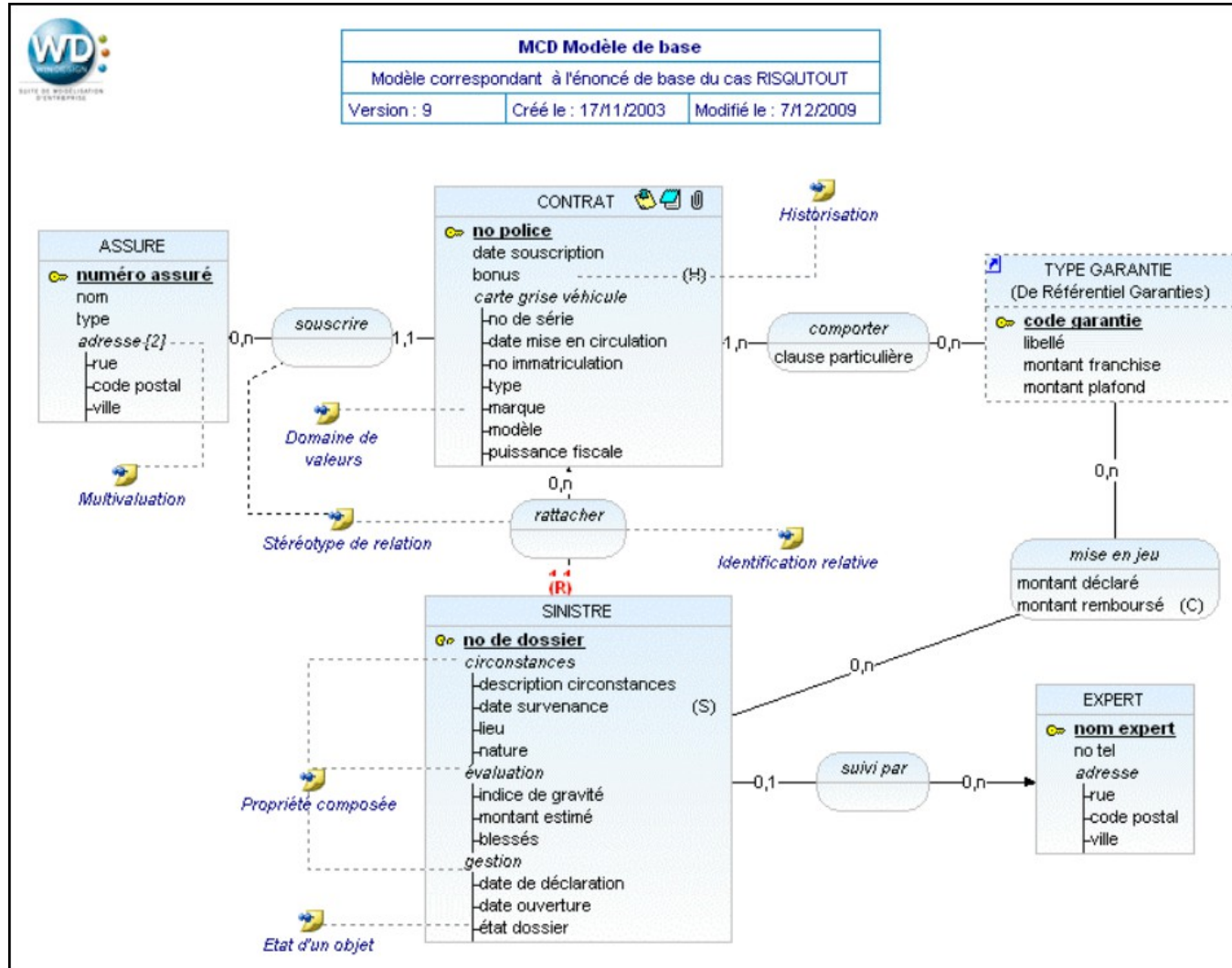
Transformation vers le modèle logique des données

Modèle entité association



Modèle logique des données

Transformation automatique : exemple de WinDesign Database



Transformation vers le modèle logique des données

Modèle entité association



Modèle logique des données

Transformation automatique : exemple de WinDesign Database

The screenshot displays the WinDesign Database interface. On the left, an Entity-Relationship model is shown with an entity 'ASSURE' containing attributes: 'numéro assuré' (primary key), 'nom', 'type', and 'adresse {2}' (array of attributes: 'rue', 'code postal', 'ville'). A relationship 'so' is shown with cardinality '0,n'. Other elements like 'no police', 'Historisation', 'Multivaluation', 'Stéréo', 'Propriété', and 'Etat d' are also visible.

At the top, a metadata box for 'MCD Modèle de base' indicates it corresponds to the 'RISQUTOUT' case, with version 9, created on 17/11/2003, and modified on 7/12/2009.

The main content area is divided into two columns:

Période	Thème
J 2 Matinée	MODELE LOGIQUE DE DONNEES <ul style="list-style-type: none">▪ Principes généraux de la modélisation logique des données▪ Concepts de base<ul style="list-style-type: none">- table, attribut, types de données- clé primaire, clé étrangère et contraintes référentielles, index▪ Transformation MCD ⇒ MLD<ul style="list-style-type: none">- préparation à la transformation,- options de la transformation automatique, règles de nommage,<i>Exercices d'application en continu sur la cas Risquetout</i>▪ Optimisation du MLD<ul style="list-style-type: none">- dénormalisation- tables de valeurs codées- index, choix d'implémentation de liens référentiels- historique des suppressions- mise en conformité du MCD<i>Exercices d'application en continu sur la cas Risquetout</i>▪ Concepts avancés<ul style="list-style-type: none">- vue SQL- règle (implémentation et codage),- trigger (référentiel et utilisateur)- implantation physique (storage et tablespace)<i>Exercices d'application en continu sur la cas Risquetout</i>

Plan

- Résumé des épisodes précédents
- **Introduction au langage SQL**
- Langage de définition des données
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions

Introduction au langage SQL

SQL

- Structured Query Language
- Langage standardisé pour effectuer des opérations sur des bases de données.
- **LDD : langage de définition de données**, pour gérer les structures de la base
- **LMD : langage de manipulation de données**, pour interagir avec les données.

Attention, certaines syntaxes ou fonctions sont propres au système de base de données utilisé.

Introduction au langage SQL

SQL

- Structured Query Language (“query” = “requête”)
- Langage standardisé pour effectuer des opérations sur des bases de données.
- **LDD : langage de définition de données**, pour gérer les structures de la base
- **LMD : langage de manipulation de données**, pour interagir avec les données.

Attention, certaines syntaxes ou fonctions sont propres au système de base de données utilisé.

requête : instruction demandant une action sur la base de données.

Alternative au langage SQL : clic-clic-poët-poët avec PhpMyAdmin

Plan

- Résumé des épisodes précédents
- Introduction au langage SQL
- **Langage de définition des données**
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions

Langage de définition des données

Bases

Une base regroupe toutes les données nécessaires pour un besoin fonctionnel précis : **une application ↔ une base de données.**

Possible de créer autant de bases de données que nécessaires, interaction entre les bases de données possible, mais alourdit la syntaxe SQL.

Création d'une base de données

```
CREATE DATABASE [IF NOT EXISTS] db_name  
[create_specification]
```

Les spécifications permettent notamment de définir l'encodage de caractères de la base :

```
CREATE DATABASE db_name DEFAULT CHARACTER SET latin1  
COLLATE latin1_swedish_ci;
```

Langage de définition des données

Bases

Une base regroupe toutes les données nécessaires pour un besoin fonctionnel précis : **une application ↔ une base de données.**

Possible de créer autant de bases de données que nécessaires, interaction entre les bases de données possible, mais alourdit la syntaxe SQL.

Suppression d'une base de données

```
DROP DATABASE [IF EXISTS] nom_de_base_de_donnees
```

Langage de définition des données

Bases

Une base regroupe toutes les données nécessaires pour un besoin fonctionnel précis : **une application ↔ une base de données.**

Possible de créer autant de bases de données que nécessaires, interaction entre les bases de données possible, mais alourdit la syntaxe SQL.

Modification d'une base de données

```
ALTER DATABASE nom_de_base_de_donnees  
alter_specification [, alter_specification] ...
```

Langage de définition des données

Tables

Rappel : Une table correspond à une entité.

Une base de données contient une ou plusieurs tables.

Création d'une table :

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_de_table  
[(create_definition,...)] [table_options]
```

- 1. `create_definition` représente la liste des champs avec leur type et leurs éventuelles options.
- 2. `table_option` permet de préciser notamment le système d'encodage des caractères, et le moteur de la table (`ENGINE`).

Introduction au langage SQL

SQL

- Structured Query Language (“query” = “requête”)
- Langage standardisé pour effectuer des opérations sur des bases de données.
- **LDD : langage de définition de données**, pour gérer les structures de la base
- **LMD : langage de manipulation de données**, pour interagir avec les données.

Attention, certaines syntaxes ou fonctions sont propres au système de base de données utilisé.

requête : instruction demandant une action sur la base de données.

Alternative au langage SQL : clic-clic-poët-poët avec PhpMyAdmin

Création d'une table avec PhpMyAdmin

The screenshot shows the phpMyAdmin interface for the 'gambette_xkcd' database. The left sidebar contains the database name and a list of tables: 'authors', 'french', and 'images'. A 'Nouvelle table' button is visible at the bottom of the sidebar.

The main area displays a table structure overview for the 'gambette_xkcd' database. The table lists three existing tables: 'authors', 'french', and 'images'. Each table row includes checkboxes for selection, an 'Afficher' button, and icons for 'Structure', 'Rechercher', 'Insérer', 'Vider', and 'Supprimer'. The columns are 'Table', 'Action', 'Lignes', 'Type', 'Interclassement', 'Taille', and 'Perte'.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> authors	Afficher Structure Rechercher Insérer Vider Supprimer	25	MyISAM	latin1_swedish_ci	3,2 Kio	-
<input type="checkbox"/> french	Afficher Structure Rechercher Insérer Vider Supprimer	2 036	MyISAM	latin1_swedish_ci	1,5 Mio	-
<input type="checkbox"/> images	Afficher Structure Rechercher Insérer Vider Supprimer	1 046	MyISAM	latin1_swedish_ci	68,5 Kio	-
3 tables	Somme	3 107	MyISAM	latin1_swedish_ci	1,5 Mio	0

Below the table structure, there are options for 'Tout cocher / Tout décocher' and a dropdown for 'Pour la sélection :'. There are also links for 'Version imprimable' and 'Dictionnaire de données'.

The 'Créer une nouvelle table sur la base gambette_xkcd' dialog is open, showing the 'Nom' field with the value 'Personne' and the 'Nombre de colonnes' field with the value '4'. An 'Exécuter' button is located at the bottom right of the dialog.

Création d'une table avec PhpMyAdmin

The screenshot shows the phpMyAdmin interface for creating a new table. The table name is 'Personne'. The columns are defined as follows:

Colonne	Type	Taille/Valeurs*1	Défaut2
Id	INT		Aucune
Prenom	VARCHAR	100	Aucune
Nom	VARCHAR	100	Aucune
Adresse	VARCHAR	255	Aucune

Additional settings shown at the bottom of the dialog:

- Commentaires sur la table: (empty)
- Moteur de stockage: MyISAM
- Interclassement: (empty)

An 'Annuler' (Cancel) button is visible at the bottom right of the dialog box.

Création d'une table avec PhpMyAdmin

The screenshot displays the 'Nouvelle table' (New Table) window in phpMyAdmin. The table name is 'gambette_xkcd'. The columns are defined as follows:

Column Name	Type	Length	Collation
Prenom	VARCHAR	100	Aucune
Nom	VARCHAR	100	Aucune
Adresse	VARCHAR	255	Aucune

Additional settings include:

- Commentaires sur la table: (empty)
- Moteur de stockage: MyISAM
- Interclassement: (empty)
- Définition de PARTITION: (empty)

At the bottom, the interface shows the text 'Sauvegarder Ou Ajouter 1 colonne(s) Exécuter'. The 'Exécuter' button is circled in red. An 'Annuler' button is located at the bottom right of the window.

Langage de définition des données

Tables

Rappel : Une table correspond à une entité.

Une base de données contient une ou plusieurs tables.

Création d'une table :

- 1. La liste des champs doit être précisée :

```
col_name type [NOT NULL | NULL] [DEFAULT  
default_value] [AUTO_INCREMENT] [[PRIMARY] KEY]  
[reference_definition]
```

Seuls le nom et le type sont obligatoires.

Par défaut un champ est défini en NULL.

Les champs sont séparés par des virgules.

Langage de définition des données

Tables

Rappel : Une table correspond à une entité.

Une base de données contient une ou plusieurs tables.

Création d'une table :

- 1. L'option `AUTO_INCREMENT` permet de confier la gestion du champ par le moteur de base de données.

A chaque insertion dans la table, la valeur du champ sera automatiquement incrémentée.

Cette option n'est possible que sur des champs de type entier.

Le type `SERIAL` est un raccourci pour définir un champ `UNSIGNED BIGINT AUTO_INCREMENT UNIQUE`.

Langage de définition des données

Tables

Rappel : Une table correspond à une entité.

Une base de données contient une ou plusieurs tables.

Création d'une table :

- 2. Les options facultatives de la tables permettent de préciser (en outre) :
 - le moteur de la table :
 - MyIsam (par défaut)
 - InnoDB (gère les transactions)
 - Memory (chargée en mémoire)
 - Le système d'encodage de caractères, par défaut `latin1_swedish_ci` correspondant à ISO-8859.

Langage de définition des données

Tables

Rappel : Une table correspond à une entité.

Une base de données contient une ou plusieurs tables.

Exemples de création d'une table

```
CREATE TABLE IF NOT EXISTS Coord (id int(11) NOT NULL
auto_increment, name varchar(255) collate
latin1_general_ci NOT NULL, type varchar(255) collate
latin1_general_ci NOT NULL, coord varchar(255)
collate latin1_general_ci NOT NULL, url varchar(255)
collate latin1_general_ci NOT NULL, PRIMARY KEY
(id)) ENGINE=MyISAM DEFAULT CHARSET=latin1
COLLATE=latin1_general_ci AUTO_INCREMENT=201 ;
```

Langage de définition des données

Index

Un index permet au moteur d'**accéder rapidement à la donnée recherchée**.

Si vous recherchez un champ ayant une valeur donnée et qu'il n'y a pas d'index sur ce champ, le moteur devra parcourir toute la table.

Index à utiliser avec parcimonie : pénalisent les temps d'insertion et de suppression des données dans la table.

Une clé primaire est par définition un index unique sur un champ non nul.
Un index peut être nul.

```
CREATE TABLE IF NOT EXISTS Personne(id int NOT NULL  
primary key auto_increment, nom varchar(100) not  
null, prenom varchar(100), annee_naiss year default  
"1950") ENGINE=InnoDB
```


Langage de définition des données

Index

Un index permet au moteur d'**accéder rapidement à la donnée recherchée**.

Si vous recherchez un champ ayant une valeur donnée et qu'il n'y a pas d'index sur ce champ, le moteur devra parcourir toute la table.

Index à utiliser avec parcimonie : pénalisent les temps d'insertion et de suppression des données dans la table.

Une clé primaire est par définition un index unique sur un champ non nul.
Un index peut être nul.

```
CREATE TABLE IF NOT EXISTS Personne(id int NOT NULL  
primary key auto_increment, nom varchar(100) not  
null, prenom varchar(100), annee_naiss year default  
"1950") ENGINE=InnoDB
```

```
CREATE TABLE IF NOT EXISTS Personne(nom varchar(100)  
not null, prenom varchar(100), annee_naiss year  
default "1950", primary key (nom, prenom), index  
personne_annee (annee_naiss)) ENGINE=InnoDB
```

Langage de définition des données

Modification d'une table

```
CREATE TABLE Nom_de_table
ADD [COLUMN] column_definition [FIRST | AFTER
col_name ]
| ADD INDEX [index_name] [index_type]
(index_col_name,...)
| ADD PRIMARY KEY [index_type] (index_col_name,...)
| ALTER [COLUMN] col_name {SET DEFAULT literal | DROP
DEFAULT}
| ALTER TABLE tbl_name
| ADD FOREIGN KEY [index_name] (index_col_name,...)
| CHANGE [COLUMN] old_col_name column_definition
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP INDEX index_name
| DROP FOREIGN KEY fk_symbol
```

Langage de définition des données

Modification d'une table

Renommage d'une table :

```
RENAME TABLE Nom_de_table TO Nouveau_nom_de_table
```

Suppression d'une table :

```
DROP TABLE Nom_de_table
```

Attention, cette action est irréversible, toutes les données contenues dans la table sont évidemment supprimées.

Plan

- Résumé des épisodes précédents
- Introduction au langage SQL
- Langage de définition des données
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions

Plan

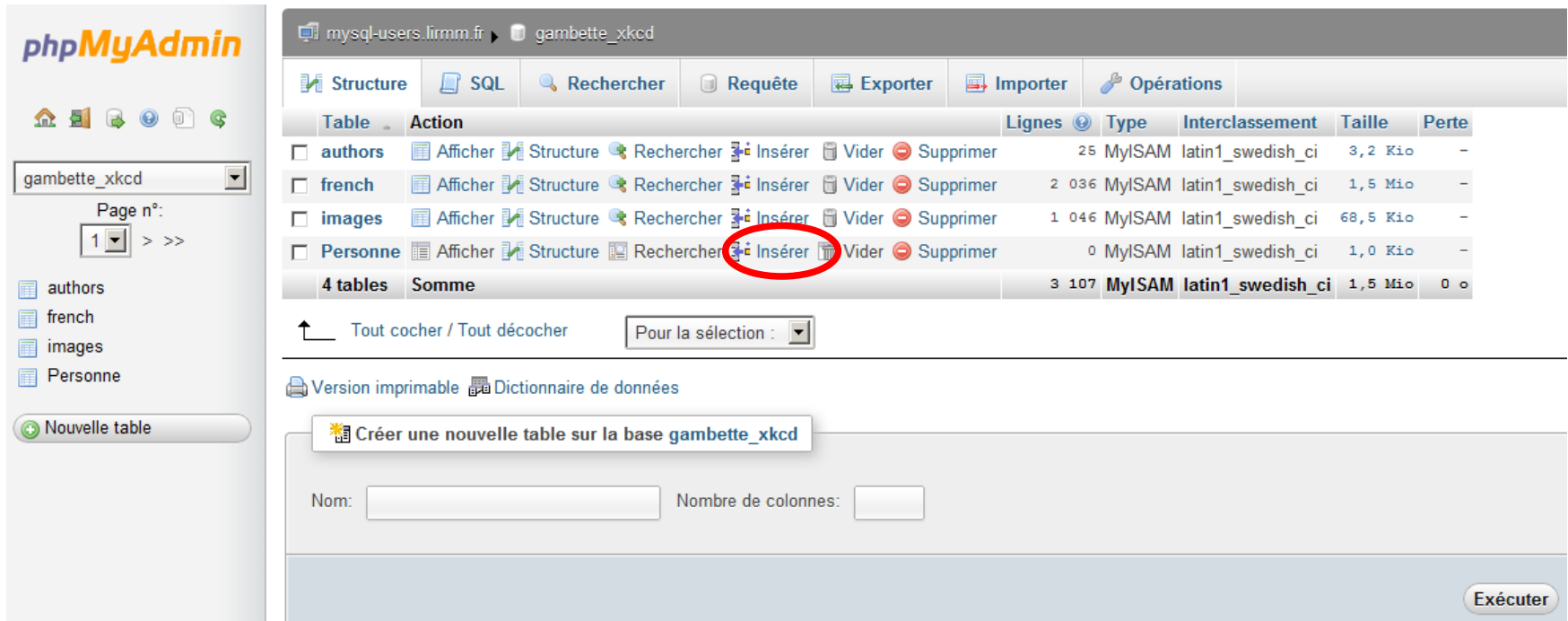
- Résumé des épisodes précédents
- Introduction au langage SQL
- Langage de définition des données
- **Langage de manipulation des données**
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions

Langage de manipulation des données

Les commandes principales sont :

- INSERT pour **ajouter** les données
- SELECT pour **consulter** les données
- UPDATE pour **modifier** les données
- DELETE pour **supprimer** les données

Insertion d'occurrences dans une table avec PhpMyAdmin



The screenshot shows the phpMyAdmin interface for a MySQL database named 'gambette_xkcd'. The main table list is displayed with columns for 'Table', 'Action', 'Lignes', 'Type', 'Interclassement', 'Taille', and 'Perte'. The 'Personne' table is selected, and its 'Insérer' button is circled in red. Below the table list, there are options for 'Tout cocher / Tout décocher' and 'Pour la sélection :'. A dialog box titled 'Créer une nouvelle table sur la base gambette_xkcd' is open, showing input fields for 'Nom:' and 'Nombre de colonnes:'. An 'Exécuter' button is visible at the bottom right of the dialog.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> authors	Afficher Structure Rechercher Insérer Vider Supprimer	25	MyISAM	latin1_swedish_ci	3,2 Kio	-
<input type="checkbox"/> french	Afficher Structure Rechercher Insérer Vider Supprimer	2 036	MyISAM	latin1_swedish_ci	1,5 Mio	-
<input type="checkbox"/> images	Afficher Structure Rechercher Insérer Vider Supprimer	1 046	MyISAM	latin1_swedish_ci	68,5 Kio	-
<input type="checkbox"/> Personne	Afficher Structure Rechercher Insérer Vider Supprimer	0	MyISAM	latin1_swedish_ci	1,0 Kio	-
4 tables	Somme	3 107	MyISAM	latin1_swedish_ci	1,5 Mio	0

Version imprimable Dictionnaire de données

Créer une nouvelle table sur la base gambette_xkcd

Nom: Nombre de colonnes:

Exécuter

Insertion d'occurrences dans une table avec PhpMyAdmin

phpMyAdmin

mysql-users.lirmm.fr ▶ gambette_xkcd ▶ Personne

Afficher Structure SQL Rechercher Insérer Exporter Importer Opérations

Colonne	Type	Fonction	Null	Valeur
Id	int(11)			
Prenom	varchar(100)			Philippe
Nom	varchar(100)			Gambette
Adresse	varchar(255)			49T rue Haguenot

Exécuter

Ignorer

Colonne	Type	Fonction	Null	Valeur
Id	int(11)			
Prenom	varchar(100)			
Nom	varchar(100)			
Adresse	varchar(255)			

Exécuter

Insertion d'occurrences dans une table avec PhpMyAdmin

The screenshot displays the PhpMyAdmin interface for a MySQL database. The top navigation bar shows the path: `mysql-users.lirmm.fr` > `gambette_xkcd` > `Personne`. Below this, a menu contains options: Structure, SQL, Rechercher, Requête, Exporter, Importer, and Opérations. A green notification bar at the top center states: "1 ligne insérée. Identifiant de la ligne insérée : 1". The main area shows the SQL query: `INSERT INTO `gambette_xkcd`.`Personne` (`id`, `Prenom`, `Nom`, `Adresse`) VALUES (NULL, 'Philippe', 'Gambette', '49T rue Haquenot');`. Below the query, a table summary table is visible.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> authors	Afficher Structure Rechercher Insérer Vider Supprimer	25	MyISAM	latin1_swedish_ci	3,2 Kio	-
<input type="checkbox"/> french	Afficher Structure Rechercher Insérer Vider Supprimer	2 036	MyISAM	latin1_swedish_ci	1,5 Mio	-
<input type="checkbox"/> images	Afficher Structure Rechercher Insérer Vider Supprimer	1 046	MyISAM	latin1_swedish_ci	68,5 Kio	-
<input type="checkbox"/> Personne	Afficher Structure Rechercher Insérer Vider Supprimer	1	MyISAM	latin1_swedish_ci	2,0 Kio	-
4 tables	Somme	3 108	MyISAM	latin1_swedish_ci	1,5 Mio	0

At the bottom of the table summary, there are controls: "↑ Tout cocher / Tout décocher" and a dropdown menu "Pour la sélection :".

At the very bottom, there are links for "Version imprimable" and "Dictionnaire de données".

Langage de manipulation des données - INSERT

Insérer des données dans une table :

```
INSERT [INTO] tbl_name [(col_name, ...)]  
VALUES ({expr | DEFAULT}, ...)
```

liste des champs

liste des valeurs des champs

Le nombre de `col_name` doit correspondre au nombre d'`expr`.

Le fait de préciser les champs est optionnel :

- s'il est indiqué, il faut indiquer les valeurs dans le même ordre que les champs
- s'il n'est pas indiqué, il faut donner les valeurs de chaque champs dans l'ordre des champs stockés dans la table

Pour les champs ayant l'option `AUTO_INCREMENT`, il est possible :

- soit de ne pas préciser le champ dans la liste,
- soit de passer la valeur `NULL`.

Le système se chargera d'attribuer automatiquement une valeur.

Langage de manipulation des données - SELECT

Lire des données dans une ou plusieurs tables :

```
SELECT [DISTINCT] select_expression, ...
FROM table references
    [WHERE where_definition]
    [ORDER BY {unsigned_integer | nom_de_colonne}
            [ASC | DESC] , ...]
    [LIMIT [offset,] lignes]
```

`select_expression` indique la colonne à lire, une constante, ou une valeur calculée. Par exemple, écrire `concat(champ1, champ2) AS new` signifie qu'on va récupérer un nouveau champ virtuel `new` qui contient la valeur du champ `champ1` suivie de la valeur du champ `champ2`.

Si deux champs ont le même nom `champ1` dans plusieurs tables `table1` et `table2`, on les désigne par `table1.champ1` et `table2.champ1`.

Le `DISTINCT` permet de ne lire que des valeurs distinctes.

Le `FROM` permet de lister les tables à utiliser dans la recherche des données.

Le `ORDER BY` permet de trier le résultat de la requête (`ASC` : croissant, `DESC` : décroissant).

Langage de manipulation des données - SELECT

Le `WHERE` permet de préciser les critères de recherche et d'associer les tables entre elles.

Tous les opérateurs `=`, `<=>`, `<`, `>`, `!=`, `>=`, `<=`, `<>`, `BETWEEN`, `IN`, `NOT IN`, `IS NULL`, `IS NOT NULL`, ... sont supportés.

On peut faire des critères de sélection « *avec joker* » sur les chaînes de caractères grâce au mot-clé `LIKE`.

Le critère :

```
champ1 LIKE '%a%'
```

signifie que le champ `champ1` est une chaîne de caractères qui contient :

n'importe quoi suivi d'un « *a* » suivi de n'importe quoi

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

Personne	
<u>id</u>	<u>int</u>
nom	varchar(30)
prenom	varchar(30)
adress#	int

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

```
SELECT nom FROM Personne  
WHERE prenom = 'Lisa' ORDER BY 1
```

On désire lire tous les noms et prénoms associés dans un champ séparés par un espace.

Personne	
<u>id</u>	<u>int</u>
nom	varchar(30)
prenom	varchar(30)
adress#	int

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

```
SELECT nom FROM Personne  
WHERE prenom = 'Lisa' ORDER BY 1
```

On désire lire tous les noms et prénoms associés dans un champ séparés par un espace.

```
SELECT concat(nom, ' ', prenom) as pers  
FROM Personne ORDER BY 1
```

On désire lire les ID de toutes les personnes ayant une adresse renseignée.

Personne	
<u>id</u>	<u>int</u>
nom	varchar(30)
prenom	varchar(30)
adress#	int

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

```
SELECT nom FROM Personne  
WHERE prenom = 'Lisa' ORDER BY 1
```

On désire lire tous les noms et prénoms associés dans un champ séparés par un espace.

```
SELECT concat(nom, ' ', prenom) as pers  
FROM Personne ORDER BY 1
```

On désire lire les ID de toutes les personnes ayant une adresse renseignée.

```
SELECT id FROM Personne  
WHERE adress IS NOT NULL
```

Personne	
<u>id</u>	<u>int</u>
nom	varchar(30)
prenom	varchar(30)
adress#	int

Langage de manipulation des données - SELECT

Lire des données en combinant les informations de plusieurs tables :

Pour chercher des données contenues dans une table ainsi que dans une autre table liées par le biais d'une clé étrangère, il est **indispensable de préciser l'égalité entre les 2 champs.**

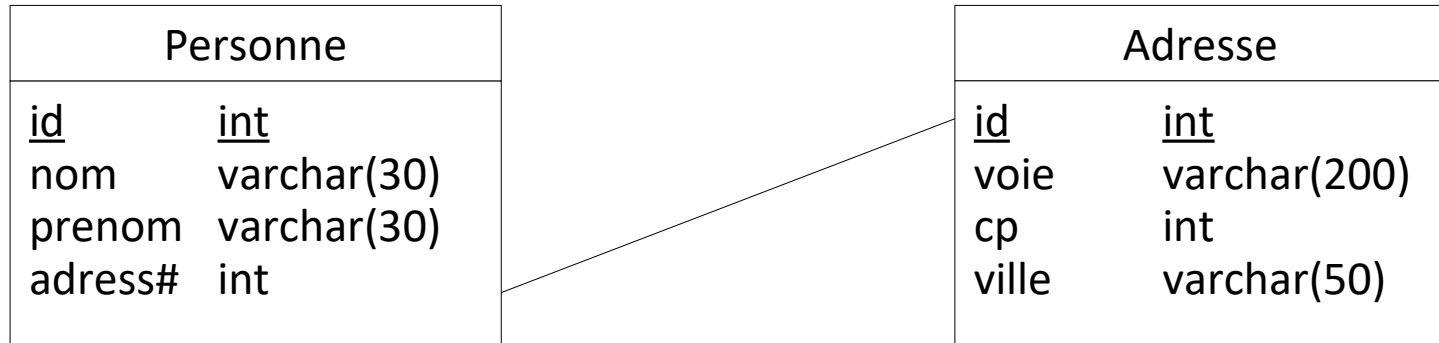
Si on ne précise pas que la clé étrangère de la table `table1` correspond à la clé primaire de la table `table2`, ce sont toutes les possibilités d'associations de lignes de `table1` avec toutes les lignes de la table `table2` qui seront listées par la requête ! Il faut donc **préciser que la clé étrangère de la table `table1` correspond à la clé primaire de la table `table2`.**

Par exemple, pour sélectionner les clés primaires des objets de la table `table2` liés aux objets de la table `table1` dont l'attribut `nom` est "toto", on écrit :

```
SELECT table2.cleprimaire
FROM table1,table2
WHERE table1.cle_etrangere=table2.cle_primaire
AND table1.nom='toto'
```

Langage de manipulation des données - SELECT

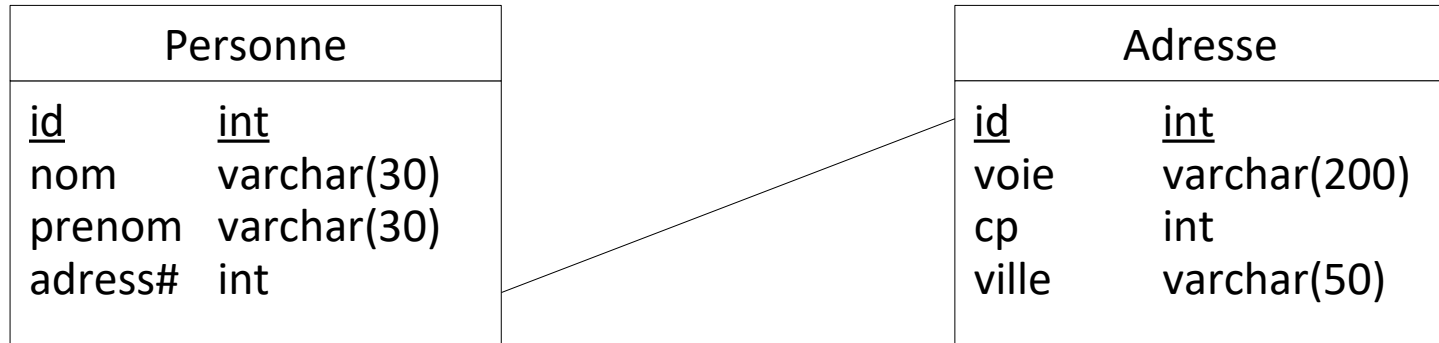
Exemple



Sélectionner le nom et l'adresse des personnes dont le nom commence par Simps :

Langage de manipulation des données - SELECT

Exemple



Sélectionner le nom et l'adresse des personnes dont le nom commence par Simps :

```
SELECT Personne.nom, Adresse.voie  
FROM Personne, Adresse  
WHERE Personne.adress = Adresse.id  
AND Personne.nom LIKE 'Simps%'
```

Langage de manipulation des données - UPDATE

Modifier des données dans une table :

```
UPDATE tbl_name  
SET col_name1=expr1 [,col_name2=expr2 ...]  
[WHERE where_definition] [LIMIT row_count]
```

Le `SET` permet d'attribuer une nouvelle valeur au champ.

Il est possible de mettre à jour plusieurs champs en même temps.

Le `WHERE` permet de préciser quelles données on désire mettre à jour.

Son fonctionnement sera détaillé avec la commande `SELECT`.

Sans clause `WHERE`, toutes les données de la table sont mises à jour.

La `LIMIT` permet de limiter le nombre de lignes à modifier.

Langage de manipulation des données

Lien entre requêtes dans un INSERT

Il est possible d'insérer dans une table des données issues d'une autre requête.

```
INSERT [INTO] tbl_name [(col_name, ...)]  
    SELECT ...
```

Lien entre tables dans un UPDATE

Il est possible de mettre à jour des données en fonction de données d'autres tables :

```
UPDATE tbl_name [, tbl_name ...]  
    SET col_name1=expr1 [, col_name2=expr2 ...]  
    [WHERE where_definition]
```

C'est toujours la table dont le nom est accolé au mot UPDATE qui est mise à jour.

Langage de manipulation des données - DELETE

Supprimer des données dans une table :

```
DELETE FROM table_name  
[WHERE where_definition] [LIMIT row_count]
```

Le `WHERE` permet de préciser quelles données on désire supprimer.

Sans clause `WHERE`, toutes les données de la table sont supprimées. On préfère alors utiliser la commande spéciale `TRUNCATE TABLE`.

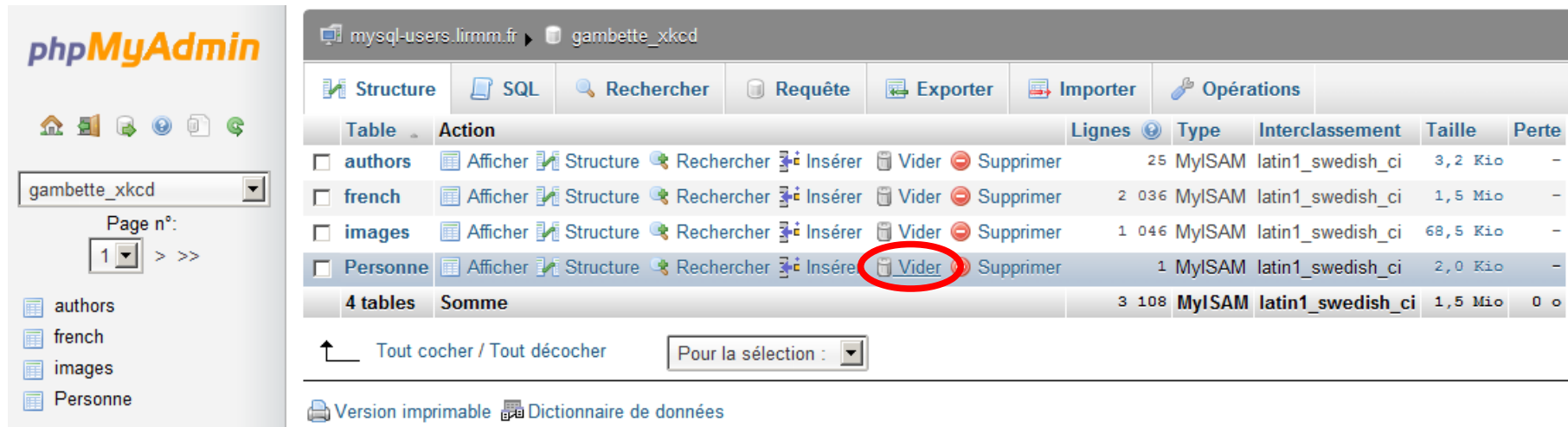
Langage de manipulation des données - DELETE

Supprimer des données dans une table :

```
DELETE FROM table_name  
[WHERE where_definition] [LIMIT row_count]
```

Le WHERE permet de préciser quelles données on désire supprimer.

Sans clause WHERE, toutes les données de la table sont supprimées. On préfère alors utiliser la commande spéciale TRUNCATE TABLE.



The screenshot shows the phpMyAdmin interface for a MySQL database named 'gambette_xkcd'. The left sidebar displays the database name and a list of tables: 'authors', 'french', 'images', and 'Personne'. The main area shows a table list with columns for 'Table', 'Action', 'Lignes', 'Type', 'Interclassement', 'Taille', and 'Perte'. The 'Personne' table is selected, and its 'Vider' button is circled in red. Below the table list, there are options for 'Tout cocher / Tout décocher' and 'Pour la sélection :'. At the bottom, there are links for 'Version imprimable' and 'Dictionnaire de données'.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> authors	Afficher Structure Rechercher Insérer Vider Supprimer	25	MyISAM	latin1_swedish_ci	3,2 Kio	-
<input type="checkbox"/> french	Afficher Structure Rechercher Insérer Vider Supprimer	2 036	MyISAM	latin1_swedish_ci	1,5 Mio	-
<input type="checkbox"/> images	Afficher Structure Rechercher Insérer Vider Supprimer	1 046	MyISAM	latin1_swedish_ci	68,5 Kio	-
<input checked="" type="checkbox"/> Personne	Afficher Structure Rechercher Insérer Vider Supprimer	1	MyISAM	latin1_swedish_ci	2,0 Kio	-
4 tables	Somme	3 108	MyISAM	latin1_swedish_ci	1,5 Mio	0 o

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- **SQL avancé : les jointures**
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Jointures

Utilisation des jointures

→ Sélectionner les données se trouvant dans plusieurs tables.

→ Préciser les données sur lesquelles travailler lors d'un :

- `SELECT` (lecture)
- `UPDATE` (mise à jour)
- `DELETE` (suppression)

Jointures

Utilisation des jointures

→ Sélectionner les données se trouvant dans plusieurs tables.

→ Préciser les données sur lesquelles travailler lors d'un :

- `SELECT` (lecture)
- `UPDATE` (mise à jour)
- `DELETE` (suppression)

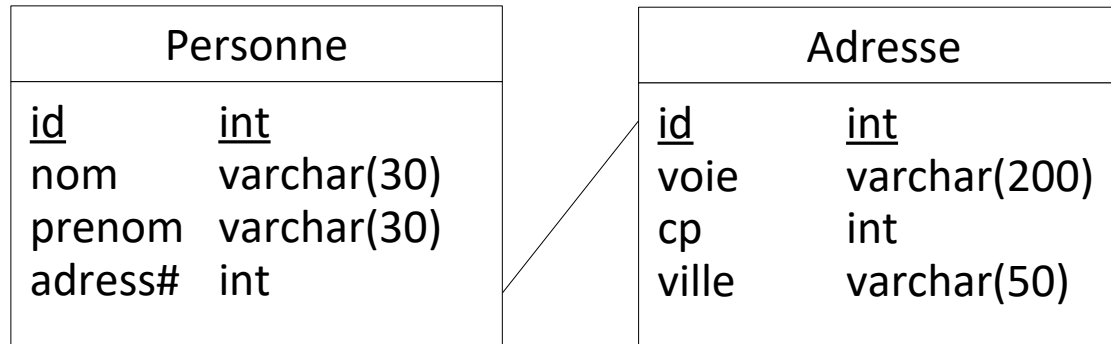
Principe

Une jointure a lieu **entre deux tables**. Elle exprime une correspondance entre deux clés par un **critère d'égalité**.

Si les données à traiter se trouvent **dans trois tables**, la correspondance entre les trois tables s'exprime par **deux égalités**.

Jointures

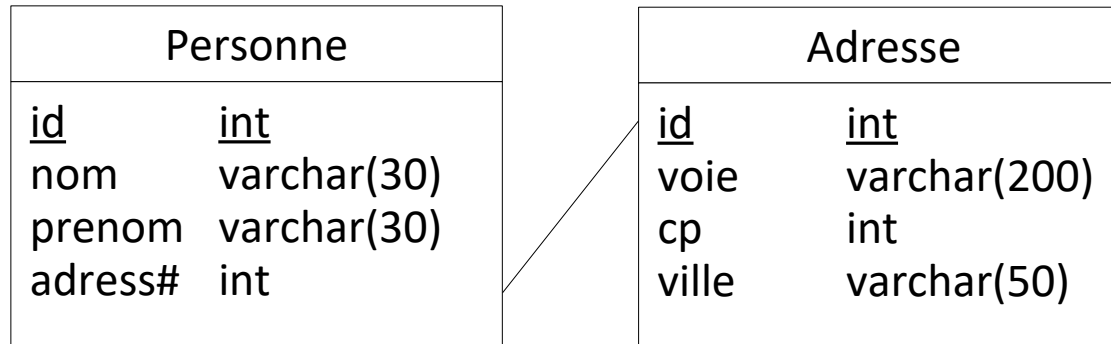
Exemple



Pour lire l'adresse correspondant à la personne, il faut écrire :

Jointures

Exemple

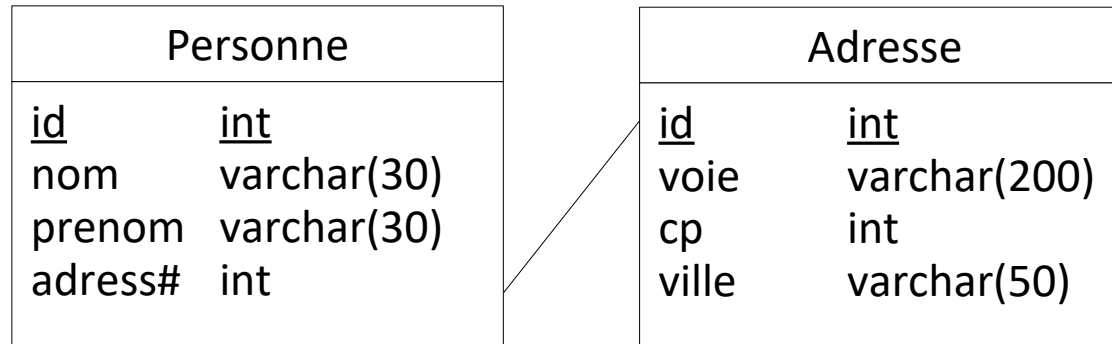


Pour lire l'adresse correspondant à la personne, il faut écrire :

```
SELECT * FROM Personne, Adresse  
WHERE Personne.adress = Adresse.id
```

Jointures

Exemple



Pour lire l'adresse correspondant à la personne, il faut écrire :

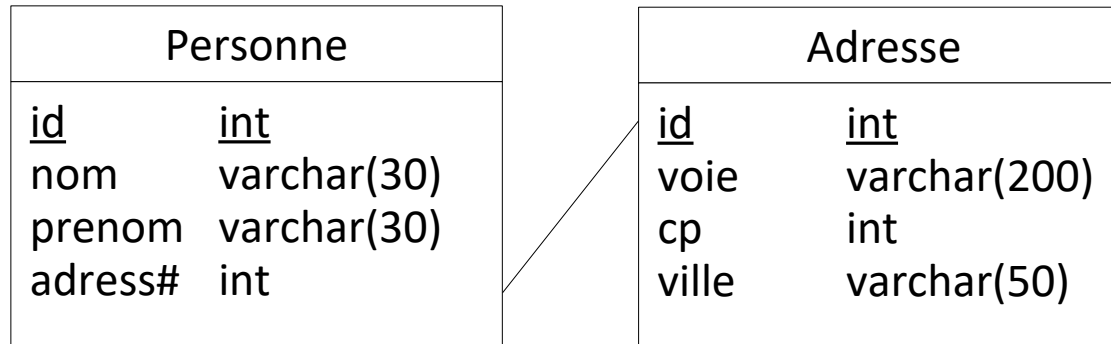
```
SELECT * FROM Personne, Adresse  
WHERE Personne.adress = Adresse.id
```

Attention : dans la clause `WHERE` se mélangent les associations entre les tables et les conditions de sélection des données. Ne pas les confondre !

```
SELECT * FROM Personne, Adresse  
WHERE Personne.adress = Adresse.id  
AND Personne.nom = 'Durand'
```

Jointures

Exemple



Pour lire l'adresse correspondant à la personne, il faut écrire :

```
SELECT * FROM Personne, Adresse
WHERE Personne.adress = Adresse.id
```

Attention : dans la clause WHERE se mélangent les associations entre les tables et les conditions de sélection des données. Ne pas les confondre !

```
SELECT * FROM Personne, Adresse
WHERE Personne.adress = Adresse.id
AND Personne.nom = 'Durand'
```

jointure pour associer les deux tables

critère de sélection

Remarque : ordre sans importance

Jointures fermées et ouvertes

Problèmes de la jointure par = :

- Mélange des critères de sélection et des jointures
- Mise en relation des données uniquement quand les deux attributs sont remplis (jointure **fermée**)

Jointures fermées et ouvertes

Problèmes de la jointure par = :

- Mélange des critères de sélection et des jointures
- Mise en relation des données uniquement quand les deux attributs sont remplis (jointure **fermée**)

Exemple :

On désire lire toutes les personnes et accessoirement donner leur adresse si celle-ci est connue.

La requête :

```
SELECT * FROM Personne, Adresse  
WHERE Personne.adress = Adresse.id
```

ne retournera pas les personnes n'ayant pas d'adresse référencée.

Jointures fermées et ouvertes

Jointure JOIN

L'association se fait directement entre les tables en précisant les colonnes concernées.

```
SELECT * FROM table1 INNER JOIN table2 ON  
table1.cle_primaire = table2.cle_etrangere
```

Exemple

```
SELECT * FROM Personne, Adresse  
WHERE Personne.adress = Adresse.id
```

équivalent à :

```
SELECT * FROM Personne  
INNER JOIN Adresse ON Personne.adress = Adresse.id
```

Jointures fermées et ouvertes

Il existe **trois types d'associations** :

- `INNER JOIN` : jointure fermée, les données doivent être à la fois dans les 2 tables
- `LEFT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- `RIGHT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

Jointures fermées et ouvertes

Il existe **trois types d'associations** :

- `INNER JOIN` : jointure fermée, les données doivent être à la fois dans les 2 tables
- `LEFT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- `RIGHT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

Exemple

Lire toutes les personnes et accessoirement donner leur adresse si celle-ci est connue.

Jointures fermées et ouvertes

Il existe **trois types d'associations** :

- `INNER JOIN` : jointure fermée, les données doivent être à la fois dans les 2 tables
- `LEFT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- `RIGHT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

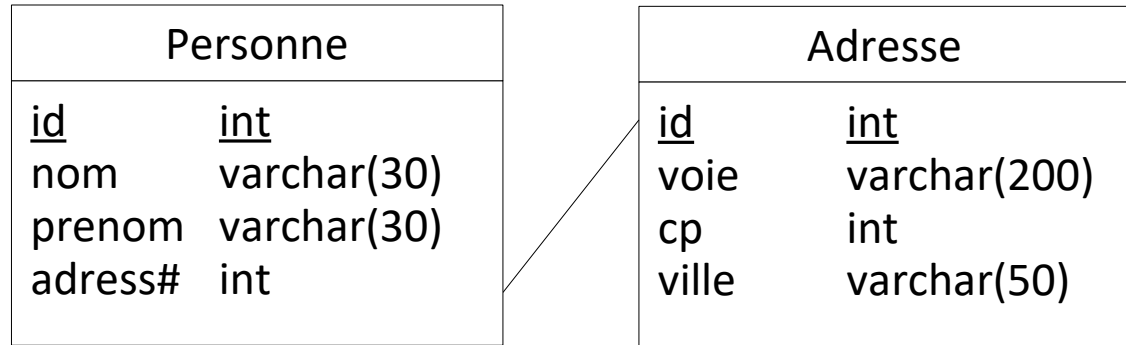
Exemple

Lire toutes les personnes et accessoirement donner leur adresse si celle-ci est connue.

```
SELECT * FROM Personne LEFT OUTER JOIN Adresse
```

Les personnes pour lesquelles l'adresse n'est pas connue auront les champs de la table Adresse à NULL.

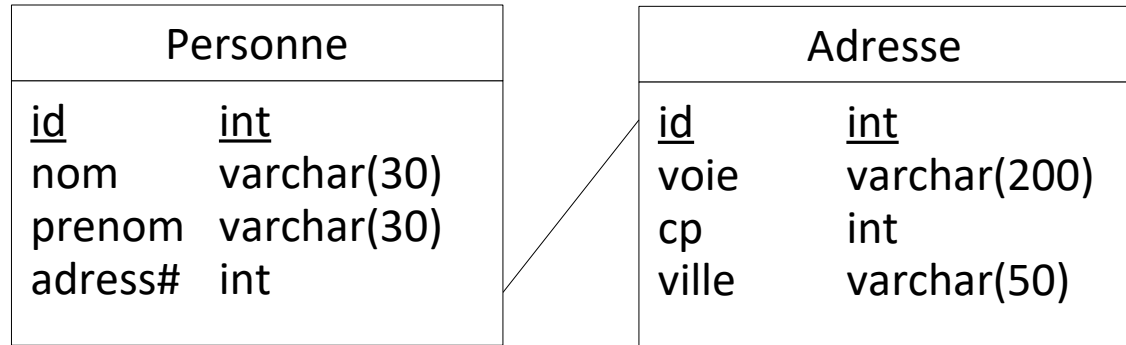
Jointures fermées et ouvertes



Exemples avec critère

Lire toutes les personnes qui s'appellent Durand et accessoirement donner leur adresse si celle-ci est connue.

Jointures fermées et ouvertes



Exemples avec critère

Lire toutes les personnes qui s'appellent Durand et accessoirement donner leur adresse si celle-ci est connue.

```
SELECT * FROM Personne LEFT JOIN adresse
ON Personne.Adress=Adresse.ID
WHERE Personne.Nom = 'Durand'
```

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- **SQL avancé : les groupements**
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Groupements

Utilisation des groupements

→ effectuer des opérations sur un ensemble de données :

- MIN (retourne le minimum)
- MAX (retourne le maximum)
- COUNT (retourne le nombre)
- SUM (retourne la somme)

Afin de préciser au moteur SQL que cette opération porte sur une sélection de données, il faut préciser la clause `GROUP BY`.

Groupements

Utilisation des groupements

→ effectuer des opérations sur un ensemble de données :

- MIN (retourne le minimum)
- MAX (retourne le maximum)
- COUNT (retourne le nombre)
- SUM (retourne la somme)

Afin de préciser au moteur SQL que cette opération porte sur une sélection de données, il faut préciser la clause `GROUP BY`.

Exemple

Compter le nombre de valeurs de `champ1` :

```
SELECT champ2, COUNT(champ1) FROM table1  
GROUP BY champ2
```

Groupements

Pour créer des critères de sélection qui **portent sur un ensemble de données**,
→ associer la clause `GROUP BY` à la clause `HAVING` (ou `NOT HAVING`).

Exemple

Sélectionner les données dont le nombre de répétitions est supérieur à n .

```
SELECT * FROM table1 GROUP BY champ2  
HAVING count(champ2) > n
```

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- **SQL avancé : les transactions**
- SQL avancé : l'intégrité référentielle

Transactions

En mode classique, les requêtes s'enchaînent. La première peut fonctionner alors que la suivante peut rencontrer une erreur. La base de données contient alors des données dans certaines tables et pas dans d'autres.

Transactions

En mode classique, les requêtes s'enchaînent. La première peut fonctionner alors que la suivante peut rencontrer une erreur. La base de données contient alors des données dans certaines tables et pas dans d'autres.

Pour éviter cela on utilise des **transactions** (blocs de requêtes SQL) :

`START TRANSACTION` (pour ouvrir la transaction)

[Liste de requêtes SQL]

`COMMIT TRANSACTION`

ou

`ROLLBACK TRANSACTION`

Confirme et exécute l'ensemble
des requêtes SQL

Annule l'ensemble des requêtes
SQL

Transactions

En mode classique, les requêtes s'enchaînent. La première peut fonctionner alors que la suivante peut rencontrer une erreur. La base de données contient alors des données dans certaines tables et pas dans d'autres.

Pour éviter cela on utilise des **transactions** (blocs de requêtes SQL) :

START TRANSACTION (pour ouvrir la transaction)

[Liste de requêtes SQL]

COMMIT TRANSACTION

ou

ROLLBACK TRANSACTION

Confirme et exécute l'ensemble des requêtes SQL

Annule l'ensemble des requêtes SQL

Verrou et dead-lock

Lorsqu'une modification sur une table est en cours, les données sont verrouillées en lecture et en écriture.

→ Situation de verrouillages mutuels entre deux transactions : **dead-locks**.

Repérés par le SGDB qui émet un rollback sur l'une des transactions.

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Intégrité référentielle

Rappel :

Dans un modèle physique de données, les tables sont liées entre elles par le biais d'une clé étrangère.

La clé étrangère d'une table permet de lier la table à la clé primaire de l'autre table.

Lors de la sélection des données, l'association entre les deux tables est effectuée grâce à une jointure entre ces deux tables.

Personne

<u>id</u>	nom	prenom	id_adresse
1	Durand	Marie	4
2	Simpson	Bart	3
3	Dubois	Jean	NULL
4	Simpson	Lisa	3

Adresse

<u>id</u>	voie	cp	ville
1	1, rue Ici	75002	Paris
2	12, rue labas	75015	Paris
3	742 Evergreen Terrasse		Springfield
4	Chemin perdu	66000	Perpignan

Chaque référence à une adresse est respectée.

Intégrité référentielle

Rappel :

Dans un modèle physique de données, les tables sont liées entre elles par le biais d'une clé étrangère.

La clé étrangère d'une table permet de lier la table à la clé primaire de l'autre table.

Lors de la sélection des données, l'association entre les deux tables est effectuée grâce à une jointure entre ces deux tables.

Personne

<u>id</u>	nom	prenom	id_adresse
1	Durand	Marie	4
2	Simpson	Bart	3
3	Dubois	Jean	NULL
4	Simpson	Lisa	3

Adresse

<u>id</u>	voie	cp	ville
1	1, rue Ici	75002	Paris
2	12, rue labas	75015	Paris
3	742 Evergreen Terrasse		Springfield
4	Chemin perdu	66000	Perpignan

Si on supprime l'occurrence 4 de la table Adresse, il devient impossible de retrouver l'adresse de Marie Durand

Intégrité référentielle

Lors de la création des tables, il est possible de confier ce contrôle à la base de données.

Le fonctionnement désiré devra être précisé lors de la création des clés externes.

Note :

Sur MySQL, cette fonctionnalité n'est disponible qu'avec le moteur des tables : InnoDB.

Lors d'un create table, il faut ajouter une clause

```
FOREIGN KEY (champ1, [..., champN])  
REFERENCES table(champ1, [..., champN])  
ON UPDATE action  
ON DELETE action
```

Intégrité référentielle

```
Lors d'un CREATE TABLE, il faut ajouter une clause  
FOREIGN KEY (champ1, [..., champN])  
REFERENCES table(champ1, [..., champN])  
ON UPDATE action  
ON DELETE action
```

Les actions possibles sont :

- RESTRICT : si une référence est trouvée, la suppression ou la modification sera interdite.
- SET NULL : si une référence est trouvée, la suppression ou la modification aura pour effet en plus de l'action de mettre à jour la référence avec la valeur NULL.
- CASCADE : si une référence est trouvée, la suppression ou la modification aura pour effet en plus de l'action d'effectuer la même opération sur les données trouvées.
- NO ACTION : pas de contrôle d'intégrité référentielle.