DUT MMI – IUT de Marne-la-Vallée

# *M2203 – Bases de données*

**Philippe Gambette**
**Thanh-Long Dang**

Page web du cours : http://tinyurl.com/M2203-2014

- **Contact**

- Courriel : philippe.gambette@gmail.com
  (M2203 doit apparaître dans le sujet du courriel).
- Avant ou après le cours.
- Possibilité de poser des questions, de demander des exercices supplémentaires d'entrainement.

- **Enseignants**

Cours par Philippe Gambette, TD & TP par Thanh-Long Dang

- **Notes et devoirs**

- QCM (… le retour) : **nécessité de relire le cours au moins à chaque veille de cours et de TP/TD**
- éventuellement une note de travail à la maison
- devoir final le 11 juin a priori (tous documents autorisés)

---

- **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

# Introduction aux bases de données

**Base de données**
Une **base de données** est un lot d'informations stocké dans un dispositif informatique.

**Système de gestion de bases de données**
Un **système de gestion de bases de données** (SGBD) est un module informatique chargé de gérer les données en en permettant la création, la modification, la suppression et la lecture.

**Objectifs de la conception** d'une base de données :
- **indépendance** : la BD est un module dissocié du système d'information, le format des données est indépendant du système.
- **accès** : la BD gère les accès aux données en gérant les accès concurrentiels
- **cohérence** : la BD assure l'intégrité des données.
- **sécurité** : la BD gère les accès aux données en fonctions des utilisateurs.
- **administration** : la BD peut être administrée au sauvegardée de façon autonome.

**Exemples** de petites bases de données :

**Critiques de films**

Avis cinéma de Monique Pantel (http://monique.pantel.free.fr)

Films, articles.

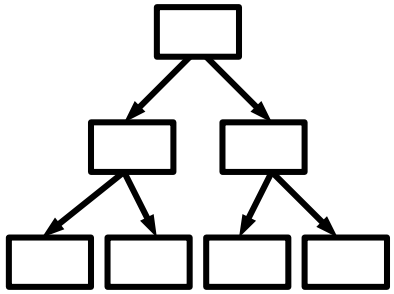
**Livre géolocalisé interactif**

Lisbonne par Fernando Pessoa (http://lisbon.pessoa.free.fr)

Lieux, mots du texte.

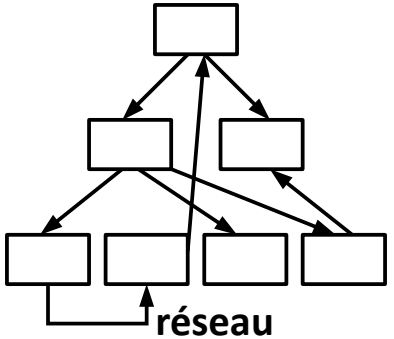
# Différents types de bases de données

**Base hiérarchique**
Lie les enregistrements dans une structure arborescente où chaque enregistrement n'a qu'un seul possesseur.


hiérarchique

**Base en réseau**
Est une base hiérarchique mais permet en plus d'établir des relations transverses.

**Base relationnelle**
Stocke les informations décomposées et organisées dans des matrices appelées relations ou tables.

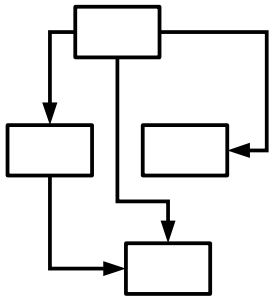

réseau

**Base objet**
Stocke les informations groupées sous forme de collections d'objets persistants.

**Base XML**
S'appuie sur le modèle de données fourni par XML.

Dans le Top3 des chercheurs en informatique français les plus cités, Serge Abiteboul : 30738 citations, 68 articles cités plus de 68 fois.

http://www.cs.ucla.edu/~palsberg/h-number.html


relationnelle

# Le modèle relationnel

**L'entité**
Concept concret ou abstrait du monde à modéliser. Elle se représente par un cadre contenant son nom.

**L'attribut**
Donnée élémentaire qui sert à caractériser les entités et les associations. Les attributs sont listés dans l'entité.

**L'identifiant (ou clé)**
Attribut(s) particulier(s) permettant d'identifier chaque occurrence d'une entité. Les attributs servant d'identifiant sont soulignés.

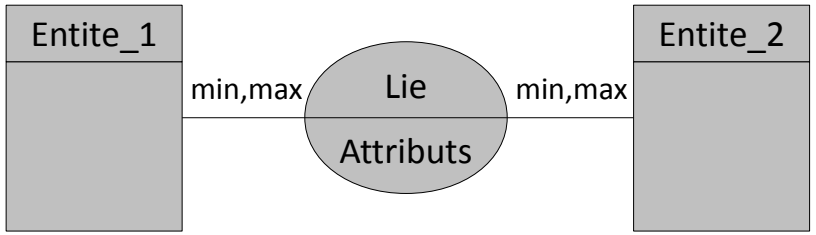
**L'occurrence**
Élément particulier d'une entité ou d'une association. L'occurrence ne fait pas partie du modèle E-A mais est expliquée ici pour mieux comprendre l'entité.

| **Nom** | | | |
|---|---|---|---|
| <u>Identifiant</u> | Champ_1 | Champ_2 | Champ_N |
| id_val_a | ch_1_a | ch_2_a | ch_N_a |
| id_val_b | ch_1_b | ch_2_b | ch_N_b |
| id_val_c | ch_1_c | ch_2_c | ch_N_c |
| id_val_d | ch_1_d | ch_2_d | ch_N_d |
| id_val_e | ch_1_e | ch_2_e | ch_N_e |

# Le modèle relationnel

**L'association binaire**

Permet de relier deux entités entre elles.
Elle se représente par le biais d'un ovale ou d'un losange contenant son nom et ses éventuels attributs.
Leur nom est généralement un verbe.




**Les cardinalités**

Couple de valeurs indiqué à l'extrémité de chaque lien d'une association.
La cardinalité minimum indique le caractère optionnel (0) ou obligatoire (1) de la relation.
La cardinalité maximum indique le caractère unique (1) ou multiple (*n*) de la relation.

Couples de cardinalités possibles :

| **Card.** | **Lecture** |
|---|---|
| 0,1 | Lien vers 0 ou 1 |
| 1,1 | Lien vers 1 |
| 0,*n* | Lien vers 0 ou *n* |
| 1,*n* | Lien vers 1 ou *n* |

Associations selon les cardinalités maximum :

| **Entite_1** | **Entite_2** | **Lecture** |
|---|---|---|
| 1 | 1 | association 1 à 1 |
| 1 | *n* | association 1 à plusieurs |
| *n* | 1 | association 1 à plusieurs |
| *n* | *n* | association plusieurs à plusieurs |

# Choix de modélisation

**Conception du modèle**

Il n'existe **pas de modèle de données idéal**.

Le modèle doit correspondre à un **besoin précis**.

Il est indispensable que chaque décision, chaque façon de faire, soit réfléchie.

En cas de multiples possibilités, il faut s'assurer que le fonctionnement mis en place répondra au besoin.

**Interprétation**

Une base données doit permettre de stocker toutes les informations nécessaires à son utilisation.

Toute la complexité réside dans l'organisation de ces attributs.

Toute **redondance** est **interdite**.

Il faut essayer de créer un modèle à la fois **évolutif** mais aussi **suffisant** pour le besoin.

Un choix difficile est notamment le fait d'utiliser un attribut dans l'entité ou de créer une association.

Le choix des cardinalités est également primordial.

**Usages**

Pour nommer les entités et les attributs, il est **interdit d'utiliser espaces et accents**.

Les relations sont nommées par des **verbes** à la forme active ou passive.

Attention à la **casse**, certaines bases de données y sont sensibles, d'autres non. Il est donc fréquent de devoir appliquer une normalisation propre à l'entreprise.

Nous appliquerons dans ce cours pour les attributs et entités la règle : **première lettre majuscule puis minuscules**.

**Aberrations**

Toute répétition d'entité doit être supprimée.

Les relations binaires un pour un ne doivent pas apparaître dans un modèle E-A mais nous verrons qu'elles existent fréquemment dans un modèle de données.

Les relations ***n*-aires** sont souvent **complexes à comprendre et à interpréter**. Il faut donc toujours se demander si elles sont indispensables.

# Exercice 1 – Le restaurant

Un restaurant veut pouvoir gérer son stock de façon automatique en fonction des commandes passées. Ainsi chaque plat de la carte est associé aux ingrédients. Lorsqu'un client commande un plat, on doit donc pouvoir en déduire les ingrédients consommés et ainsi connaître le stock.

Q1. De quelles informations avons-nous besoin pour gérer la demande ?

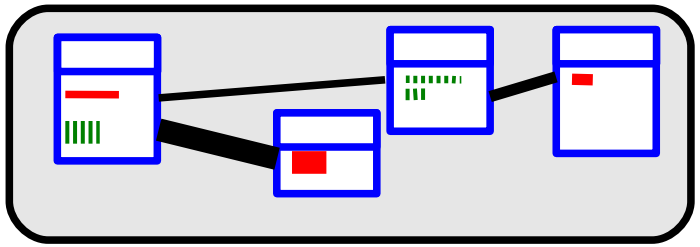
Q2. Quelles sont les entités ? Quels sont les attributs ?

Q3. Dessinez le schéma entité-association qui correspond à un système de gestion automatique du stock d'un restaurant

# Exercice 1 – Le restaurant (suite)

# Modèle logique des données

**Modèle logique des données :**
constitué de …
constituées d'attributs, parmi lesquels :



• une …
→ **identifie de manière unique** chaque occurrence de la table.

• éventuellement une ou plusieurs …
clés primaires dans une autre table
→ les clés étrangères créent des **liens entre tables**

**Transition du modèle entité-association au modèle logique des données :**

| | | |
|---|---|---|
| Entité | → | Table |
| Identifiant | → | Clé primaire |
| Association 1 à 1 | → | Clés dans la “table à 1” |
| Association 1 à plusieurs | → | Clé étrangère dans la “table à 1” |
| Association plusieurs à plusieurs | → | Table supplémentaire avec deux clés étrangères |
| Association *n*-aire | → | Table supplémentaire avec *n* clés étrangères |

# Transformation vers le modèle logique des données

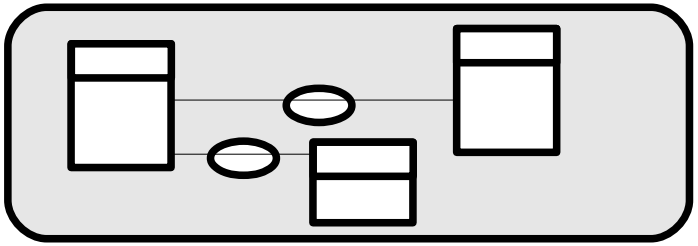
**Exemples :**

**Modèle entité-association**

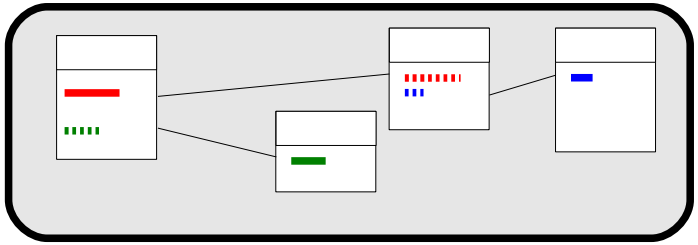
**Modèle logique des données**

# Du MCD au MLD au MPD

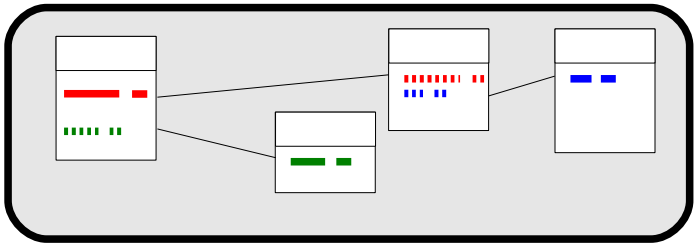
Modèle entité-association
(modèle conceptuel des données)



Modèle logique des données



Modèle physique des données



# Modèle physique des données

**Modèle physique des données** : modèle logique des données avec **attributs typés**
Les types de données peuvent varier selon les systèmes de gestion de bases de données.

**Champs numériques**

| Type | Val min | Val max |
|---|---|---|
| BIT | 0 | 1 |
| TINYINT | -128 | 127 |
| BOOL | TRUE | FALSE |
| SMALLINT | -32768 | 32767 |
| MEDIUMINT | -8388608 | 8388607 |
| INT | -2147483648 | 2147483647 |
| BIGINT | -9,22337E+18 | 9,22337E+18 |
| SERIAL | BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE | FLOAT |
| FLOAT | -3.402823466E+38 | -1.175494351E-38 |
| | 0 | 0 |
| | 1.175494351E-38 | 3.402823466E+38 |
| DOUBLE | -1.7976931348623157E+308 | -2.2250738585072014E-308 |
| | 0 | 0 |
| | 2.2250738585072014E-308 | 1.7976931348623157E+308 |
| DECIMAL(S,D) | S<=65 (précision) | D<=30 (décimale) |
| FIXED | synonyme DECIMAL | |
| NUMERIC | synonyme DECIMAL | |
| DEC | synonyme DECIMAL | |

# Modèle physique des données

**Champs alpha-numériques et binaires**

| Type | Longueur max |
|---|---|
| CHAR(S) | 255 (selon version) |
| VARCHAR(S) | 255 (selon version) |
| BINARY(S) | 255 (selon version) |
| VARBINARY(S) | 255 (selon version) |

| Type | Longueur max |
|---|---|
| TINYBLOB | 256 |
| BLOB | 65 536 (64 Ko) |
| MEDIUMBLOB | 16 777 216 (16 Mo) |
| LONGBLOB | 4 294 967 296 (4 Go) |
| TINYTEXT | 256 |
| TEXT | 65 536 (64 Ko) |
| MEDIUMTEXT | 16 777 216 (16 Mo) |
| LONGTEXT | 4 294 967 296 (4 Go) |

**Champs alpha-numériques et binaires**

| Type | Val min | Val max |
|---|---|---|
| DATETIME | '1000-01-01 00:00:00' | '9999-12-31 23:59:59' |
| DATE | '1000-01-01' | '9999-12-31' |
| TIMESTAMP | '1970-01-01 00:00:01' | '2038-01-19 03:14:07' |
| TIME | '-838:59:59' | '838:59:59' |
| YEAR | 1901 | 2155 |

# Transformation vers le modèle physique des données

**Exemples :**

**Modèle entité-association**

**Modèle physique des données**

# 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.

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

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

# LDD : 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;
```

**Suppression d'une base de données**

```
DROP DATABASE [IF EXISTS] db_name
```

**Modification d'une base de données**

```
ALTER DATABASE db_name alter_specification [, alter_specification] …
```

# LDD : 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] tbl_name
[(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`).

• 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 est le type sont obligatoires. Par défaut un champ est défini en `NULL`. Les champs sont séparés par des virgules.

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`.

2. Les options facultatives de la table 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.

**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 ;
```

# LDD : 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_anne
(Annee_naiss)) ENGINE=InnoDB
```

**Modification d'une table**

```
CREATE TABLE tbl_name
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
```

Renommage d'une table :

```
RENAME TABLE nom_de_table TO
nouveau_nom_de_table
```

Suppression d'une table :

```
DROP TABLE tbl_name
```

# LMD : Langage de manipulation des données

Les commandes principales sont :

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

**Insérer des données dans une table :**

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

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

Le fait de préciser les champs est optionnel mais impose en cas de non indication de donner les expressions de chaque colonne dans l'ordre.

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.

**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.

# LMD : Langage de manipulation des données

**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`.

**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.

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).

# LMD : Langage de manipulation des données

**Exemples**

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

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

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

| Personne | |
|---|---|
| ID | int |
| Nom | varchar(30) |
| Prenom | varchar(30) |
| Adress# | int |

# LMD : Langage de manipulation des données

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

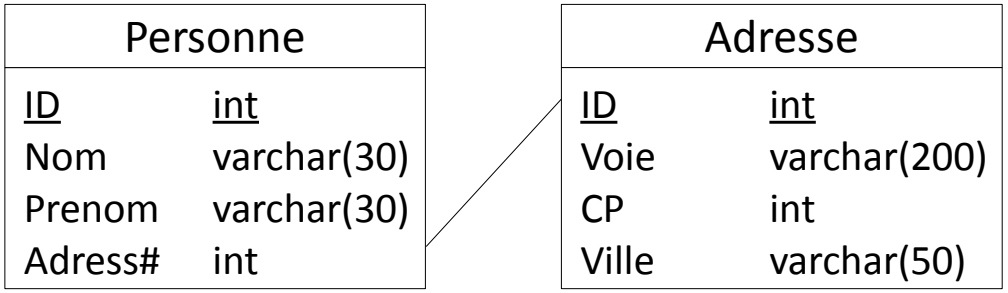
Tous les opérateurs `=, <=>, <, >, !=, >=, <=, <>, BETWEEN, IN, NOT IN, IS NULL, IS NOT NULL,` ... sont supportés.

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

**Attention :** si toutes les tables listées dans la clause `FROM` ne sont pas associées dans la clause `WHERE`, le moteur effectuera un produit cartésien des tables non liées.

Ainsi si 3 tables de 500, 1000, et 2500 lignes sont appelées dans le `FROM` sans association dans la clause `WHERE`, le résultat sera de :

500 * 1000 * 2500 = 1 250 000 000 lignes.




**Exemple**

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

# SQL avancé : les jointures

**Utilisation des jointures**

→ Sélectionner les données se trouvant dans plusieurs tables.

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

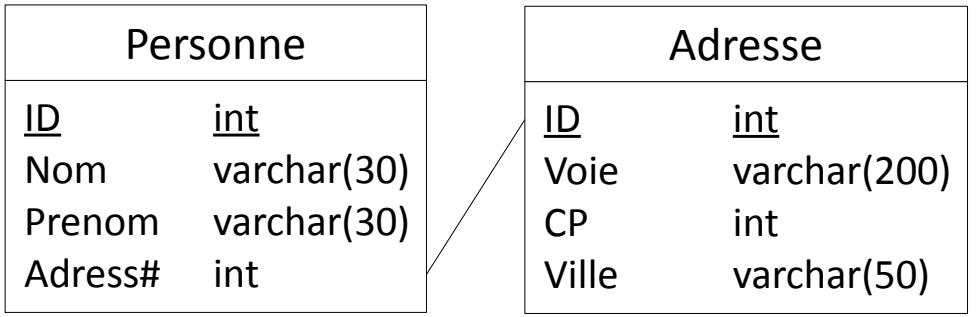
- Select (lecture)
- Update (mise à jour)
- Delete (suppression)

**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**.

**Exemple**




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

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

**Remarque :** ordre sans importance

# SQL avancé : les 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.

**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 :

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.

# SQL avancé : les 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 :

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*.

# SQL avancé : les 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.

# SQL avancé : l'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**

| Id | Nom | Prenom | Id_adresse |
|---|---|---|---|
| 1 | Durand | Marie | 4 |
| 2 | Simpson | Bart | 3 |
| 3 | Dubois | Jean | NULL |
| 4 | Simpson | Lisa | 3 |

**Adresse**

| Id | 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

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.

# SQL avancé : l'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.

# Modélisation MERISE

**Méthode MERISE :** méthode d'analyse, de conception et de réalisation de systèmes d'informations.

Méthode MERISE **pas seulement pour les bases de données** :

- Exprimer le besoin
- Créer les modèles conceptuels
- Créer les modèles logiques
- Créer les modèles physiques

cahier des charges → langage SQL

---

**UML** (Unified Modeling Language) :

- autre langage de modélisation
- langage dédié à l'objet

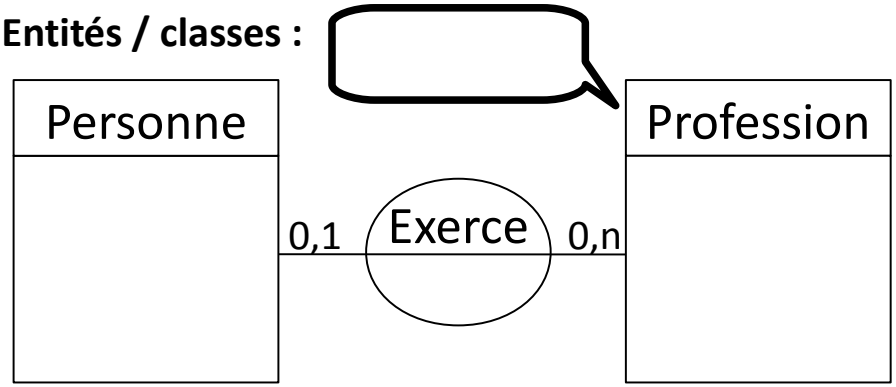
plusieurs types de diagramme, dont un utile en bases de données :
le **diagramme de classes**

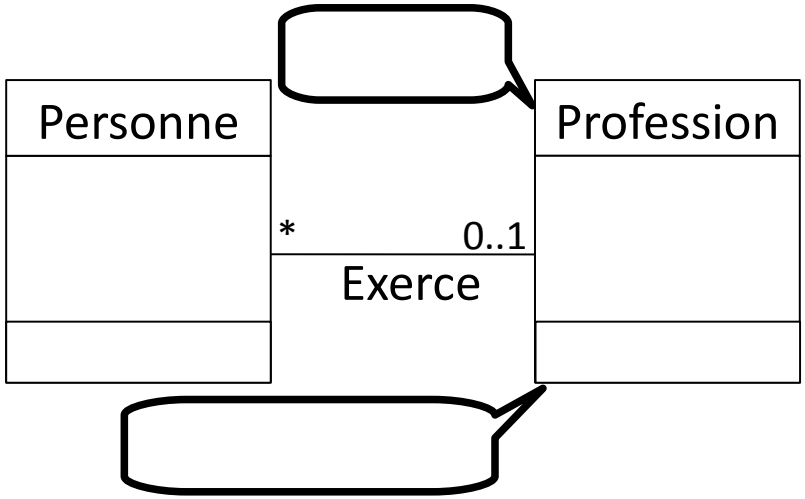
---

Traduction entre *modèle conceptuel des données de MERISE* et *diagramme de classes UML !*

# Modélisation MERISE & UML

**Entités / classes :**



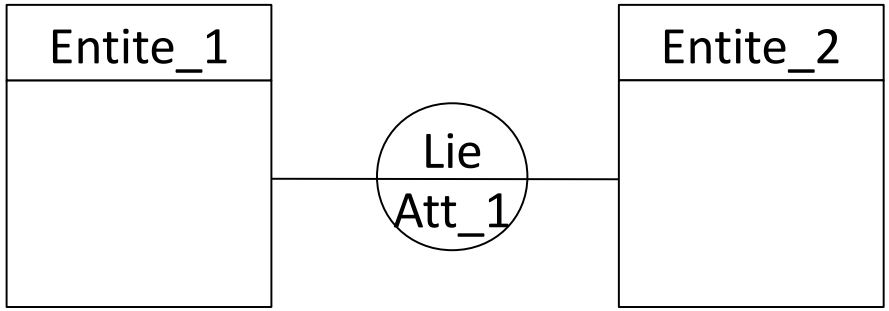



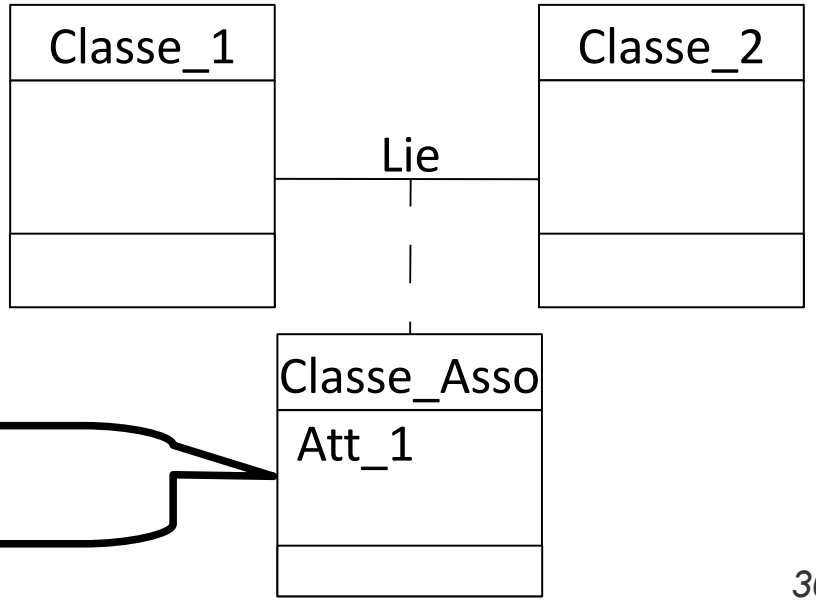

**Cardinalités :**

0,1 1,1 0,n 1,n

0..1 1 * 1..*

**Associations avec attributs :**



# Les « plus » d'UML : agrégation

**Agrégation :**

- Associations **non symétriques**
- Une classe joue un **rôle prépondérant** par rapport à l'autre

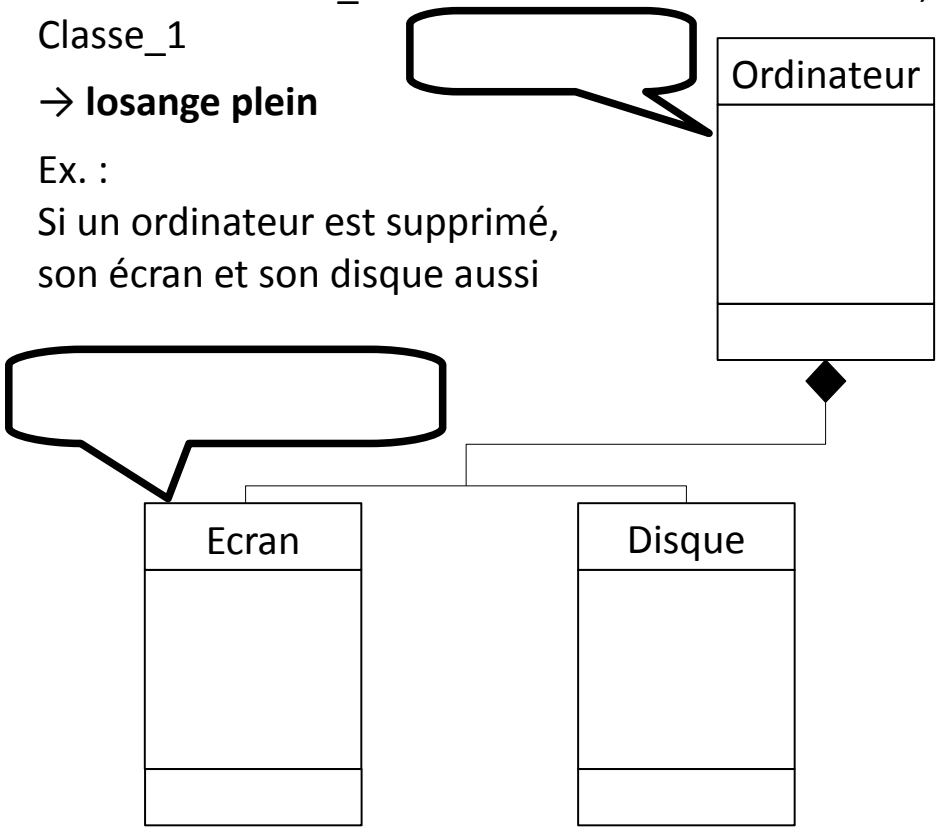
Deux formes d'agrégation :

**Composition**

Une classe Classe_2 est sous-ensemble d'une autre, Classe_1

→ **losange plein**

Ex. :
Si un ordinateur est supprimé, son écran et son disque aussi

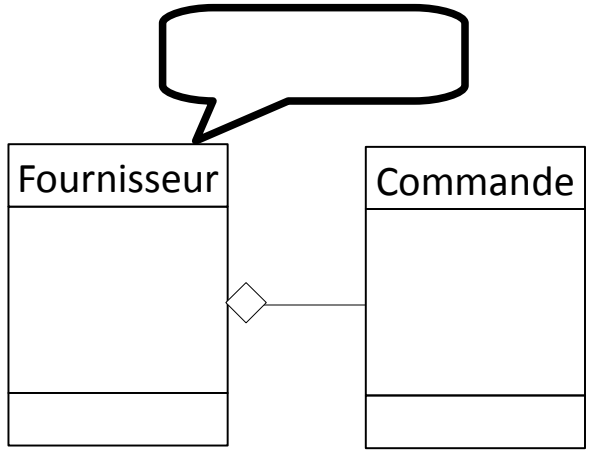



**Agrégation partagée**

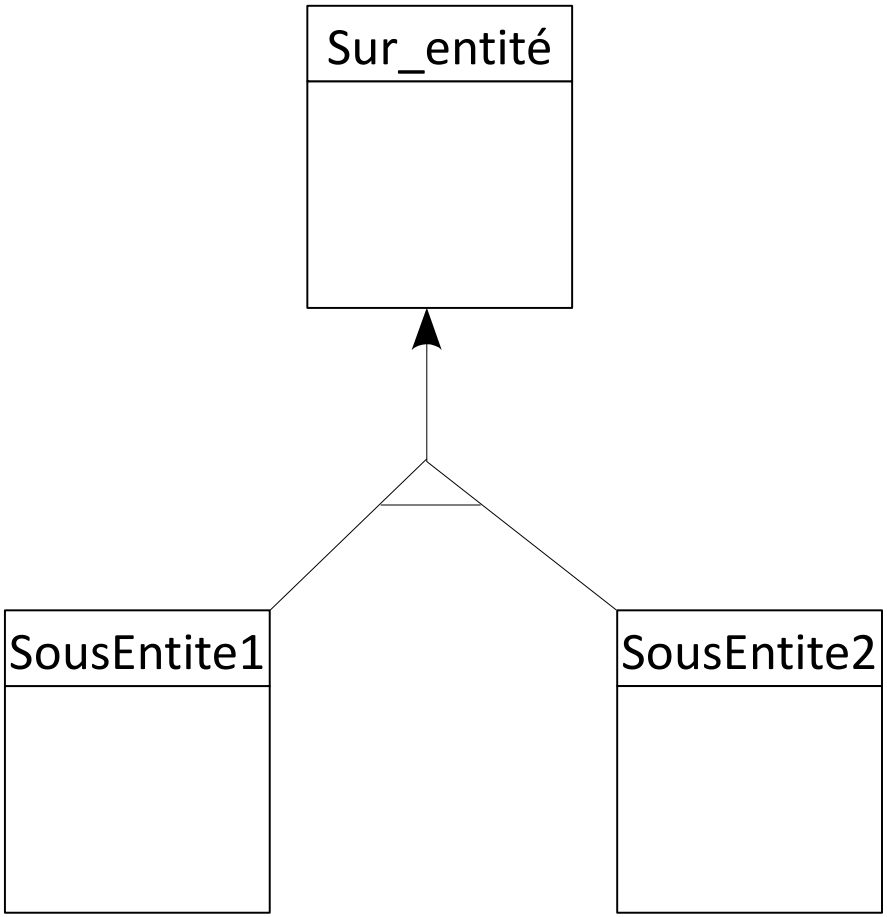
Une classe Classe_2 est dépendante d'une autre, Classe_1

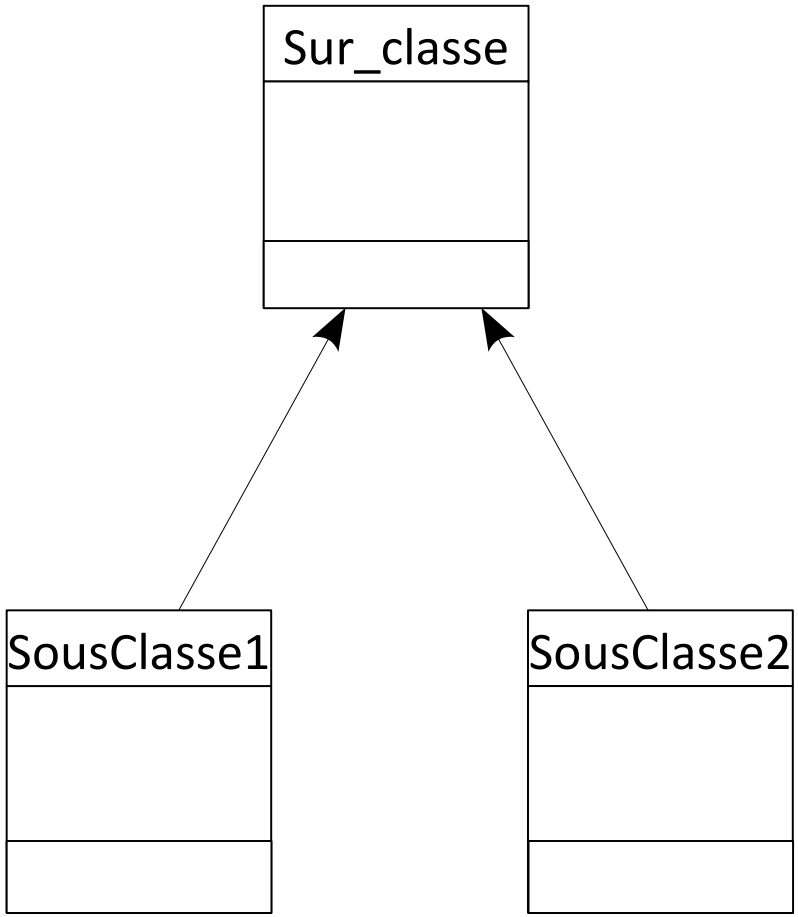
→ **losange vide**

Ex :
Si on supprime le fournisseur, on ne supprime pas forcément les commandes associées



**Héritage :**


Sur_entité
SousEntite1
SousEntite2


Sur_classe
SousClasse1
SousClasse2