# Exploitation, fouille de données, gestion de bases de données

Lisa Di Jorio, Anne Laurent

## Table des matières

# 1 Introduction

Ce document est une introduction à la conception et la manipulation de base de données d'une part, et la fouille de données d'autre part. Ce document est produit dans le contexte du module GMIN107 du Master Energie. Le module est posé sur 21 heures, découpées de la manière suivante : 6h de cours, 4h de td, 10h de tp. Le module sera sanctionné par un projet a rendre. Le responsable du module est le professeur Anne Laurent (Anne.Laurent@lirmm.fr) et il est enseigné par Lisa Di Jorio (lisa.dijorio@gmail.com).

Après avoir présenté quelques généralités sur les bases de données, ce document se décline en trois grands chapitres :

- Modélisation des bases de données
- Manipulation des bases de données
- Fouille de données

Avant de parler de base de données, il est nécessaire de bien définir ce qu'est une *donnée*. D'un point de vue informatique, une donnée est un élément contenant une information. Ces informations, de différentes natures, peuvent prendre différents formats dont voici quelques exemples :

- Les données numériques enregistrent une valuation numérique (entier, nombre à virgule). Par exemple, nous pouvons citer les données issues de capteurs météorologiques
- Les données binaires enregistrent des éléments ne pouvant prendre que deux valeurs. L'exemple le plus naturel est une donnée ne pouvant que prendre les valeurs oui (1) ou non (0)
- Les données textuelles enregistrent des valuations qualitatives. Par exemple, on peut considérer les informations descriptives ou explicatives, ou encore les pages web.
- Les données graphiques sont des images : png, jpeg, etc... Elles regroupent aussi bien les images "réelles" telles que les photographies que les images "virtuelles" telles que les images 3D ou encore les graphes générés.

Une *base de données* désigne un **ensemble d'informations stockées** sur un système informatique. Une base de données répond à de nombreuses problématiques dont le stockage efficace (comment ne pas saturer le disque dur ?), le tri et la sélection des données (comment accéder rapidement aux informations stockées ?). On discerne donc deux organisations distinctes : **l'organisation logique**, qui désigne le modèle sémantique selon lequel les données sont stockées (par nature, de manière hiérarchique, en réseau...) et **l'organisation physique**, qui désigne la manière dont les données sont organisées sur le disque dur (fichiers séquentiels, tables de hachage...).

Dans ce cours, nous nous intéressons au modèle de données le plus utilisé : le *modèle de données relationnel*. Il est démontré que le modèle relationnel permet de résoudre tout type de requête, ce qui en fait un outil puissant et particulièrement apprécié des entreprises. Concrètement, les modèles relationnels sont utilisés via des logiciels appelés **systèmes de gestion de bases de données** (ou SGBD). Il en existe beaucoup : Microsoft SQL Server, PostgreSQL, Oracle, MySQL... Ici, nous utiliserons MySQL, un SGBD gratuit, relativement complet et fonctionnant sur les systèmes Linux et Windows.

La *fouille de données* (data mining en anglais) est une partie intégrante du processus d'Extraction de Connaissance dans les grandes Bases de Données, ou ECD (Knowledge Discovery in large Databases, ou KDD), décrit a la figure 1.

Le processus d'extraction de connaissances dans les bases de données est un processus complet et

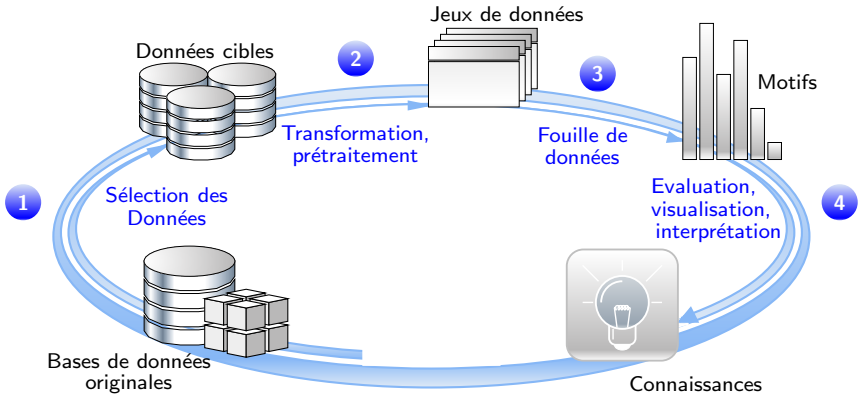


Figure 1 – Processus d'extraction de connaissances

complexe de découverte de connaissances au préalable inconnues à partir de grandes bases. Apparu au début des années 90, ce processus suscite un fort intérêt industriel, notamment pour son champ d'application très large, pour son coût de mise en œuvre relativement faible, et surtout pour l'aide qu'il peut apporter à la prise à la décision. Ce processus peut être découpé en quatre grandes étapes illustrées par la figure 1.

La fouille de données est l'étape centrale du processus d'extraction de connaissances. Elle consiste à découvrir de nouveaux modèles au sein de grandes quantités de données. Cependant, il est rarement possible d'appliquer directement la fouille de données sur les données brutes. Les premières opérations du processus ECD correspondent donc à la transformation des données avant de pouvoir appliquer des algorithmes de fouille de données.

## 2 Modélisation des bases de données

En informatique, nous discernons lors de la création d'un produit trois grandes phases : la conception, le développement et la livraison. La phase de conception consiste a étudier les demandes, les données ainsi que les traitements a effectuer. C'est en général dans cette phase que s'appliquent les techniques de modélisation. Il en découle la description des bases de données éventuelles à créer, les programmes à écrire et la manière dont tout cela va être intégré. Les tâches effectuées durant cette phase assurent la cohérence du produit fini. Par exemple, c'est durant cette étape que l'on va identifier les modules du produit susceptibles d'évoluer. Ainsi, la modélisation proposée facilitera l'évolution future du produit.

Cette étape permet de clairement définir ce qui existe, ce qui doit être créé, ce qui doit être modifié, qui fait quoi ainsi que les coûts et le temps engendrés.

Le développement concerne la partie de réalisation à proprement dit du produit. En base de données, il s'agit de la création de la base ainsi que de son remplissage par les données. L'une des méthodes les plus utilisées en France est la méthode MERISE (Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise). MERISE décrit, après abstraction, le modèle (le système) du produit :

- Le **Modèle Conceptuel des Données** (ou MCD) est un schéma représentant la structure du système d'information. Il se place du point de vue des données, et cible les dépendances ou

relations entre les différentes données du système d'information

- Le **Modèle Conceptuel des Traitements** (ou MCT) est un schéma représentant les traitements, en réponse aux événements à traiter
- Le **Modèle Logique des Données** (ou MLD) consiste a réécrire le MCD en vue d'une implémentation logicielle ultérieure. On désigne souvent cette étape sous le terme de dérivation
- Le **Modèle Logique des Traitements** (ou MLT, ou Modèle Organisationnel des Traitements, MOT) permet de décrire les contraintes dues a l'environnement de travail
- Le **Modèle Physique des Données** (ou MPD) est la traduction du stockage physique des données (taille, type). Elle correspond a l'implémentation du MLD dans un SGBD.
- Le **Modèle Opérationnel des Traitements** (ou MOpT) permet de spécifier les fonctions que devra implémenter le programmeur.

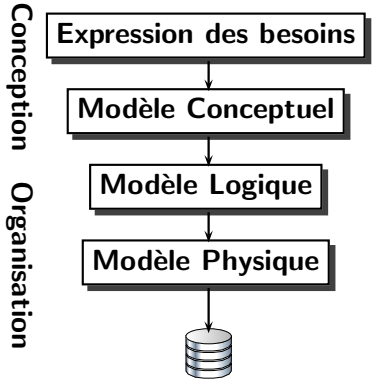


Figure 2 – Processus MERISE

La figure 2 illustre la chronologie des étapes de modélisation de la méthode MERISE. Afin de mieux comprendre les différences entre les différentes étapes, nous prendrons l'exemple d'un schéma très simple d'une partie du fonctionnement universitaire.

Lors d'un entretien avec les différents acteurs du service administratif de l'université, vous apprenez que les étudiants s'inscrivent à l'université pour obtenir un diplôme. Le système doit retenir leur date d'inscription. Le diplôme est obtenu en fonction des notes obtenues sur des modules (ces notes vont de 0 à 20). Les modules sont attribués à un ou plusieurs diplômes et ont des coefficients différents. Les professeurs enseignent les modules, ils donnent un certain nombre d'heures de cours par module. Les étudiants suivent ces cours, passent un partiel et obtiennent une note.

Les différentes informations de cet énoncé engendrent les réflexions et actions suivantes durant le déroulement de la méthode MERISE :

- Durant la conception du MCD, les différents objets et les relations qui les lient sont identifiés. On cherche également à identifier les propriétés des différents objets ainsi qu'à les quantifier. Par exemple, on détecte les étudiants qui possèdent comme propriétés un nom, un prénom, une adresse, etc..., les professeurs, les modules. On remarque que les étudiants suivent un ou plusieurs modules, alors que les professeurs enseignent aucun ou plusieurs modules
- Durant la conception du MCT, les différentes actions des objets ainsi que leurs impacts sur les autres objets et les relations sont identifiés. Par exemple, un étudiant s'inscrit à un diplôme, un professeur fait passer un partiel, un professeur corrige un partiel et affecte une note à un étudiant, etc...
- Durant le MLD, on décide des tables qui composeront notre base de données, ainsi que des

attributs qui constitueront ces tables. Par exemple, on décide de créer une table Etudiant, avec les attributs identifiant, nom, prénom et adresse. Cette étape nous permettra de produire un dictionnaire de données

- Durant le MLT, on détermine les différentes composantes de l'espace de travail ainsi que leur rôle. Par exemple, on identifiera le profil administrateur dont le rôle sera de saisir les inscriptions des étudiants, le profil des professeurs dont le rôle sera l'enseignement et la notation. On déterminera les spécifications des interfaces informatiques ainsi que les dialogues homme-machine.
- Durant le MPD, on décide d'utiliser le SGBD Oracle et non MySQL, en réponse aux différentes contraintes exprimées lors des étapes précédentes. On crée effectivement les tables de la base de données, et on remplit cette base. Par exemple, on va enregistrer l'ensemble des étudiants, des professeurs ainsi que les modules que suivent les étudiants et qu'enseignent les professeurs.
- Durant le MOpT, les programmeurs vont implémenter les fonctions qui doivent être automatiquement effectuées a l'occasion de différentes saisies. Par exemple, ils vont implémenter une fonction qui permet de dire si oui ou non un étudiant a obtenu un diplôme.

## 2.1 Le Modèle Conceptuel des Données

Le MCD repose sur les notions d'entité et d'association et sur la notion de relation. Il débouche concrètement sur un schéma appelé **schéma entité-association**. Le MCD doit clairement identifier les objets, les propriétés de ces objets, les relations entre les objets ainsi que les cardinalités relatives à ces objets.

### 2.1.1 La propriété

La propriété est un élément porteur d'une information simple. Par exemple, le nom, le prénom, l'identifiant d'un étudiant sont des propriétés. De même, le nombre d'heures, le coefficient ou encore le nom sont des propriétés d'un module. Lorsque l'on se situe au niveau du SGBD, c'est-à-dire que l'on a stocké et organisé ces propriétés, on parle alors d'attribut. Par exemple, le nom est un attribut de la table Etudiant.

Les valeurs prises par les propriétés sont appelées occurrences. Par exemple, le tableau 1 montre quelques propriétés ainsi que leur possibles occurrences.

| **Propriété** | **Occurrences** |
|---|---|
| Nom | Durant, Normand, Dupond, Dupont |
| Prenom | Nicolas, Paul, Pierre, Jacques |
| Adresse | 123 rue du monde 23000 ville |

Table 1 – Exemple de propriétés et de leurs occurrences

Les propriétés les plus simple sont dites **atomiques** ; par exemple, le nom et le prénom sont des propriétés atomiques. En revanche l'adresse qui peut être redécomposée en plusieurs sous-propriétés telles que la rue, le code postal et la ville n'est pas une propriété atomique. Le choix d'une propriété atomique ou non dépendra des besoins du système.

Par exemple dans ce cas, est-il courant que plusieurs personnes aient la même adresse ? Veut-on faire des statistiques par quartier ? Après audit auprès des administrateurs, il ressort que la très grande majorité des étudiants vivent seuls dans des studios, et que l'université n'utilisera l'adresse que dans le

but d'envoyer des courriers officiels et qu'aucune application utilisant la géolocalisation n'est prévue à court ou long terme. Compte tenu de toutes ces informations, nous décidons de ne pas décomposer la propriété adresse.

### 2.1.2 L'entité ou objet

L'entité est définie comme un objet de gestion considéré d'intérêt pour représenter l'activité à modéliser. L'entité a un rôle dans le système que l'on veut décrire, c'est ce qui la différencie de la propriété. L'entité regroupe les éléments d'une même classe. Par exemple, Nicolas Durant, Paul Normand ou Pierre Dupond sont considérés comme des éléments appartenant à la même classe Etudiant. Si l'on reprend l'exemple précédent, les entités sont Etudiant, Diplome, Module, et Professeur.

Chaque entité est définie par plusieurs propriétés. Parmi ces propriétés, l'une doit permette d'identifier de manière précise chaque ensemble d'occurrences (on doit bien comprendre que l'on parle bien de Paul Normand et pas de Nicolas Dupond) et assurer qu'il n'existe pas de doublon. Une telle propriété s'appelle l'**identifiant**. Lors de la représentation du MCD, les identifiants sont soulignés. Par construction, le MCD impose que toutes les propriétés d'une entité ont vocation à être renseignées (il n'y a pas de propriété "facultative").

Le MCD doit, de préférence, ne contenir que le cœur des informations strictement nécessaires pour réaliser les traitements conceptuels : les informations calculées (comme la moyenne d'un étudiant) ne doivent pas y figurer.

Les ensembles de propriétés valuées sont appelés des n-uplets. Par exemple si l'on considère les entités et leur propriétés de la figure 3, (1, Master Energie, Physique) est un n-uplet de l'entité Diplome.

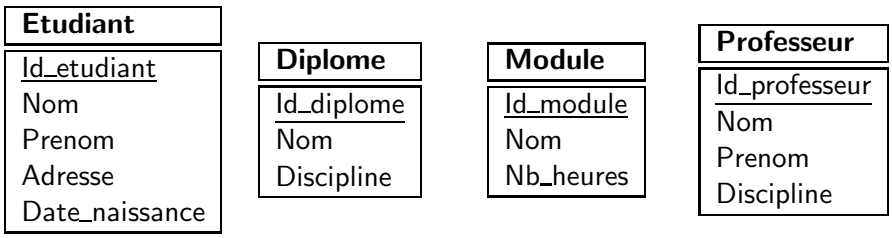


Figure 3 – Entités du système universitaire

### 2.1.3 L'association ou relation

L'association est un lien sémantique entre une ou plusieurs entités : l'association peut être réflexive, de préférence binaire, parfois ternaire, voire de dimension supérieure. Toutefois, la mise en œuvre dans les pas suivants d'une association de dimension supérieure a trois s'avère complexe, et est fortement déconseillée. Elle peut également être porteuse d'une ou plusieurs propriétés, comme ne pas en posséder du tout. Si l'on reprend notre exemple, nous trouverons les relations s'inscrire entre Etudiant et Diplome, contenir entre Diplome et Module et enseigner entre Module et Professeur. Comme le montre la figure 4, la relation s'inscrire comporte une propriété, *date_inscr*, qui permet de conserver une trace de la date d'inscription au module.

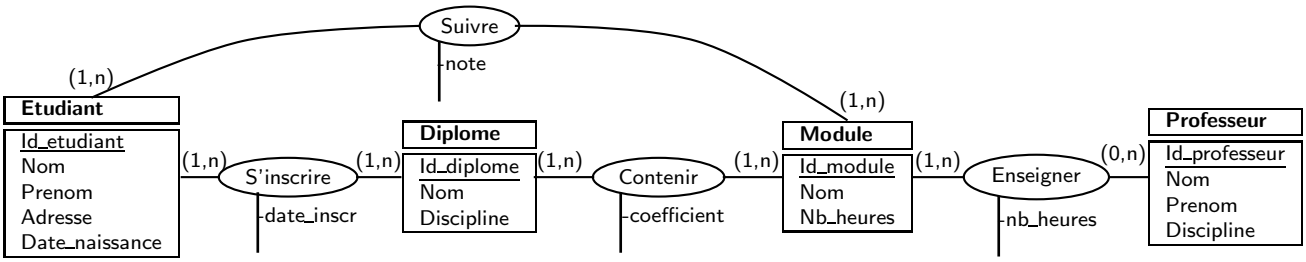


Figure 4 – Schéma Entités-Association du système universitaire

Cette description sémantique est enrichie par la notion de cardinalité, qui indique le nombre minimum (0 ou 1) et maximum (1 ou n) de fois où une occurrence quelconque d'une entité peut participer à une association. Par exemple, l'on sait qu'un diplôme contient au minimum un module, et un certain nombre de modules maximum. Cela se traduit par la cardinalite (1,n) disposée entre l'entité Module et l'association Contenir. Notez que dans ce modèle, un module doit être enseigné par au moins un professeur, mais en revanche un professeur n'est pas forcément un enseignant : il peut consacrer son temps à la recherche. Les cardinalités expriment les sémantiques suivantes :

- (0,n) aucun ou plus au minimum, et autant que l'on veut au maximum
- (1,n) au moins un au minimum, et autant que l'on veut au maximum
- (0,1) aucun ou un au minimum, un seul au maximum
- (1,1) un et seulement un

## 2.2 Les dépendances fonctionnelles

Les **dépendances fonctionnelles** sont un outil supplémentaire permettant de s'assurer que l'on a construit un MCD cohérent et efficace. Il est également possible d'établir un schéma entité-association en étudiant uniquement les dépendances fonctionnelles entre attributs. La maîtrise des dépendances fonctionnelles est également primordiale pour assurer une bonne normalisation du schéma relationnel.

Un attribut Y dépend fonctionnellement d'un attribut X si et seulement si une valeur de X induit une unique valeur de Y. On note une dépendance fonctionnelle par une flèche simple : $X \to Y$. Par exemple, si $X$ est le numéro d'un étudiant et $Y$ son nom, nous avons bien $X \to Y$, car l'on peut obtenir le nom d'un étudiant à partir de son numéro. Par contre, nous n'avons pas $Y \to X$ car deux étudiants peuvent avoir le même nom.

La transitivité est définie de la manière suivante : si $X \to Y$ et $Y \to Z$, alors $X \to Z$. Par exemple, nous avons *id_etudiant* $\to$ *id_diplome* $\to$ *id_module* $\to$ *note*. Par transitivité, *id_etudiant* $\to$ *note*. Cette dépendance fonctionnelle est dite **transitive**, car elle n'est pas directe. En revanche, *id_etudiant* $\to$ *id_diplome* est une **dépendance fonctionnelle directe**.

Il est possible d'organiser les dépendances fonctionnelles directes sous la forme d'un réseau appelle **graphe de couverture minimale**.

## 2.3 Les formes normales

Les formes normales sont habituellement utilisées en base de données relationnelles, mais il est plus facile de vérifier qu'elles sont bien respectées dès l'étape de création du MCD, afin de ne pas revenir en arrière lors des étapes ultérieures.

En base de données, les formes normales permettent d'assurer la cohérence des données, ainsi que d'éviter des anomalies d'écriture, de lecture ou encore des contres performances. Il existe huit formes normales (ou NF) : de 1NF à 6NF et les formes BCNF et DKNF. Dans ce cours, nous ne verrons que les formes normales 1NF, 2NF, 3NF et BCNF car elles sont suffisantes pour construire un modèle relationnel juste.

Une relation en 1NF garantie un accès plus rapide aux données et évite les besoins de mises à jour. Une relation est en 1NF si toutes les propriétés sont atomiques (elles ne peuvent contenir qu'une valeur et pas une liste de valeur) et sont constantes dans le temps (privilégier par exemple la propriété *date de naissance* plutôt que la propriété *age*, cela évite de remettre l'âge à jour tous les ans). Par exemple, la figure 5 n'est pas en 1NF, car la propriété étudiant contiendra une liste d'étudiants. Il convient donc de créer, comme sur la figure 4 deux entités reliées par une association.

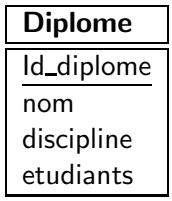


Figure 5 – Entité Diplome qui ne respecte pas la 1NF

La seconde forme normale évite la redondance des données. Une relation est en 2NF si elle est en 1NF et si toute propriété ne composant pas un identifiant dépend d'un identifiant. En d'autres termes, si l'identifiant est composé de plusieurs propriétés, toutes les autres propriétés doivent dépendre de tous les attributs qui composent l'identifiant, et non pas d'une partie seulement. Par exemple, l'entité représentée a la figure 6 ne respecte pas la 2NF : nous avons $\{id\_commande, id\_client \rightarrow nom\_article\}$, et $\{id\_client \rightarrow nom\_client\}$. Donc le nom du client ne dépend que de son identifiant, et non pas de la composition $\{id\_commande, id\_client\}$. Dans ce cas, il convient de créer une entité Client contenant la propriété *nom_client*, et une association entre les entités Commande et Client. Pour respecter la 2NF, il suffit de créer une entité Client ayant pour identifiant *id_client* et comme propriété *nom_client*.

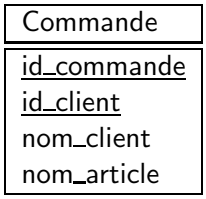


Figure 6 – Entité Commande qui ne respecte pas la 2NF

La 3NF évite également la redondance des données. Pour être en 3NF, une relation est en 2NF et toute propriété ne composant pas un identifiant dépend directement d'un identifiant. En d'autres termes, il ne doit pas y avoir de dépendance directe entre propriétés non-clefs. Par exemple, l'entité Capacité de la figure 7 dépend directement du modèle, et non de la propriété *id_avion*. Pour être en 3NF, il faut créer une entité Modèle, et déplacer la propriété *capacité* dans cette nouvelle entité.

La Forme Normale de Boyce-Codd, ou BCNF, assure la cohérence des données, ainsi que la non-redondance. Pour être en BCNF, il faut qu'une relation soit en 3NF et que toutes les propriétés non-clefs ne soient pas source de dépendance fonctionnelle (DF) vers une partie de la clef. Par exemple, supposons

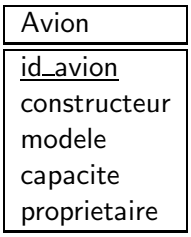


Figure 7 – Entités Diplome qui ne respecte pas la 3NF

que l'on ait créé l'entité Universite telle que représentée à la figure 8, avec un identifiant composé des propriétés (*id_etudiant, id_matiere*) et les dépendances fonctionnelles suivantes :

- $\{$*id_etudiant, id_matiere* $\rightarrow$ *id_enseignant, note*$\}$
- $\{$*id_enseignant* $\rightarrow$ *id_matiere*$\}$

Cette relation n'est pas en BCNF, car *id_matiere*, qui fait partie de l'identifiant, dépend directement d'une autre propriété. Pour passer en BCNF, il faut créer une nouvelle entité Enseignant, et la relier à Universite au travers d'une association Enseigner.

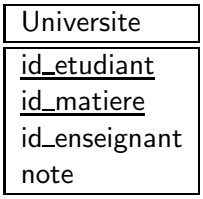


Figure 8 – Entités Diplome qui ne respecte pas la 3NF

## 2.4 Le Modèle Conceptuel des Traitements

Le MCT modélise la dynamique du système d'information en schématisant l'ensemble des actions possibles et les traitements qui en découlent. Le MCT ne traite pas des logiciels qui doivent être utilisés pour réaliser les actions, ni de quand et où elles doivent être réalisées. Le MCT repose sur les notions d'événements, d'opération et de processus.

Un **événement** est assimilable à un message porteur d'informations donc potentiellement de données mémorisables, comme par exemple *l'étudiant demande à être inscrit à un diplôme*. Un événement peut déclencher une opération ou encore être le résultat d'une opération. Nous le représentons par une ellipse.

Une **opération** se déclenche uniquement par le stimulus d'un ou de plusieurs événements synchronisés. Elle est constituée d'un ensemble d'actions correspondant à des règles de gestion de niveau conceptuel, stables pour la durée de vie de la future application. Le déroulement d'une opération est ininterruptible : les actions à réaliser en cas d'exceptions, les événements résultats correspondants doivent être formellement décrits. Si l'on reprend l'exemple précédent, une opération peut être *l'administrateur vérifie que l'étudiant a obtenu le diplôme prérequis*. Nous représentons une opération par un rectangle

Un **processus** est un enchaînement pertinent d'opérations du point de vue de l'analyse, par exemple l'ensemble des opérations et des événements qui conduisent à l'inscription d'un étudiant.

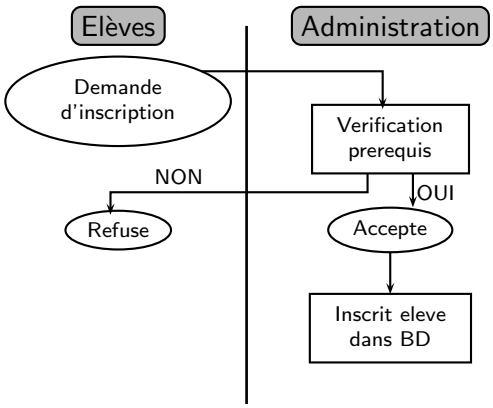


Figure 9 – Processus d'inscription à l'université

## 2.5 Le Modèle Logique des Données

Le MLD est également appelé **étape de dérivation**. Il permet de décrire les différentes composantes du modèle relationnel dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelle. L'étape du MLD se traduit par :

- La création du dictionnaire des données
- La formalisation des tables relationnelles, ainsi que des contraintes d'intégrité et de domaine associées.

### 2.5.1 La table, la ligne, l'attribut et les clefs

En base de données, une Entité est stockée sur le disque dur sous la forme d'un tableau à deux dimensions appelé **table**. Chaque colonne de la table contient les propriétés de l'entité, et sont appelées des **attributs**. Un ensemble d'occurrences (une ligne de la table) est appelé une **transaction**.

| ETUDIANT | | | | |
|---|---|---|---|---|
| id_etudiant | nom | prenom | adresse | date_naissance |
| 1 | Durant | Nicolas | 123 rue ... | 10/10/1993 |
| 2 | Dupond | Hélène | 345 boulevard ... | 25/03/1995 |
| 3 | Dubois | Thomas | 678 place ... | 01/01/1989 |
| ... | ... | ... | ... | ... |

Table 2 – Exemple de table de données : la table ETUDIANT

La table 2 est un exemple de la formalisation de l'entité Etudiant. Elle contient cinq attributs : {id_étudiant, nom, prenom, adresse et date_naissance}. La ligne (1, Durant, Nicolas, 123 rue..., 10/10/1993) est une transaction, et il y a trois transactions représentées dans cet exemple.

Les identifiants ont pour vocation d'assurer l'unicité d'une transaction. Par exemple, il est possible d'avoir deux étudiants distincts Nicolas Dupond dans une même université. Un numéro unique attribué à chacun permet de les différencier. Une fois la relation passée en table, nous appelons les identifiants des **clefs primaires**. Évidement, une clef primaire peut être composée de plusieurs attributs.

Souvent, un attribut $A_1$ d'une table $T_1$ ne peut prendre que les valeurs d'un autre attribut $A_2$ d'une table $T_2$. Par exemple, les identifiants des professeurs de la table ENSEIGNER doivent être présents dans la table PROFESSEUR : un professeur n'appartenant pas à l'université ne peut pas enseigner dans cette université. De tels attributs sont appelés **clefs étrangères**.

Par convention, une table est notée de la manière suivante : NOM TABLE (Attribut$_1$, Attribut$_2$, ..., #Attribut$_n$). La clef primaire est soulignée et les clefs étrangères sont précédées d'un dièse. Par exemple, nous pouvons écrire :

- ETUDIANT (id_etudiant, nom, prenom, adresse, date_naissance)
- ENSEIGNER (id_module, #id_professeur, nombre_heures)

Notez les points suivants :

- Une table doit contenir une clef primaire
- La valeur prise par une clef primaire ne peut jamais être nulle (pas d'étudiants sans numéro unique)
- Une clef étrangère peut également être clef primaire (c'est le cas de l'*id_professeur* de la table ENSEIGNER)
- Une clef étrangère peut être composée de plusieurs attributs

### 2.5.2 Le dictionnaire des données

Le dictionnaire des données permet de définir et de décrire le vocabulaire commun aux tables. Il s'agit ici de définir clairement l'ensemble des attributs de chaque table ainsi que les caractéristiques associées. Le dictionnaire de données doit contenir :

- Le nom des attributs
- Le type des attributs : texte (préférable pour les clefs et les attributs qui ne font pas l'objet de calculs), numérique (préférable pour les attributs qui font l'objet de calculs), date, monnaie, etc...
- Description sommaire des attributs
- Exemple de valeur que peut prendre chaque attribut

Le dictionnaire de données ne doit pas contenir de nom composés de plusieurs mots, ni de synonymes, ni d'homonymes. Par exemple, le tableau 3 montre le dictionnaire de données couvrant les entités Etudiant et Diplome ainsi que la relation s'inscrire. On notera que la propriété *nom* de l'entité Etudiant a été renommée *nom_etudiant* afin d'éviter les homonymies avec les professeurs et les diplômes.

| **Nom** | **Type** | **Description** | **Exemple** |
|---|---|---|---|
| id_etudiant | INT | identifiant unique de l'étudiant | 23 |
| Nom_etudiant | VARCHAR | nom de famille de l'étudiant | Dupont |
| Prenom_etudiant | VARCHAR | prénom de l'étudiant | Hélène |
| Date_inscription | DATE | date d'inscription au diplôme | 01/09/2011 |
| id_diplome | INT | identifiant unique du diplôme | 34 |
| nom_diplome | VARCHAR | nom du diplôme | Master Energie |
| discipline_diplome | VARCHAR | discipline principale du diplôme | Physique |
| ... | ... | ... | ... |

Table 3 – Dictionnaire de données couvrant les entités *Etudiant*, *Diplome* et l'association *s'inscrire*

### 2.5.3 La dérivation

La transcription d'un MCD en modèle relationnel ou dérivation consiste à identifier formellement les tables qui composeront la base de données. Il existe des règles simples permettant de dériver les tables.

**Règle 1 :** toute entité est transformée en table, et son identifiant en clef primaire.
En reprenant le MCD de la figure 4, nous obtenons :

- ETUDIANT (id_etudiant, nom_etudiant, prenom_etudiant, adresse, date_naissance)
- DIPLOME (id_diplome, nom_diplome, discipline_diplome)
- MODULE (id_module, nom_module, nb_heure_module)
- PROFESSEUR (id_professeur, nom_professeur, prenom_professeur, discipline_professeur)

Les autre règles permettent de déterminer si une association va être transformée en table ou non.

**Règle 2 :** Les relations de type un à un (1,1) deviennent des clés étrangères. On place une clef étrangère dans la table cote (0/1,1) qui référence la clef primaire de l'autre table.
Par exemple, dans le cas de la figure 10, il n'est pas nécessaire de créer une table RECEVOIR, car nous sommes dans un cas de relation 1 à plusieurs. Il suffit de reporter l'identifiant du client dans la table FACTURE en tant que clef étrangère. Nous obtenons alors :

- CLIENT (id_client, nom, prénom)
- FACTURE (id_facture, date, #id_client)

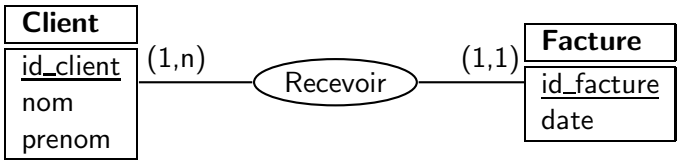


Figure 10 – Un exemple d'association de 1 a plusieurs

**Règle 3 :** une association binaire de type plusieurs à plusieurs devient une table supplémentaire (parfois appelée table de jonction, table de jointure ou table d'association) dont la clef primaire est composée de deux clefs étrangères (qui référencent les deux clefs primaires des deux tables en association). Les attributs de l'association deviennent des colonnes de cette nouvelle table.
Par exemple, toutes les associations du MCD modélisant université sont de type 1 à plusieurs. Elles se traduiront donc par les nouvelles tables suivantes :

- INSCRIRE (id_etudiant, id_diplome, date_inscription)
- SUIVRE (id_etudiant, id_module, note)
- CONTENIR (id_diplome, id_module, coefficient)
- ENSEIGNER (id_module, id_professeur, nb_heures)

**Règle 4 :** une association binaire de type 1 à 1 est traduite comme une association binaire de type 1 à n, sauf que la clef étrangère se voit imposer une contrainte d'unicité en plus d'une éventuelle contrainte de non-vacuité (cette contrainte d'unicité impose à la colonne correspondante de ne prendre que des valeurs distinctes).
Par exemple, dans le cas de la figure 11, l'association Diriger ne sera pas dérivée en table. Nous obtiendrons :

- EMPLOYE (id_employe, nom)
- SERVICE (id_service, nom_service, #responsable)

Dans cette version, l'attribut responsable référence un *id_employe*. Dans les contraintes, responsable doit être non vide, et doit exister dans la table EMPLOYE.

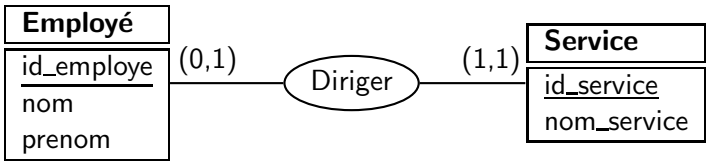


Figure 11 – Un exemple d'association de 1 a 1

**Règle 5 :** une association non-binaire est traduite par une table supplémentaire dont la clef primaire est composée d'autant de clefs étrangères que d'entités en association. Les attributs de l'association deviennent des colonnes de cette nouvelle table.
Par exemple, le MCD de la figure 12 sera dérivé de la manière suivante :

- HORAIRE (id_horaire, date, heure_début)
- FILM (id_film, titre, durée)
- SALLE (id_salle, capacité)
- PROJETER (#id_horaire, #id_film, #id_salle)

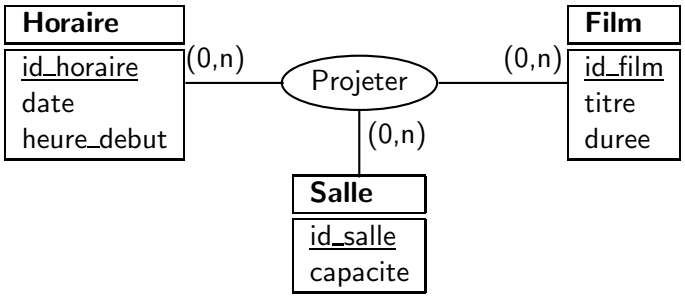


Figure 12 – Un exemple d'association non binaire

## 2.6 Le Modèle Physique des Données

Le modèle logique permet de décrire les différentes tables qu'il faudra matérialiser, ainsi que les relations qui les lient. Il ne décrit cependant pas la technologie qui doit être utilisée, ni le type de stockage qui sera utilisé. Ces actions sont réalisées lors de l'étape de modélisation physique des données. Il s'agit ici de choisir le SGBD que l'on souhaite utiliser (MySQL, Oracle, Access, MongoDB...) et entrer les commandes de création et remplissage des tables.

Ce module couvre le langage SQL / MySQL. Nous étudierons donc dans le présent document les commandes associées a la création et a la gestion d'une base de données Oracle ou MySQL, utilisant toutes deux le langage SQL.

# 3 Le langage SQL

Cette section explique comment créer une base de données et les tables associées. Elle se compose d'exemples en ligne de commande et avec phpMyAdmin.

## 3.1 Création de la base de données

Syntaxe de création d'une base de données :

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

create_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

Crée la base db_name si elle n'existe pas. La clause CHARACTER SET spécifie le jeu de caractères (encodage) par défaut pour les tables de cette base. La clause COLLATE spécifie la collation (règle de comparaison des caractères) par défaut de la base de données.

Pour créer la base de données université, avec un encodage en utf8 :

```
CREATE DATABASE universite DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Avec PhpMyAdmin, il suffit d'entrer le nom d'une nouvelle base de données sur la page d'accueil du logiciel. Une fois le formulaire validé, le nom de la nouvelle base apparaît sur le côté gauche.

**Remarque :** pour pouvoir créer une base de donnée, il faut posséder des privilèges d'administration. Afin d'utiliser une base pour créer des tables ou encore effectuer des requêtes, il faut la sélectionner. Sous phpMyAdmin, cliquer sur le nom de la base souhaité dans le menu gauche. En ligne de commande :

```
USE db_name
```

Avec PhpMyAdmin, cliquer sur le nom de la base. Elle sera alors sélectionnée.

## 3.2 Création de table

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

create_definition:
    column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY
       [index_type] (index_col_name,...)
  | KEY [index_name]
       [index_type] (index_col_name,...)
  | INDEX [index_name]
       [index_type] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX]
       [index_name] [index_type] (index_col_name,...)
  | [FULLTEXT|SPATIAL] [INDEX]
       [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY
       [index_name] (index_col_name,...)
       [reference_definition]
  | CHECK (expr)

column_definition:
    col_name type [NOT NULL | NULL]
      [DEFAULT default_value] [AUTO_INCREMENT]
      [[PRIMARY] KEY] [COMMENT 'string']
      [reference_definition]

type:
    TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  | NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME
  | TIMESTAMP
  | DATETIME
  | CHAR(length) [BINARY | ASCII | UNICODE]
  | VARCHAR(length) [BINARY]
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT
  | TEXT
  | MEDIUMTEXT
  | LONGTEXT
  | ENUM(value1,value2,value3,...)
  | SET(value1,value2,value3,...)
  | spatial_type
```

```
index_col_name:
    col_name [(length)] [ASC | DESC]

reference_definition:
    REFERENCES tbl_name [(index_col_name,...)]
               [MATCH FULL | MATCH PARTIAL]
               [ON DELETE reference_option]
               [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL
    | NO ACTION | SET DEFAULT

table_options: table_option [table_option] ...

table_option:
    {ENGINE|TYPE} = {BDB | HEAP | ISAM
      | InnoDB | MERGE | MRG_MYISAM
      | MYISAM}
  | AUTO_INCREMENT = value
  | AVG_ROW_LENGTH = value
  | CHECKSUM = {0 | 1}
  | COMMENT = 'string'
  | MAX_ROWS = value
  | MIN_ROWS = value
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'string'
  | DELAY_KEY_WRITE = {0 | 1}
  | ROW_FORMAT = { DEFAULT | DYNAMIC |
       FIXED | COMPRESSED }
  | RAID_TYPE = { 1 | STRIPED | RAID0 }
       RAID_CHUNKS = value
       RAID_CHUNKSIZE = value
  | UNION = (tbl_name[,tbl_name]...)
  | INSERT_METHOD = { NO | FIRST | LAST }
  | DATA DIRECTORY = 'absolute path to directory'
  | INDEX DIRECTORY = 'absolute path to directory'
  | [DEFAULT] CHARACTER SET charset_name
      [COLLATE collation_name]

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...
```

La commande de création de table est l'une des plus complètes. Il s'agit de spécifier pour chaque nouvelle table la liste des attributs avec leurs spécifications (contraintes, types) ainsi que le type d'enregistrement de la table. Dans cette section, nous décrirons les parties incontournables de la commande, sans entrer dans les détails des options spécifiques à des contextes particulier. Pour plus de détails, vous pouvez vous reporter au manuel MySQL, dont l'adresse est donnée à la section 3.7.

### 3.2.1 Les types de données

Voici les différents types qu'un attribut peut prendre, pour un entier :

- TINYINT [M] [UNSIGNED] : stocke des nombres entiers allant de -128 à 127 s'il n'est pas UNSIGNED, et de 0 à 255 s'il est UNSIGNED
- SMALLINT [M] [UNSIGNED] : stocke des nombres entiers allant de -32 768 à 32 767 s'il n'est pas UNSIGNED, et de 0 à 65 535 s'il est UNSIGNED
- MEDIUMINT [M] [UNSIGNED] : stocke des nombres entiers allant de -8 388 608 à 8 388 607 s'il n'est pas UNSIGNED, et de de 0 à 16 777 215 s'il l'est.
- INT [M] [UNSIGNED] : stocke des nombres entiers allant de -2 147 483 648 à 2 147 483 647 s'il n'est pas UNSIGNED, et de 0 à 4 294 967 295 s'il l'est.
- INTEGER [M] [UNSIGNED] : même chose que le type INT.
- BIGINT [M] [UNSIGNED] : stocke les nombres entiers allant de -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 s'il n'est pas UNSIGNED, et de 0 à 18 446 744 073 709 551 615 s'il l'est

Dans chaque cas, la valeur [M] représente la taille que peut avoir l'entier. Par exemple, TINYINT 3 permet de stocker des chiffres allant de 0 à 999 (car 999 est composé de 3 chiffres), alors que INT 5 permet de stocker des chiffres allant de 0 à 99 999 (car 99 999 est composé de 5 chiffres).

Voici les différents types qu'un attribut peut prendre, pour un nombre à virgule :

- FLOAT[(M,D)] [UNSIGNED] : stocke des nombres flottants à précision simple allant de -1.175494351E-38 à 3.402823466E+38 s'il n'est pas UNSIGNED et de 0 à 3.402823466E+38 s'il l'est
- DOUBLE [(M,D)] : stocke des nombres flottants à double précision allant de -1.7976931348623157E+308 à -2.2250738585072014E-308, 0, et de 2.2250738585072014E-308 à 1.7976931348623157E+308 s'il n'est pas UNSIGNED et de 0 à 1.7976931348623157E+308 s'il l'est.
- REAL[(M,D)] : même chose que le type DOUBLE
- DECIMAL[(M[,D])] Contient des nombres flottants stockés comme des chaînes de caractères.
- NUMERIC [(M,D)] : même chose que le type DECIMAL

Dans chaque cas, la valeur [M] représente le nombre de chiffres total (virgule comprise) que peut avoir le nombre, et D est le nombre de décimales (nombre de chiffre après la virgule). Par exemple, DOUBLE (5,2) permet de stocker des nombres allant de 0 à 999.99 (999.99 contient 5 chiffres, avec 2 chiffres après la virgule).

Voici les différents types qu'un attribut peut prendre, pour désigner des dates et l'heure :

- DATE : stocke une date au format 'AAAA-MM-JJ' allant de '1000-01-01' à '9999-12-31'
- DATETIME : stocke une date et une heure au format 'AAAA-MM-JJ HH :MM :SS' allant de '1000-01-01 00 :00 :00' à '9999-12-31 23 :59 :59'
- TIMESTAMP [M] : stocke une date sous forme numérique allant de '1970-01-01 00 :00 :00' à l'année 2037. L'affichage dépend des valeurs de M : AAAAMMJJHHMMSS, AAMMJJHHMMSS, AAAAMMJJ, ou AAMMJJ pour M égal respectivement à 14, 12, 8, et 6
- TIME : stocke l'heure au format 'HH :MM :SS', allant de '-838 :59 :59' à '838 :59 :59'
- YEAR : année à 2 ou 4 chiffres allant de 1901 à 2155 ( 4 chiffres) et de 1970-2069 (2 chiffres).

Voici les différents types qu'un attribut peut prendre, pour désigner des textes :

- CHAR (M) : stocke des caractères sur une taille M. Employez donc ce ce type de données pour

des mots de longueur identique.
- VARCHAR (M) [BINARY] : stocke des chaînes de 255 caractères maximum. L'option BINARY permet de tenir compte de la casse.
- TINYBLOB (L) : stocke des chaînes de 255 caractères maximum. Ce champ est sensible à la casse.
- TINYTEXT : stocke des chaînes de 255 caractères maximum. Ce champ est insensible à la casse.
- BLOB : stocke des chaînes de 65535 caractères maximum. Ce champ est sensible à la casse.
- TEXT : stocke des chaînes de 65535 caractères maximum. Ce champ est insensible à la casse.
- MEDIUMBLOB : stocke des chaînes de 16777215 caractères maximum.
- MEDIUMTEXT : chaîne de 16 777 215 caractères maximum. Ce champ est insensible à la casse.
- LONGBLOB : stocke des chaînes de 4 294 967 295 caractères maximum. Ce champ est sensible à la casse.
- LONGTEXT : stocke des chaînes de 4 294 967 295 caractères maximum.

Enfin, les type "inclassables" :
- BIT : même chose que CHAR(1)
- BOOL : même chose que CHAR(1)
- ENUM('valeur_possible1','valeur_possible2','valeur_possible3',...) : 65 535 valeurs maximum.
- SET('valeur_possible1','valeur_possible2',...) : de 0 à 64 valeurs maximum

### 3.2.2 Les contraintes

On discerne deux types de contraintes en SQL :
- Les **contraintes de domaine** spécifient l'ensemble des valeurs qu'un attribut peut prendre
  - *Définir une valeur par défaut* : utiliser la clause DEFAULT suivie par la valeur à affecter.
  - *Forcer la saisie d'un champ* : utiliser la clause NOT NULL empêchera la saisie de valeurs nulles
  - *Emettre une condition sur un champ* : la clause CHECK suivie d'une condition logique
  - *Tester l'unicité d'une valeur* : la clause UNIQUE permet de vérifier que la valeur saisie pour un champ n'existe pas déjà dans la table
- **Les contraintes d'intégrité référentielle** spécifient qu'une valeur insérée dans une table désigne bien un seul enregistrement
  - Les clés primaires assurent l'unicité d'un enregistrement. Elle se définissent grâce à la clause PRIMARY KEY.
  - Les clés étrangères assurent que la connaissance désignée dans une table A existe bien dans une table B. Elles se définissent grâce à la clause FOREIGN KEY (colonne1, colonne2, ...) REFERENCES Nom_de_la_table_étrangère(colonne1,colonne2,...).
  - **Important :** sous MySQL, la gestion des clés étrangères n'est possible que pour les tables de type InnoDB

Dans notre exemple d'université, on note les contraintes suivantes :
- Note sur Suivre doit être supérieur ou égal à 0 et inférieur ou égal à 20. C'est une contrainte implémentée avec la clause CHECK
- La table Suivre référence les tables Etudiants (pour id_étudiant) et Module (pour id_module)

### 3.2.3 Des exemples

Créer la table ETUDIANT, avec l'attribut id_etudiant en clef primaire qui s'auto-incrémente (le système trouvera tout seul une clef unique à lui assigner), le nom et le prénom comme un texte de 20 caractères maximum, l'adresse comme un texte, la date de naissance comme une date :

```
 CREATE TABLE Etudiant (
id_etudiant INT( 2 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
Nom VARCHAR( 20 ) NOT NULL ,
Prenom VARCHAR( 20 ) NOT NULL ,
Adresse TEXT NOT NULL ,
Date_naissance DATE NOT NULL
) ENGINE = InnoDB
```

Créer la table MODULE avec les attributs id_module, nom_module et nb_heure_module :

```
CREATE TABLE `universite`.`Module` (
`id_module` INT( 2 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`Nom_module` VARCHAR( 20 ) NOT NULL ,
`Nb_heures_modules` INT( 3 ) NOT NULL
) ENGINE = InnoDB;
```

Créer la table SUIVRE, qui référence ETUDIANT et MODULE :

```
CREATE TABLE Suivre (
id_module INT( 3 ) NOT NULL ,
id_etudiant INT( 3 ) NOT NULL ,
Note DOUBLE( 4,2 ) NOT NULL ,
CONSTRAINT pk_suiv PRIMARY KEY (id_module, id_etudiant),
CONSTRAINT fk_suiv_mod FOREIGN KEY (id_module) REFERENCES MODULE (id_module),
CONSTRAINT fk_suiv_et FOREIGN KEY (id_etudiant) REFERENCES ETUDIANT (id_etudia
CONSTRAINT chk_Note CHECK (Note>=0.0 AND Note <= 20.0)
) ENGINE = InnoDB
```

Test sur la clef primaire :

```
INSERT INTO Etudiant (id_etudiant ,Nom) VALUES (1, 'Durant')
INSERT INTO Etudiant (id_etudiant ,Nom) VALUES (1, 'Dupond')
```

La première ligne passe, mais la seconde renvoie : *#1062 - Duplicate entry '1' for key 'PRIMARY'*, car la clef primaire 1 existe déjà. Pour ne pas avoir d'erreur, il faut insérer une clef unique :

```
INSERT INTO Etudiant (id_etudiant ,Nom) VALUES (2, 'Dupond')
```

Test sur les clefs étrangères (l'étudiant 5 n'existe pas) :

```
INSERT INTO `Suivre` ( `id_module` , `id_etudiant` , `note`)
VALUES ( 1, 5, 15.9 )
```

Revoie : *#1452 - Cannot add or update a child row : a foreign key constraint fails (‘universite‘.‘Suivre‘, CONSTRAINT ‘fk_suiv_et‘ FOREIGN KEY (‘id_etudiant‘) REFERENCES ‘Etudiant‘ (‘id_etudiant‘))*

Avec PhpMyAdmin, utiliser le formulaire “Créer une nouvelle table sur la base” une fois la base sélectionnée. Vous devez donner un nom et le nombre d'attributs de la table. Un formulaire s'ouvre alors, permettant pour chaque attribut de spécifier son nom, son type, sa taille, sa valeur par défaut, si c'est une clef primaire ou non. N'oubliez pas de choisir “InnoDB” dans le moteur de recherche avant d'enregistrer si vous souhaitez référencer des clefs étrangères. Par contre, il n'est pas possible de créer ces clefs étrangères avec PhpMyAdmin. Vous devrez utiliser les commandes de modification de table décrites 3.3.

## 3.3 Modification de table

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
  | ADD INDEX [index_name] [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [index_name] [index_type] (index_col_name,...)
  | ADD [FULLTEXT|SPATIAL] [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | table_options
```

Quelques exemples utiles :

```
ALTER TABLE `Etudiant` DROP `Nom`;
ALTER TABLE `Etudiant` ADD `Nom` VARCHAR( 20 ) NOT NULL;
ALTER TABLE `Enseigner` ADD CONSTRAINT fk_ens_prof
    FOREIGN KEY (id_professeur) REFERENCES Professeur (id_professeur);
```

Pour modifier une table avec PhpMyAdmin, sélectionnez la table, puis allez dans l'onglet "structure". La liste des champs s'affiche. Pour les modifier, cliquez sur l'image représentant un crayon à côté de l'attribut.

## 3.4 Suppression de table

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

Exemple :

```
DROP TABLE `Etudiant`
```

Avec PhpMyAdmin, sélectionnez la table, puis cliquez sur l'onglet "supprimer".

## 3.5 Consultation des données

### 3.5.1 Sélection simple des données

En MySQL, la sélection des données se fait avec la commande SELECT :

```
SELECT [STRAIGHT_JOIN]
       [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
       [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
       [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO {OUTFILE | DUMPFILE} 'nom_fichier' export_options]
    [FROM table_references
      [WHERE where_definition]
      [GROUP BY {unsigned_integer | nom_de_colonne | formula} [ASC | DESC], ...
      [HAVING where_definition]
      [ORDER BY {unsigned_integer | nom_de_colonne | formula} [ASC | DESC] ,...]
      [LIMIT [offset,] lignes]
      [PROCEDURE procedure_name(argument_list)]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

- La clause SELECT est suivie des colonnes que l'on souhaite sélectionner
- La clause ALL (équivalente de *) désigne tous les attributs
- La clause DISTINCT supprime les doublons à l'affichage
- La clause FROM désigne la ou les tables à partir desquelles on lit les données
- La clause WHERE permet de restreindre la sélection sur les valeurs des attributs

```
SELECT * FROM Etudiant; //Selectionner tous les etudiants
SELECT * FROM Suivre WHERE Note > 15 AND Note < 18 AND id_module = 3;
//Selectionner tous les id_etudiant, id_module, notes dont les notes sont com
```

```
et dont le module a pour identifiant 3
SELECT m.id_module, s.id_module, Suivre.note FROM
  Module m, Suivre s; //Remarquez la notation pointée qui permet de
                              discerner les attributs de chaque table
```

Avec PhpMyAdmin, sélectionnez la table, puis allez dans l'onglet "Rechercher".

### 3.5.2 Les fonctions SQL

MySQL offre un panel de fonctions sur les dates, les chaînes de caractères ou encore permettant d'effectuer des opérations d'aggrégation. Voici les plus utiles :

- count permet de compter le nombre d'enregistrements
- sum permet de sommer les valeurs de plusieurs enregistrements
- max permet de sélectionner l'enregistrement ayant la valeur maximale
- min permet de sélectionner l'enregistrement ayant la valeur minimale
- avg permet de connaître la moyenne sur plusieurs enregistrements
- char_length permet de connaître la longueur d'une chaîne de caractères
- expr LIKE pat permet de comparer des chaînes de caractères
- curdate permet de connaître la date courante
- datediff(expr,expr2) retourne le nombre de jours entre la date expr et la date expr2

```
SELECT count (DISCTINCT id_etudiant) FROM Etudiant; //Connaitre le nombre
d'etudiants de la base
SELECT sum (nb_heure_module) FROM ENSEIGNER WHERE id_professeur = 3;
// Connaître le nombre d'heures d'enseignements du professeur
3 tout module confondu
SELECT id_etudiant FROM Suivre WHERE note = MAX (note);
// Quel sont les etudiants ayant obtenus la plus grande note ?
SELECT avg (note) FROM Suivre; // Moyenne des notes de l'universite
```

### 3.5.3 La jointure

L'intérêt de MySQL est de pouvoir combiner les tables afin d'effectuer des requêtes plus complexes. Cette opération, appelée **jointure** consiste à sélectionner un attribut pivot entre deux tables afin de les combiner. Par exemple, supposons que l'on veuille connaître le nom des etudiants qui ont suivi le module Informatique. Dans cet exemple, nous ne connaissons pas l'identifiant de l'enregistrement Informatique. Cette requête nécessite l'utilisation de trois tables :

- La table Etudiant, qui contient le nom et l'identifiant des etudiants
- La table Module, qui contient le nom et l'identifiant du module Informatique
- La table Suivre, qui contient l'ensemble des identifiants des etudiants suivant le module Informatique

Ces tables peuvent être jointes entre elles via deux pivots :

- La table Etudiant peut être jointe à la table Suivre via l'attribut id_etudiant
- La table Module peut être jointe à la table Suivre via l'attribut id_module

```
SELECT e.Nom as nom_etudiant, m.Nom as nom_module
FROM Etudiant e, Module m, Suivre s
WHERE e.id_etudiant = s.id_etudiant
```

```
AND m.id_module = s.id_module
AND m.Nom LIKE 'Informatique'
```

### 3.5.4 Les regroupements

MySQL permet également de regrouper les données pour l'affichage. Pour cela, il faut utiliser la clause GROUP BY :

```
SELECT e.id_etudiant, e.nom, m.id_module
FROM Etudiant e, Module m
WHERE e.id_etudiant = m.id_etudiant
GROUP BY m.id_module
```

## 3.6 Mise à jour des données

En base de données, la mise à jour des données désigne l'insertion, la suppression et la modification des données.

### 3.6.1 Insertion de tuple

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

On discerne deux manières d'insérer : soit on ne spécifie pas les attributs qui seront remplis, et dans ce cas il faut remplir tous les attributs dans la clause VALUES ; soit on spécifie les attributs qui seront renseignés, et on ne met que ces valeurs dans la clause VALUES.

```
INSERT INTO Etudiant VALUES (4, 'Dubois', 'Thomas', '678␣place␣...', '01/01/19
INSERT INTO Etudiant (nom, prenom) VALUES ('Durant', 'Helene');
```

### 3.6.2 Modification de tuple

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

```
UPDATE Module SET nb_heure_module=75 WHERE Nom='Informatique';
```

### 3.6.3 Suppression de tuple

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
       [WHERE where_definition]
       [ORDER BY ...]
       [LIMIT row_count]
```

```
DELETE FROM Module WHERE Nom LIKE 'Informatique';
DELETE FROM Suivre WHERE note < 0 OR Note > 20.0;
```

### 3.7 De la lecture

- Le manuel de MySQL : `http://dev.mysql.com/doc/refman/5.0/fr/index.html`
- Un cours complet de MySQL : `http://cyberzoide.developpez.com/php4/mysql/`

## 4 La fouille de données

La fouille de données est l'étape centrale du processus d'extraction de connaissances. Elle consiste à découvrir de nouveaux modèles au sein de grandes quantités de données. Cependant, il est rarement possible d'appliquer directement la fouille de données sur les données brutes. Les premières opérations du processus ECD correspondent donc à la transformation des données avant de pouvoir appliquer des algorithmes de fouille de données.

Dans un premier temps, les bases de données qui serviront à l'extraction sont sélectionnées. En effet, il est courant que les données ne proviennent pas des mêmes sources et soient enregistrées sous divers formats. Cette phase d'acquisition consiste alors en diverses tâches d'intégration et de nettoyage : repérer lors de la sélection les inconsistances, les données trop bruitées, les nettoyer avant de les stocker dans les bases de données ciblées. Par exemple, il est courant dans le contexte médical de vouloir intégrer aux données brutes issues d'expérimentations les connaissances relatives au domaine. Ainsi, l'expert pourra, en plus de ses données expérimentales, sélectionner les annotations sémantiques s'y rapportant, souvent disponibles sous la forme d'ontologies. Les annotations apportent une information supplémentaire, qui peut être utile lors du processus de fouille. Par exemple, on peut annoter les gènes d'une base par leur rôle fonctionnel. Durant cette étape, les données mal renseignées au cours de l'expérimentation ou encore dupliquées seront supprimées, et les sources de données sémantiques seront stockées sous la forme d'une base de données.

La seconde étape est une étape de prétraitement en vue de fabriquer les jeux de données adéquats à l'étape de la fouille. Il s'agira dans ce cas de sélectionner les items appropriés au processus décisionnel en cours, normaliser les données, les agréger, réduire le nombre de dimensions etc... Par exemple, dans un contexte médical, si l'expert souhaite étudier les données de certains gènes, il n'est pas nécessaire de conserver tous ceux présents dans la base originale. Cela aura pour conséquence de réduire la taille de la base de données, et donc d'augmenter les chances de succès de l'algorithme de fouille.

La fouille de données permet alors d'extraire des schémas qui modélisent ou synthétisent l'information contenue dans les données préalablement traitées. Selon les besoins et objectifs de la fouille, les schémas sont extraits par différentes techniques :

- la **classification**, dont le but est d'affecter des données à des classes préalablement définies ;
- le **clustering** (ou *segmentation*) qui permet de partitionner les données en sous-ensembles (ou groupes) de telle manière que la similarité entre les données d'un même cluster et la dissimilarité entre différents clusters soient les plus grandes possibles ;
- la **description des données** qui peut être réalisée à l'aide des règles d'association ou des motifs séquentiels, qui permettent d'extraire des corrélations tenant compte ou non d'une notion d'ordre.

Enfin, les schémas extraits sont ensuite analysés, interprétés et validés par l'expert. Durant cette étape, il est possible d'utiliser des techniques de visualisation de données, qui regroupent tous ou une partie des résultats, et permettent à l'expert de raffiner manuellement le résultat des fouilles. L'expert dispose également de mesures de qualité afin d'évaluer la pertinence des schémas découverts.

Il existe une grande quantite d'algorithmes permettant d'effectuer les taches enoncees ci-dessus, et les enumerer ou encore les detailler sort du cadre de ce cours. La fouille de donnees sera abordee en TP au travers de l'utilisation du logiciel libre WEKA.