Systèmes d'Information
Bases de Données Relationnelles
L3 MIASHS - IDS

Guillaume Metzler
guillaume.metzler@univ-lyon2.fr

INSTITUT
DE LA
COMMUNICATION

Université de Lyon, Lyon 2, ERIC EA3083, Lyon, France

Automne 2020

Organisation du cours

Un cours décomposé en trois parties

- 21h de cours magistraux (6 séances)
- environ 9h de travaux dirigés (3 séances)
- environ 9h de travaux pratiques (6 séances)

Certaines heures de CM se transformeront en heures TD pour vous laisser du temps pour pratiquer les différentes notions

Evaluation

- Un contrôle continu de trois heures
- Une évaluation au cours d'une séance de TP

Organisation du cours

Je ne ferai pas de distinction entre CM et TD, je mélangerai ces deux parties du cours au sein d'une même séance

Je souhaiterai transformer la dernière séance de TD (du vendredi matin) en cours d'Intro au Machine Learning et parler d'orientation avec vous

Idem, je pense transformer la ou les deux dernière(s) séance(s) de TP en séance(s) de TP consacrée(s) au Machine Learning

Sommaire

Contenu du cours :

- Introduction rapide sur les bases de données
- Langage SQL : présentation des fonctions
- Modèle relationnel de Codd : troisième forme normale, dépendance fonctionnelles, normalisations
- Formalisme Entité - Association : élaboration d'un schéma conceptuel, schéma de la base de données

On finira par une étude de cas pour la *modélisation conceptuelle des données* et sa traduction pour l'obtention d'un modèle physique, *i.e.* la construction de la base de données.

Avant propos

Le cours présenté se base sur

- les cours présentés par les intervenants des années précédentes
- un cours dispensé à l'UJM de Saint-Etienne par *Francois Jacquenet*
- un livre de Jean-Luc Hainaut, *Bases de Données et modèles de calcul* (Jean-Luc-Hainaut, 2005)

Introduction

Système d'information

Une définition ?

Un **système d'information** est l'ensemble des ressources (matériels, logiciels, données, procédures, ...) structurés pour acquérir, traiter, mémoriser et rendre disponible l'information sous de multiples formes (textes, sons, images, ...) dans et entre plusieurs organisations.

Aujourd'hui, le système d'information permet d'automatiser et de dématérialiser quasiment toutes les opérations incluses dans les activités ou procédures de notre vie quotidienne : personnelle ou professionnelle.

Si l'on doit résumer en quatre étapes, son rôle est de :

COLLECTER $\rightarrow$ STOCKER $\rightarrow$ TRAITER $\rightarrow$ DIFFUSER

Bases de données : Besoins

Des besoins dans le domaine de la gestion de l'information

- **Décrire** l'information
- **Manipuler** l'information
- **Interroger** la collection d'informations
- **Exactitude** et **Cohérence**
- **Garanties** tant en terme de fiabilité que de contrôle
- **Confidentialité**
- **Efficacité**

Besoin de pouvoir décrire

Décrire les données de l'application. Par exemple, pour la SNCF cela correspond aux *trains*, *trajets* ou encore *réservations*, *conducteurs*. Cette description doit être effectuée sans faire référence à une solution informatique particulière.
—> **Modélisation conceptuelle**

Elaborer une description équivalente pour le stockage des données dans la SGBD choisie
—> **Modélisation logique**

On utilisera ensuite un langage de description des données

Besoin de pouvoir manipuler

Créer la base de données initiales avec les données représentant le réseau SNCF
→ **Langage permettant l'insertion de données**

Créer au fur et à mesure les données sur les réservations. Modifier si besoin et éventuellement supprimer toute donnée déjà rentrée
→ **Langage de Manipulation de Données (insertion, modification, suppression) :SQL**

Besoin de pouvoir interroger

Répondre à toute demande d'information portant sur les données contenues dans la base.
Par exemple :

- Jean Lestrade a-t-il une réservation pour aujourd'hui ?
  Si oui, donner les informations connues sur cette réservation.
- Quels sont les horaires des trains de Lyon à Strasbourg entre 9h et 10h le dimanche ?
- Donner les destinations au départ de Lyon sans arrêt intermédiaire.

→ **Langage de requête ou d'interrogation : SQL**

Besoin d'exactitude et de cohérence

Il faut pouvoir exprimer toutes les règles qui contraignent les valeurs pouvant être enregistrées de façon à éviter toute erreur qui peut être détectée.

Par exemple :

- Il ne faut jamais donner la même place dans le même train à deux clients
- Les arrêts d'un train sont numérotés de façon continue (il ne peut y avoir pour un train donné un arrêt n3 s'il n'y a pas un arrêt n2 et un arrêt n1)
- La date de réservation pour un train doit correspondre à un jour de circulation de ce train
- L'heure de départ d'une gare doit être postérieure à l'heure d'arrivée dans cette gare
- L'heure d'arrivée à un arrêt doit être postérieure à l'heure de départ de l'arrêt précédent

$\rightarrow$ **Langage d'expression de contraintes d'intégrité**

Besoin de garanties

Il ne faut pas que les informations (par exemples, les réservations) soient perdues àcause d'un dysfonctionnement quelconque : erreur de programmation, panne système, panne d'ordinateur, coupure de courant, plantage des serveurs, ...
→ **Garantie de confidentialité**

Il ne faut pas qu'une action faite pour un utilisateur (par exemple l'enregistrement d'une réservation) soit perdue du fait d'une autre faite simultanément pour un autre utilisateur (réservation de la même place).
→ **Garantie de contrôle de concurrence**

Besoin de confidentialité

Toute l'information doit pouvoir être protégée contre l'accès par des utilisateurs non autorisés que ce soit

- en lecture
- en écriture

Interdire par exemple aux clients de modifier les numéros de trains ou les horaires ou leur réservation.

→ **Garantie de confidentialité**

Besoin d'efficacité

Le temps de réponse du système doit être conforme aux besoins :

- en interactif : pas plus de deux secondes
- en programmation : assez rapide pour assumer la charge de travail attendue (nombre de transactions par jour).

  Usage de mécanismes d'optimisations et, éventuellement, répartition/duplication des données sur plusieurs sites

Base de Données

Les **Bases de Données** sont des collections de données qui sont structurées et cohérentes.

Les informations sont organisées de manières à être facilement triées, classées et modifiées par le biais d'un logiciel appelé **Système de Gestion de Base de Données** (SGBD).

Le travail effectuer, de conception et de structure de l'information, doit répondre de façon pérenne à un problème, la structure se doit être évolutive et **ne pas répondre uniquement à un problème ponctuel**.

Plus de précision

Le **Système de Gestion de Bases de Données** est un logiciel permettant de

- **définir une représentation** des informations
- **stocker, interroger** et **manipuler** de **grandes quantités de données** (plus que la mémoire vive)
- garantir la **longévité** des données
- garantir **l'accessibilité** de manière **concurrente** (plusieurs utilisateurs simultanés)
- assurer une **très grande fiabilité**

Composants logiciels d'une base de données

SGBD

- gère le niveau logique et physique de la base
  selon l'architecture ANSI-SPARC

Les outils frontaux L4G

- générateurs : de formes, de rapports, des applications
- intégrés au SGBD ou externes
  Powerbuilder, Borland ...
- Interfaces WEB : HTML, XML, ...
- Interface OLAP & Data Mining
  Intellignet Data Miner (IBM)

Utilitaires : chargement, statistiques, aide à la conception, ...

Qui utilise des Bases de Données

Les utilisateurs interactifs

- ils cherchent des informations sans connaître la Base de Données
- utilisent des interfaces (formulaires, web, ...)
- peuvent à la rigueur utiliser des langages tels que QBE

Les programmeurs d'applications

- construisent les interfaces pour les utilisateurs interactifs
- spécialistes de SQL

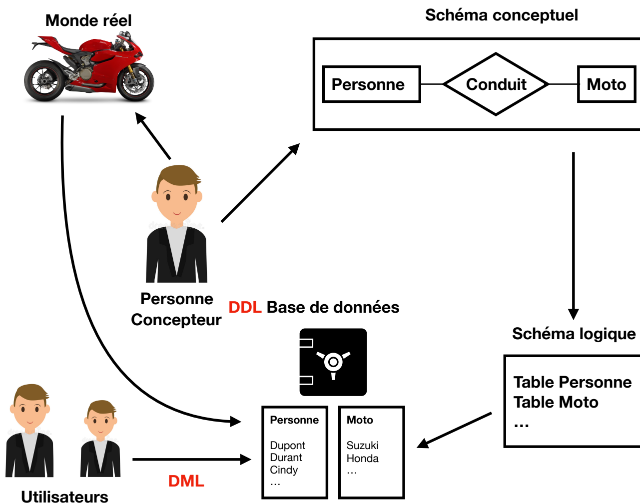
Les DBA (DataBaseAdminstrators)

- modélisent les bases de données
- créés et maintiennent les bases de données
- ont la priorité sur tous les autres usagers
- peuvent être très bien payé ... car ils sont à la base du BI !

Formalisme Entité - Association

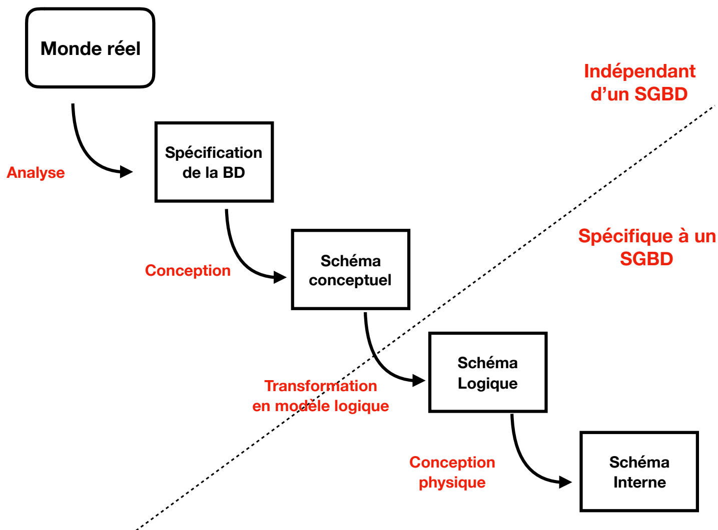
Cycle de vie d'une base de données

De multiples interactions



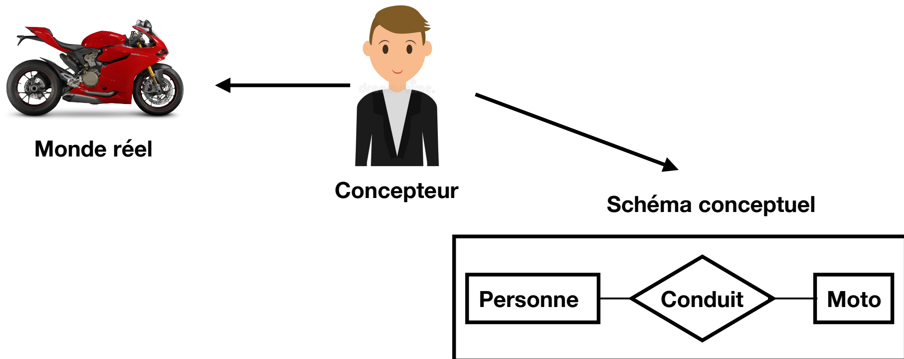
Cycle de vie d'une base de données

Une représentation plus schématique et détaillée



Focus

Ce qui va nous intéresser pour le moment c'est comment passer d'une *problématique réelle* au *schéma conceptuel* et comment écrire un tel schéma.



Rôle du concepteur

Le rôle du concepteur est fondamental ! C'est à lui que revient la charge de transcrire les observations du monde réel de façon abstraite en prenant en compte tous les liens et leurs caractéristiques. Il doit donc

- réfléchir à la structure de base (tables, attributs, relations) de façon à prendre en compte tous les aspects du problème
- avoir une représentation cohérente et qui corresponde à la réalité telle qu'elle est perçue par les utilisateurs.

C'est une étape d'autant plus importante qu'elle servira de fondations à ce qui suivra : cohérence - facilité des manipulations - logique.
C'est également une étape qui se veut indépendante de la technologie utilisée (on ne fera pas mention du langage SQL pour le moment !)

Rôle du concepteur

Pour faire simple, le concepteur doit :

- se mettre à la place de l'utilisateur
- rester fidèle à la réalité observée
- développer une représentation simple et compréhensible relative à l'application

Cela permet aussi de définir *un bon schéma conceptuel* : simplicité - fidélité - longévité - portabilité - compréhensibilité - ...

Elaborer un schéma conceptuel

Il faut tout d'abord analyser le monde réel : identifier les différents phénomènes à représenter dans la base de données

Les caractériser : définir leurs caractéristiques : contenu - structure - règles

Réalité perçue → **Représentation**

Faire abstraction des particularités : passer des *objets* aux *types ou classes d'objets*

Type d'objet : personne

Propriétés :

- **Nom**
- **Âge**
- **Poids**
- **Taille**
- **...**

De quoi s'agit-il ?

Lorsque l'on cherche à modéliser un système d'informations, on peut faire appelle à deux *formalismes* :

- le formalisme entité/association
- le formalisme UML (*Universal Modelling Language*) : c'est un ensemble de notations qui permet de représenter divers aspects d'une application informatique (1990)

Mais on va se focaliser sur le modèle entité/association : traduction d'une problématique réelle à l'aide d'un graphique → passage schéma conceptuel. Pour ensuite aboutir au schéma logique.

A propos du modèle EA

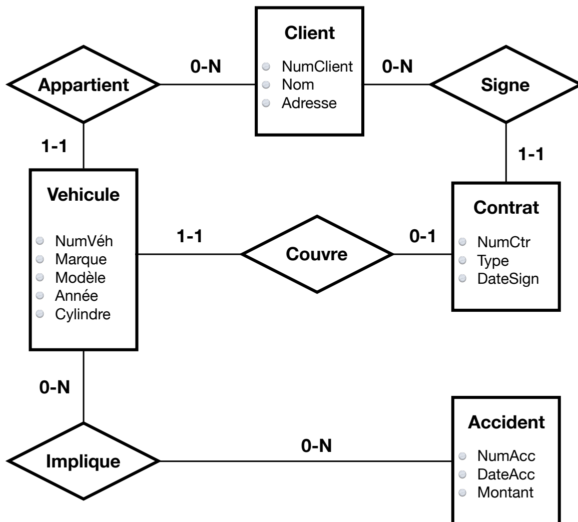
C'est certainement le modèle conceptuel le plus utilisé actuellement.

Il a été conçu en 1976 par Peter Pin-Shan Chen dans un article intitulé :

the Entity-Relationship Model: Toward a unified View of Data, ACM Transactions on Database Systems, volume 1, pages 9-36, 1976.

Il existe bien sûr d'autres modèles conceptuels : UML ou MERISE

Exemple



Questions

Que sont alors qu'une entité et une association ? Quel lien avec les bases de données

En fait c'est très simple mais un peu abstrait avouons-le !

Un schéma comme une collection de types :

- entités $\leftrightarrow$ objets
- associations $\leftrightarrow$ liens
- attributs $\leftrightarrow$ caractéristiques

La base de données, quant à elle, contiendra les valeurs des attributs (ou propriétés) des différentes entités.

Définitions

Entité

C'est un objet qui peut être à la fois concret ou abstrait, ayant une existence propre. Il peut très bien s'agir de personnes, d'aliments, voitures ou encore de molécules chimiques. Ces entités admettent un identifiant, qui vont permettre de les distinguer, mais aussi une liste de *propriétés, attributs*. L'ensemble des entités est regroupé sous le nom de *type d'entités* lorsque plusieurs entités présentent des caractéristiques semblables.

Exemple :

Dans un contexte d'assurance automobile, on peut retrouver plusieurs classes - type d'entités : VOITURE - PERSONNE - CONTRAT - ACCIDENT

Représente une population et peuvent donc être de différentes natures

Définitions

Propriété, attributs

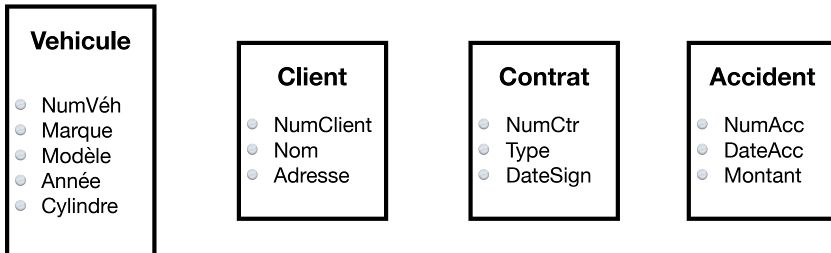
Il s'agit d'une valeur ou d'une modalité que peu prendre l'un des descripteurs. Elle permet de décrire l'entité. L'attribut peut lui même prendre plusieurs valeurs et permettra de distinguer les différents éléments de la population représentés par l'entité.

Exemple

CLIENT : *numéro de client - Nom - Adresse* VOITURE : *Numéro Véhicule - Marque - Modèle - ...*
Comment caractériseriez vous l'entité CONTRAT ou encore ACCIDENT ? *i.e.* quels sont ses attributs ?

Exemple d'entités avec leurs attributs

Si on reprend notre exemple des contrats automobiles



$\rightarrow$ on a 4 entités décrites par respectivement 5, 3, 3 et 3 attributs.

Nature des attributs

Un attribut peut être par nature **simple** ou **complexe**

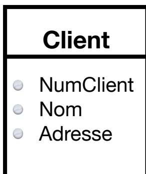
- **Simple** dans le sens où l'attribut est non décomposable : jour de l'année, prénom, année de naissance, lieu, numéro de immatriculation d'un véhicule, numéro de contrat.
  Les valeurs prises sont *simples* et sont aussi bien des nombres que des chaines de caractères
- **Complexe** dans le sens où il est décomposable : une date qui se décompose en jour - mois - année ou adresse postale complète.
  Il se caractérise donc comme une composition d'attributs simples voire même complexe.

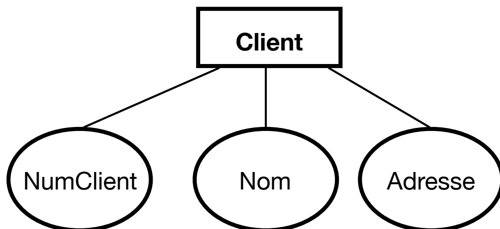
Nature des attributs

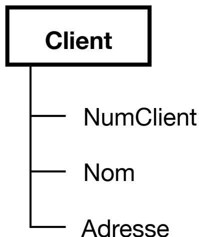
Un attribut peut également être **obligatoire** ou **facultatif**

- **Obligatoire :** au moins une valeur est attendue pour cet attribut, *i.e.* le cardinal de l'ensemble des valeurs possibles est au moins égal à 1.
  Ex : Nom, Prénom
- **Facultatif :** il n'est pas nécessaire d'affecter une valeur, le cardinal de l'ensemble des valeurs possibles peut être nul.
  Ex : montant de l'accident, salaire, numéro de téléphone, ...

Autres notations possibles





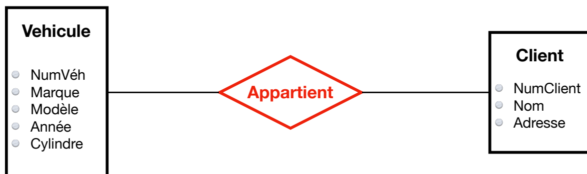


Définitions

Associations

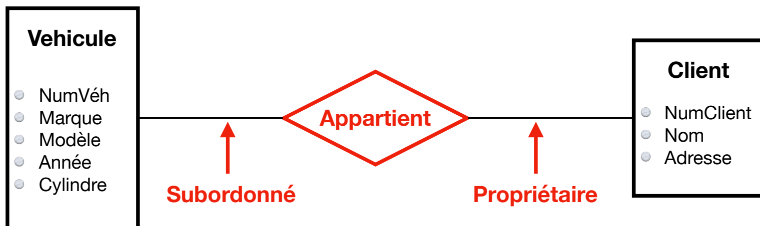
Représentation d'un lien non orienté entre plusieurs entités.
On désignera par type d'associations : la représentation d'un ensemble d'associations ayant la même sémantique et décrites par les mêmes caractéristiques

Exemple



Associations - liens

Reprenons notre association (binaire)

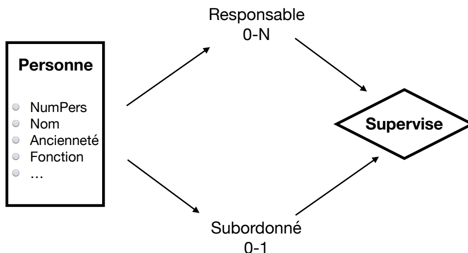


- Dans une association, chaque entité va jouer son un rôle précis
- Dans le cas d'une association binaire, on identifiera deux rôles

Associations - liens

Exemple d'une association binaire cyclique

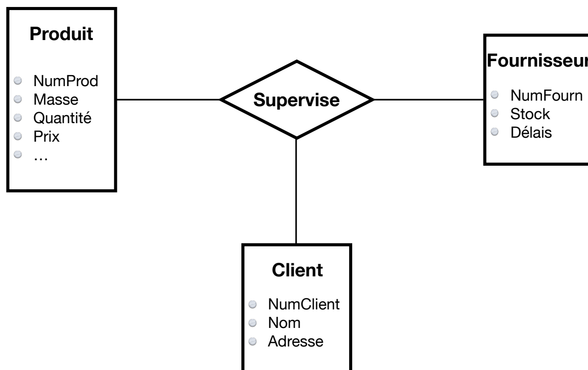
Il s'agit du cas où une entité est en relation avec elle même, cela peut être le cas si l'on raisonne sur l'entité PERSONNE avec la relation *supervise* dans le cadre d'une entreprise.



Important : spécifier le rôle de chaque entité dans ce cas là

Associations - liens

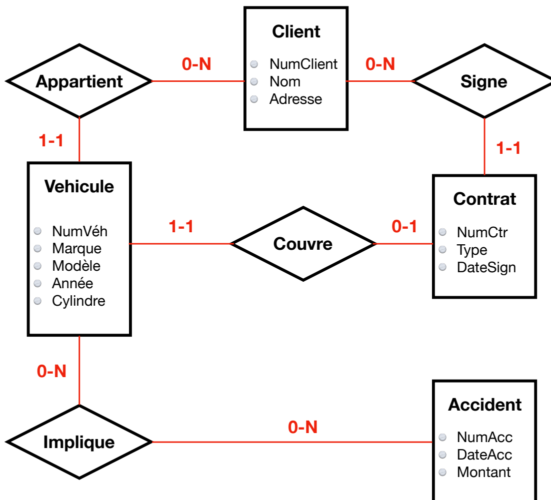
Exemple d'une association ternaire : commerce



On peut imaginer avoir des associations ternaires cycliques ou mettant un jeu encore plus d'entités, mais cela est rarement le cas en pratique

Un dernier point

Il nous reste un dernier point à définir : la cardinalité



Point vocabulaire

Cardinalité

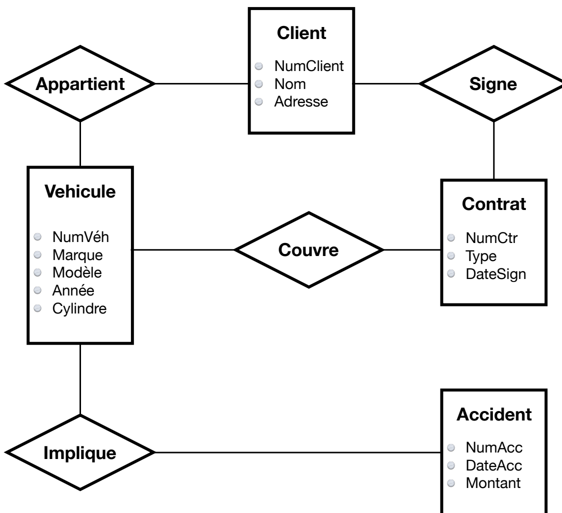
Il s'agit d'un indicateur qui montre combien, pour chacune des entités en relation, à combien d'associations chaque entité peut ou doit participer.
Il existe trois types de *cardinalités* (également appelés types *d'associations*) : *associations* ***1-1*** - *associations* ***1-N*** - *associations* ***N-N***.

De façon plus précise, on indique les participations minimales et maximales des entités à l'association

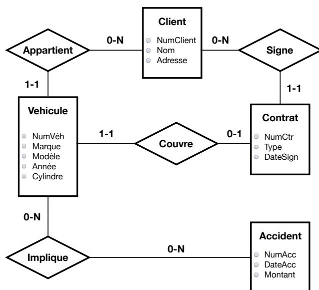
- 1-1 : une seule et unique relation
- 0-1 : au plus une relation
- 0-N : plusieurs relations possibles
- 1-N : au moins une relation
- M-N : entre M et N relations

Exemple

Indiquer la cardinalité des associations suivantes



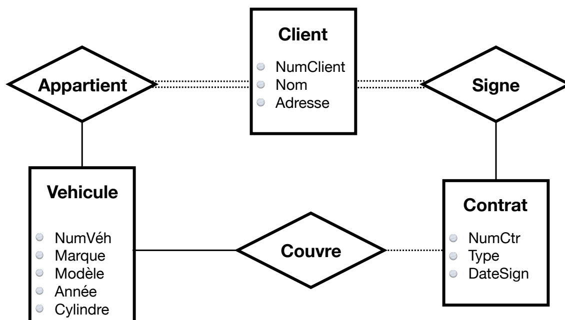
Exemple



- un client **peut ne pas** posséder de véhicule ou en posséder **plusieurs**
- un véhicule appartient à **un seul et unique** client
- un accident peut **ne pas** ou impliquer **plusieurs** véhicules
- un véhicule est couvert par **une seul et unique** assurance
- un contrat est signé par **un seul et unique** client
- un contrat couvre **au maximum** un véhicule
- ...

Autres notations

Associations	Schémas
0-1	
1-1	
0-N	
1-N	
N-M	



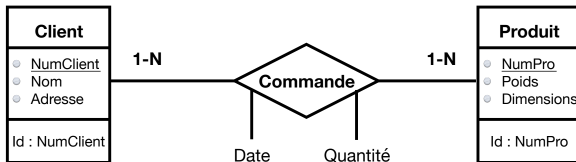
Associations porteuses de propriétés

Remarque

Dans des associations de type M-N, il est possible de caractériser l'association par des attributs pour en clarifier la nature.

Exemple

On peut spécifier la relation *Commande* entre *Produit* et *Client* en indiquant la date de la commande ainsi que la quantité commandée.



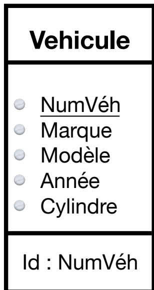
Identifiants

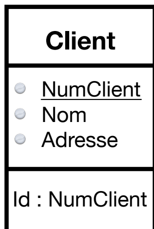
En général, mais cela n'est pas toujours le cas, un type d'entités est doté d'un attribut qui identifie les entités de ce type. Cet identifiant permet donc de **distinguer** les différentes entités d'un même type.

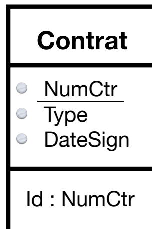
Le plus souvent, cet identifiant constitue la **clef primaire** de notre table, on l'appelle aussi **identifiant primaire**.

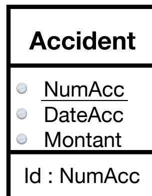
Notations : il est d'usage de faire précéder par *id :* l'attribut constituant l'identifiant primaire et de le souligner dans la liste des attributs.

Identifiants









Identifiants hybrides

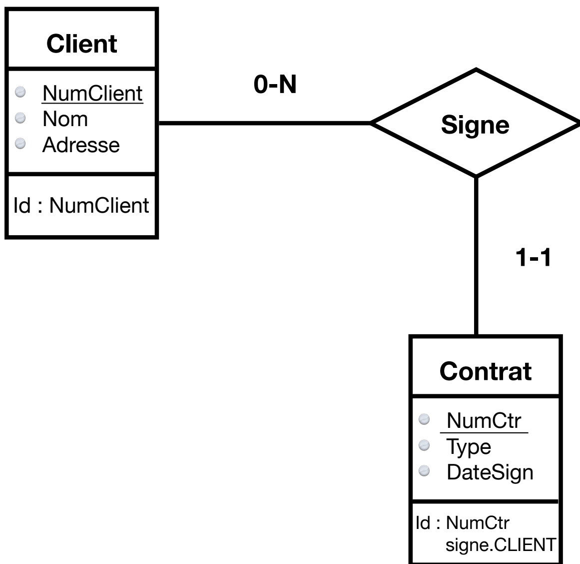
Reprenons le cas du type d'entités CONTRAT qui peut s'avérer plus complexe.

Un client peut potentiellement signer **plusieurs** contrats **différents** (imaginons qu'ils soient numérotés).

Si l'on souhaite donc **associer un contrat à un client particulier**, outre le numéro du contrat il est également nécessaire de **prendre en compte un autre attribut qui permettrait d'identifier clairement le client en question**.

Ces identifiants font références à une table précise (ici CLIENT) d'où la nature **hybride** —> identifiants issus de deux tables.

Identifiants hybrides



Quelques remarques - liens : identifiants/clé

Clé primaire

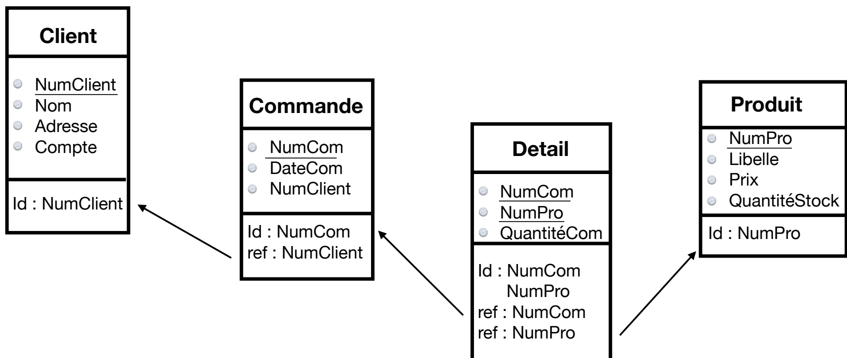
Il s'agit d'un attribut ou d'un ensemble d'attributs dont les valeurs permettent de distinguer les n-uplets les uns des autres (on retrouve cette notion d'identifiant).
Ex : *NumClient* est une clé primaire de l'entité CLIENT.

Clé étrangère

Il s'agit d'un attribut qui est une clef primaire d'une autre table ou autre type d'entités.
Elle joue un rôle important dans la notion de *contraintes référentielles* par exemple ou sont très pratiques pour créer une dépendance explicite entre deux tables.
La clé étrangère peut faire référence à n'importe quel identifiant d'une autre table (type d'entités), on choisira, la plupart du temps, son identifiant primaire.

Exemple

Un exemple graphique pour la notion de *références* (étrangères)



Remarques

La notion de références à une table nécessite d'imposer des règles logiques de structure et de contenu des tables qui sont liées entre elles.

Les références ne peuvent être effectuées n'importe comment et il est nécessaire de respecte une cohérence entre les tables.
Modifier une table doit entraîner des modifications dans les tables dépendantes et aussi prévenir de toutes modifications qui entraineraient une incohérence (logique ou non) dans la base de données.
De telles contraintes peuvent également intervenir au sein d'une table en elle-même

$\rightarrow$ **Travail sur la notion de contraintes d'intégrités**

Contraintes d'intégrités

Définition - Remarque

- Il s'agit d'un ensemble de règles définissant les états (CI statiques) et les transitions d'états (CI dynamique) possibles de la base de données
- Ces règles doivent être décrites explicitement (dans le langage approprié) si elle ne peuvent pas être décrite avec les concepts du modèle de données
- Une base de données sera qualifiée de cohérente si toutes les CI définies sont respectées par les deux de la base de données.

Regardons cela sur quelques exemples pour démystifier tout cela.

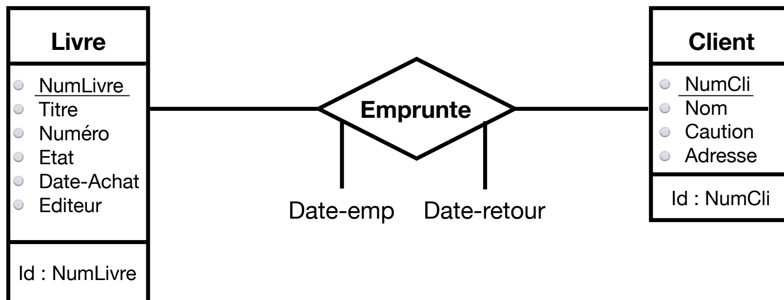
Quelques exemples

Il existe différentes contraintes portant directement sur les attributs :

- les contraintes d'unicité en liant avec la notion d'identifiant –> contraintes lors de la création ou de la modification d'un ligne.
- les contraintes de domaines : un attribut ne peut prendre ses valeurs que dans un champ de valeurs restreint
  Ex : les notes à un examen sont comprises entre 0 et 20
- les contraintes référentielles : elles précisent que certaines colonne d'une table, appelées clé étrangère (cf. ce qui a été vu plus tôt dans le cours) doivent à tout instant, pour chaque ligne, contenir les valeurs qu'on retrouve comme identifiant primaire d'une ligne dans une autre table.

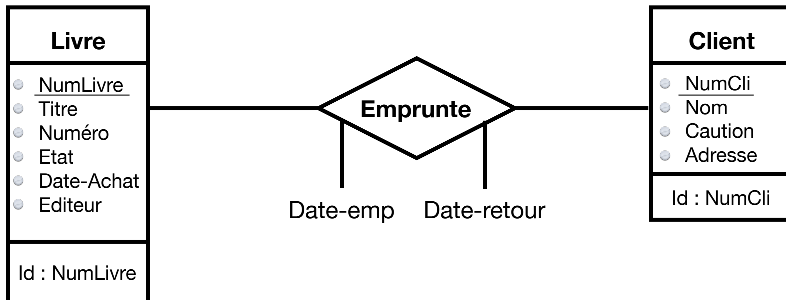
Quelques exemples

Considérons le modèle entité association suivant, citez moi au moins deux contraintes d'intégrités entre les attributs.



Quelques exemples

Considérons le modèle entité association suivant, citez moi au moins deux contraintes d'intégrités entre les attributs.



- Pour chaque occurrence d'*Emprunte*, si la date de retour existe, alors elle doit supérieure à la date d'emprunt
- Pour chaque occurrence de Livre, la date-achat doit être inférieure à la date d'emprunt de toutes les occurrences d'Emprunt qui lui sont liées

Les déclencheurs

Définition

Un *trigger*, *i.e;* un déclencheur est un mécanisme constitué d'une section de code accompagnée des conditions qui entraînent son exécution. Sa forme générale est la suivante :

```
before/after E
when C
for each row
begin
     A
end;
```

Il s'interprète de la façon suivante : *dès qu'un évènement **E** survient, si la condition **C** est satisfaite, alors exécuter l'action **A** soit avant soit après **E***

Les déclencheurs

Ces procédures sont en générales appelées **mécanismes E-C-A** (évènement - condition - action) et considère quatre types d'évènements : *insertion d'une ligne, suppression d'une ligne, modification d'une ligne* ou encore *modification d'une colonne d'une ligne*.

On peut également retrouver ces mécanismes sous une forme plus simple, plus proche des standards d'autres langages :

```
before/after E
begin
   if C then
      A
   end if
end;
```

Les déclencheurs

Exemple :

```
create trigger MAJ_CLI
before update of COMPTE on CLIENT
for each row
when (new.compte < -100)
begin
   if old.CAT = 'B2'
      then new.CAT = 'B1'
   end if;
end;
```

Les déclencheurs

Exemple :

Dans cet exemple, le déclencheur porte le nom de **MAJ CLI** et il agit sur le colonne **COMPTE** de la table **CLIENT**.

Pour chaque ligné modifiée et si la nouvelle valeur de **COMPTE** est inférieure à -100 alors, avant la modification, les valeurs de **CAT** sont rétrogradées (passage de B2 à B1).

Les suffixes *old* et *new* ainsi employés vont désigner l'état de la table avant et après modification.

Contraintes statiques

Définition

Une contrainte d'intégrité statique précise une propriété que les données doivent vérifier à tout instant. Les contraintes d'unicité, référentielles et de colonnes obligatoires en sont les exemples les plus fréquents. Ces derniers sont d'ailleurs gérés automatiquement par la plupart des systèmes de gestion de bases de données.

Exemple :

```
create trigger MAX-5-DET
before insert or update NCOM on DETAIL
for each row
begin
     if (select COUNT(*) from DETAIL where NCOM=new.NCOM) = 5
     then abort(); end if;
end;
```

Contraintes statiques

Dans cet exemple, on peut imaginer qu'on ne souhaite pas disposer de plus de 5 objets d'un même article en stock. Le déclencheur va alors vérifier que le nombre d'articles en stock pour chaque type d'objets est bien inférieur ou égal à 5 avant de mettre à jour la table.

Exercice : Ecrire un déclencheur qui changera la catégorie *CAT* du client en *old* si celui n'a pas émis de commande depuis 1er janvier 2021.

Contraintes dynamiques

Définition

Une contrainte dynamique indique quels changements d'états sont valides. On ne peut détecter une violation lors d'une modification qu'en connaissant les deux états avant et après l'évènement. Elle spécifie donc les changements d'états valides.

Exemple :

- On ne peut augmenter le prix de plus de 5%
- une personne peut passer du statut "célibataire" au statue "marié(e)" ou "pacsé(e)" mais pas aux statuts "divorcé(e)" ou "veuf(ve)"
- On ne peut ajouter les détails d'un produit qui se retrouvent en rupture de stock
- Un nouveau client d'un magasin ne peut se trouver que dans un nombre restreint d'état

Contraintes dynamiques : exemples

Dans l'exemple suivant, le déclencheur protège les produits contre toute modification du prix qui dépasserait 5%

```
create trigger MAJOR
before update of PRIX on CLIENT
for each row
begin
     if  new.PRIX > (old.PRIX * 1.05)
     then abort(); end if;
end;
```

Exercice : Ecrire un déclencheur qui autorisera la suppression d'un client que s'il n'a plus envoyé de commandes depuis le 1er Janvier 2021).

Les étapes pour établir un modèle entité association

Démarche à suivre :

- Identifier les types d'entités
- Identifier les attributs
- Associer les attributs aux différents types d'entités
- Identifier les associations entre les types d'entités
- Identifier les attributs de chaque association
- Evaluer la cardinalité des associations

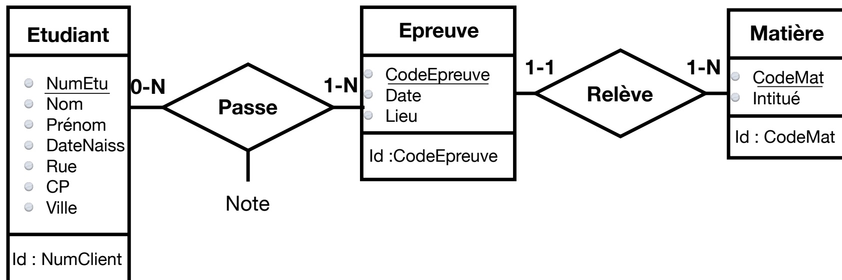
Il est important de bien lire l'énoncé, de bien analyser le contexte de votre étude afin d'identifier clairement les éléments importants qui pourront vous servir à la construction de votre modèle.

Exemple

Considérons une base de données relative à la notion d'Examen.
Construire le modèle entité association à partir de l'énoncé suivant

- les étudiants d'une université sont caractérisés par un numéro unique, leur nom, prénom, date de naisse, rue, code postal et ville
- ils passent des épreuves et obtiennent une note pour chacune
- les épreuves sont caractérisée par un code ainsi que la date et le lieu auxquels elles se déroulent
- chaque épreuve relève d'une matière unique. En revanche, une matière peut donner lieu à plusieurs épreuves
- les matières sont caractérisées par un code et un intitulé

Schéma Entité-Association



Exemple d'une structure administrative

Exemple : on considère un sous-ensemble d'une structure administrative. D'une direction (caractérisée par un nom identifiant et le nom de son président-directeur général) dépendent plusieurs départements (dotés chacun d'un nom identifiant dans sa direction et de sa localisation). Un département est découpé en services, dotés chacun d'un nom (identifiant de son département) et d'un responsable. Un service a la charge d'un certain nombre de dossiers identifiés par un numéro et dotés d'un titre et d'une date d'enregistrement/ Dans chaque service travaillent des employés identifiés par un numéro et caractérisés par leur nom et par leur adresse.

Ecrire le modèle entité-association lié à cet énoncé.

Schéma Entité-Association : Administration

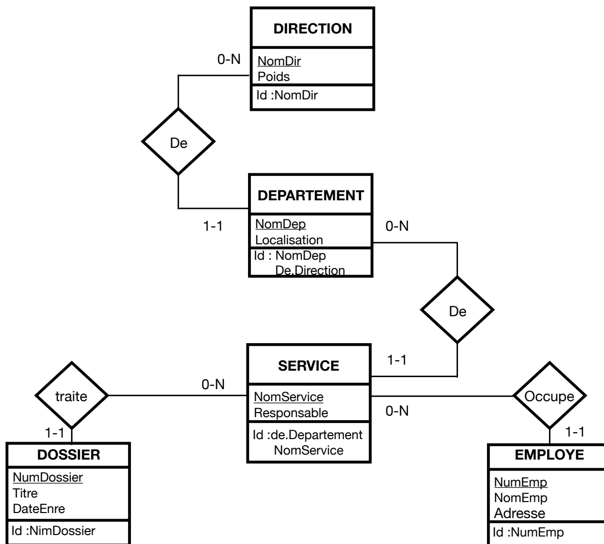


Schéma Entité-Association : Exemple

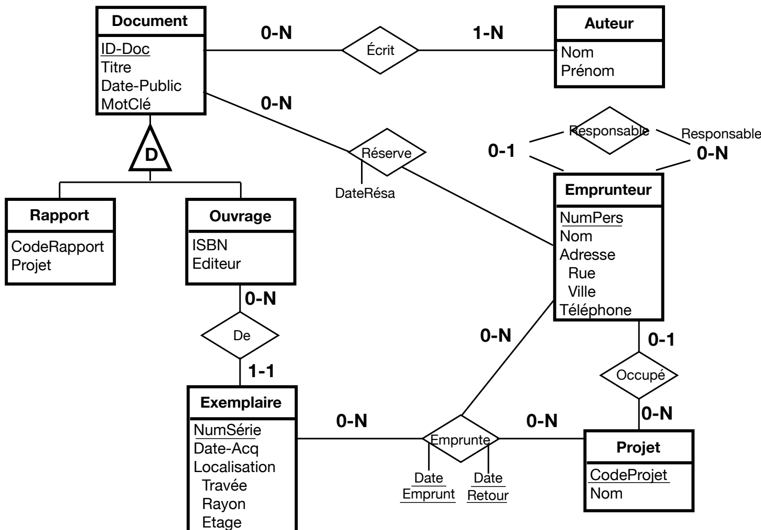
Une personne désire modéliser le système d'information correspondant aux réceptions qu'elle organise (personnes invitées, menus, ...). Ce système d'information doit lui permettre, en autre chose, de pouvoir l'aider à organiser une réception en lui offrant la possibilité de construire sa liste d'invités, ainsi que son menu et les vins associés. Une réception a lieu à une date donnée et y sont invitées des personnes dont on connait le nom, le prénom, leur sexe, leur âge et leur profession (l'identification d'une personne se fait par son nom et son prénom). Le repas servi lors d'une réception comprend un certain nombre de plats identifiés par leur nom (" poulet à la mexicaine " par exemple) et leur nature (" entrée froide ", " dessert " par exemple). Pour pouvoir réussir un menu, il faut que les vins servis soient en accord avec les plats. On dispose donc pour chaque plat d'une liste de types de vins possibles caractérisés par leur région viticole (" bourgogne " par exemple) et un type (" rouge corsé ", " blanc sec " par exemple). Pour que la réception soit réussie, il faut éviter qu'une dispute vienne gâcher l'événement et pour se faire la connaissance des amitiés et inimitiés entre personnes est primordiale. Enfin le dernier ingrédient d'une réception réussie est d'offrir au menu des plats que les invités apprécient et surtout d'éviter de leur servir des plats qu'ils n'aiment pas.

Extensions du modèle entité/associations

Jusqu'à présent, le formalisme présenté reste simple et permet de modéliser la plupart des problèmes qui peuvent se poser en pratique pour un non professionnel.

Nous allons maintenant regarder quelques extensions du modèle entité-association en présentant quelques extensions. Pour cela, on va se baser sur le schéma plus complexe suivant

Extensions du modèle entité/associations



Extensions du modèle entité/associations

- **Relations surtype/sous-type** ou **relation IS-A** : c'est lorsque le fait qu'une entité puisse appartenir à plus d'un type. Par exemples avec les rapports et les ouvrages qui forment deux classes spéciales de documents. *Tout rapport **IS A** document*.
  Les entités *rapport* et *documents* **héritent** des caractéristiques du type *document* (attributs + identifiants qui sont ceux de documents). Cette transmission, cet héritage est automatique.
- **Identifiants** : On remarque que certains types d'entités n'ont pas d'identifiants comme cela est le cas pour le type d'entité auteur.

Extensions du modèle entité/associations

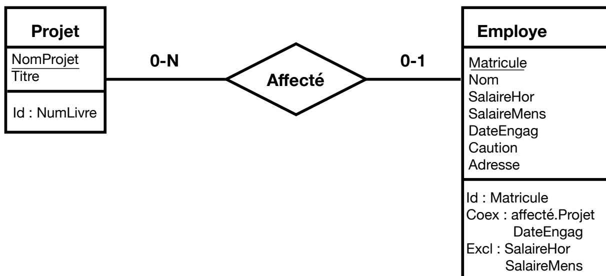
- **Associations N-aires** : ce sont des associations déjà rencontrées plus tôt et qui mettent en relations plusieurs types d'entitées.
  Ex : relation ternaire entre EMPRUNTEUR - PROJET - EXEMPLAIRE
- **Attributs de types d'associations** : nous avons également rencontré cela plus tôt. Il est courant d 'associer des attributs à des associations afin d'en spécifier les caractéristiques.
  Ex : préciser la date d'emprunt d'un exemplaire (entre EXEMPLAIRE et EMPRUNTEUR)

Extensions du modèle entité/associations

- **Attributs composés** : cela arrive lorsque notre attribut est *complexe*, ce qui est souvent le cas avec une adresse par exemple. Il est alors d'usage de le décomposer en attributs *simples/élémentaires*.
  Ex : voir les types d'identités EXEMPLAIRE et EMPRUNTEUR.
- **Attributs multivalués** : En principe chaque entité possède une valeur de chacun de ses attributs. Si un attribut et multivalué (ex : liste des vaccins effectués), alors une entité peut posséder plusieurs valeurs de cet attribut.
  Ex : Voir les attributs Mot Clé de DOCUMENT et Téléphone de EMPRUNTEUR.
  Le nombre de valeurs par entité est caractérisé par une contre de cardinalité de type *min-max*.

Extensions du modèle entité/associations

- **Contraintes d'intégrités** : voir exemple ci-dessous.



Si un employé est affecté à un projet, alors il possède une date d'engagement et inversement (Coex)
Un employé a un salaire horaire ou un salaire mensuel, mais pas les deux (Excl)

Conclusions et ... suite !

Il existe bien d'autres formalisme conceptuel, nous avons vu ensemble le *modèle entité/associations*. Nous aurions également pu aborder le modèle *Mérise* qui lui est antérieur ou encore le modèle *UML : Universal Modelling Laguage* mais ce n'est pas le but de ce cours.

On va simplement en donner l'idée mais tout cela sera à nouveau étudié l'année prochaine, plus précisément pour ceux s'orientant vers un master sécurité (OPSI)!

UML (Unified Modeling Language)

Il ne s'agit pas d'une méthode, *i.e.* cela ne fournit pas les étapes ou les normes à suivre qui serviront à modéliser le problème. Il s'agit surtout d'un **lanage graphique** tout le modèle **entité/relation** ou **entité/association**.

Méthode qui a commencé à être développé dans les années 90 (en 1995 par *Booch* et *Rumbaugh*) et qui s'impose surtout à la fin des années 2000 comme **l'outil** de modélisation des systèmes d'informations à différentes échelles (système d'informations, interactions, acteurs, transitions, ...).

UML : un langage orienté objet

UML est une langage graphique reposant sur l' **approche orientée objet (AOO)** et fera donc intervenir des notions qui sont propres à cette approche. Comme les notions de :

- **classe :** : type de données abstrait qui précise des caractéristiques (attributs) communes à une famille d'objets qui permet de créer (on dit aussi *instancier*) des objets possédants des caractéristiques.
- **encapsulation :** elle permet notamment de garantir l'intégrité des données en interdisant ou restreignant l'accès direct aux attributs des objets.
- **héritage,spécialisation,généralisation** : le premier est un mécanisme de transmission des caractéristiques d'une classe vers une sous-classe. Spécialisation et Généralisation permettent alors de construire des hiérarchies entre les classes.

UML : un langage graphique

UML est avant tout un langage graphique qui, dans sa version 2.0 comporte plus de 13 diagrammes différents utilisés pour la modélisation. On les divise en deux grandes catégories les diagrammes **statiques**

- **diagramme de classe**
- diagrammes d'objets
- diagramme de composants
- diagramme de déploiement
- diagramme de paquetage
- diagramme de structure composite

et ...

UML : un langage graphique

.. les diagrammes **dynamique ou comportementaux**

- **diagramme de cas d'utilisation**
- diagrammes d'activités
- diagramme d'états-transitions
- diagramme d'interaction
- diagramme de séquence
- diagramme de communication
- diagramme de temps

Tous ne sont pas nécessaires à la modélisation. On va se concentrer sur deux des diagrammes les plus utilisés en pratique : le **diagramme d'utilisation** et le **diagramme de classe**

Autres diagrammes importants

- **diagramme objets** –> principalement une fonction illustrative permettant de vérifier la cohérence d'un diagramme de classes à différents cas possibles.
- **diagramme états-transitions** –> représente le cycle de vie des objets appartenant à une même classe (prise en compte de l'aspect temporel). Cela permet de mettre en forme la dynamique du système.
- **diagramme d'activités** –> transcription des étapes de modélisations, il s'agit de représenter le processus des étapes de la modélisation
- **diagramme séquence/communication** –> représente la succession chronologique des opérations réalisées par un acteur (quels sont les objets que l'acteur va manipuler et comment il va passer d'une opération à une autre).

Diagramme de cas d'utilisation

A qui est-ce destiné ?

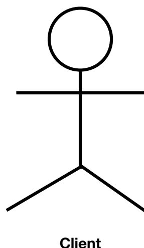
Cela sert surtout à interagir avec des non informaticiens. Il permet notamment aux non expert de pouvoir représenter le problème rencontré afin qu'un expert puisse les aider dans la modélisation de la base de données.

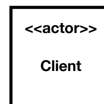
Dans quel but ?

Ce diagramme va servir à modéliser ou présenter le comportement d'un système, d'une sous-partie du système ou encore des classes tel que cela peut être vu par un utilisateur extérieur
–> expression des besoins de l'utilisateur
–> étape importante pour élaborer un système qui est conforme aux attente de l'utilisateur.

élément du diagramme : Acteur

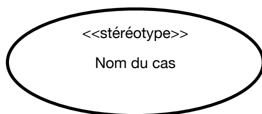
Acteur : il joue le rôle d'une personne externe qui va interagir avec le système. On peut le représenter à l'aide d'un schéma "enfantin", très graphique ou sous forme d'un *classeur stéréotypé* «actor».

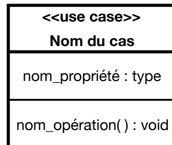




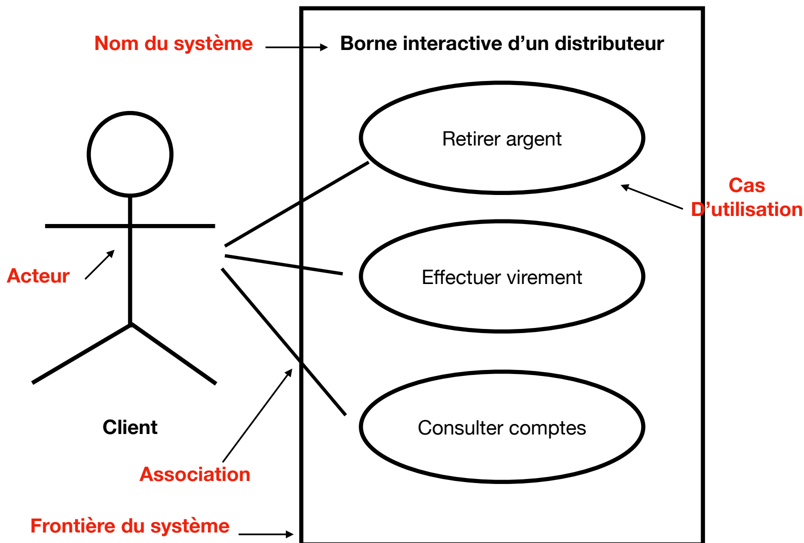
élément du diagramme : cas d'utilisation

Cas d'utilisation : unité cohérente représentant une fonctionnalité visible de l'extérieur et qui réalise un service de bout en bout (initialisation jusqu'à une fin) –> service rendu par le système.





exemple d'un diagramme *use case*

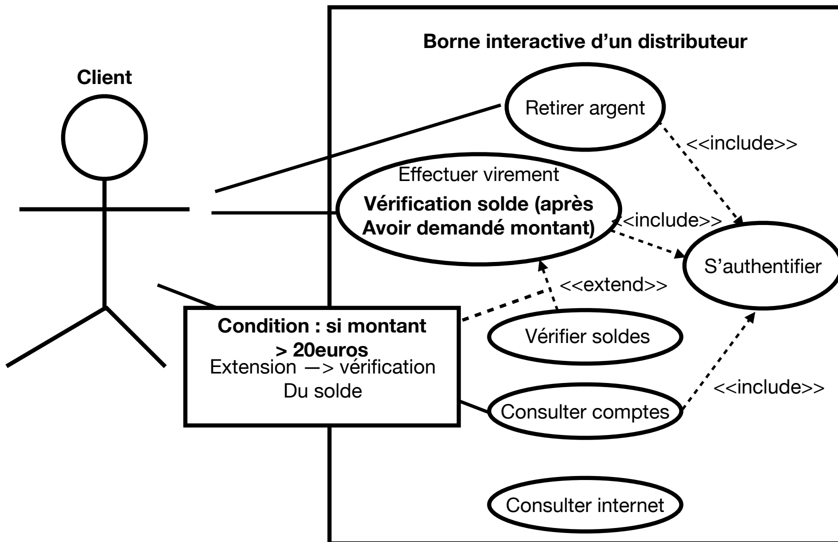


Les relations

On peut, tout comme pour la modélisation **E-A**, définir la relation d'association par le biais de :

- **la multiplicité :** on retrouve les mêmes concept de multiplicité que pour le schéma entité-association
- **rôle des acteurs :** un acteur primaire, lorsque l'action entreprise rend service à l'acteur en question. Les autres acteurs sont alors considérés comme des acteurs secondaires (ex. du distributeur : client vs agent)

Relation entre cas d'utilisation



Relation entre cas d'utilisation

Il existe deux types de relations :

- les dépendances stéréotypées, qui sont explicitées par un stéréotype (comme l'inclusion et l'extension)
- et la généralisation/spécification

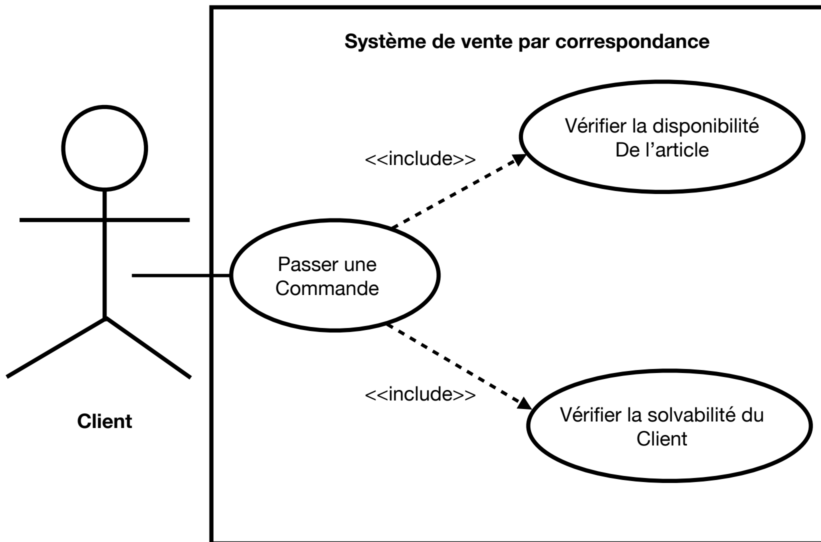
Une dépendance se représente par une flèche. Si le cas A inclut ou étend le cas B, la flèche est dirigé de A vers B

Le symbole utilisé pour la **généralisation** est une flèche avec un trait plein dont la pointe est triangulaire désignant le cas le plus général.

Relation d'inclusion

Un cas A inclut un cas B si le comportement décrit par le cas A inclut le comportement du cas B : on dit aussi que le cas A dépend de B. Lorsque A est sollicité, B l'est obligatoirement en tant que partie de A. Cette dépendance est symbolisée par le stéréotype « include ».
Par exemple, l'accès aux informations d'un compte bancaire inclut nécessairement une phase d'authentification avec un identifiant et un mot de passe.

Relation d'inclusion



Le diagramme de classe

C'est le diagramme le plus fondamental dans la modélisation orientée objet, il est même indispensable à toute modélisation.

Contrairement au diagramme du cas d'utilisation qui se focalise sur un point de vue interne, le diagramme de classe va surtout se concentrer sur le point de **vue interne** : il va chercher à modéliser les relations entre les différentes classes et leurs relations.
Toute la description qui y est faite est **statique** (comme pour les schémas E-A !).

Classes et instances de classe

Une **instance** est une concrétisation d'un concept abstrait. Par exemple :

- la Porshe qui se trouve dans votre garage est une instance du concept abstrait Automobile ;
- l'amitié qui lie Albert et Bernard est une instance du concept abstrait Amitié

Une **classe** est un concept abstrait représentant des éléments variés comme :

- des éléments concrets (ex. : des avions),
- des éléments abstraits (ex. : des commandes de marchandises ou services),
- des composants d'une application (ex. : les boutons des boîtes de dialogue),
- ou encore des structures ou des éléments comportementaux, ...

Quelques généralités

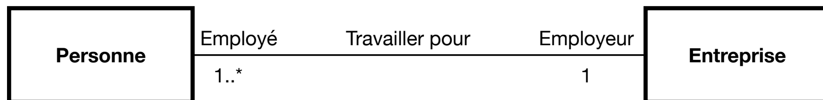
On retrouve exactement les mêmes composantes que dans un schéma entité association :

- classe $\leftrightarrow$ entité
- objet $\leftrightarrow$ un exemple ou une ligne dans une base de donnée (on parle d'instanciation)

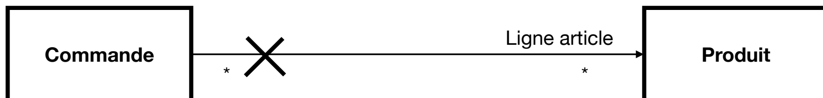
On retrouve également des éléments communs comme la description d'un lien entre deux classes (ou plusieurs classes ! On se rappelle des associations n-aires) et on peut également retrouver la notion de cardinalité avec une petite subtilité malgré tout ...

Quelques généralités

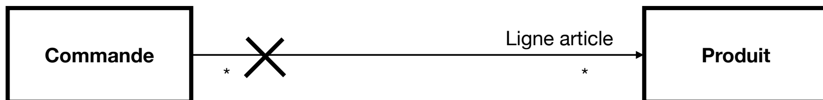
Un petit exemple de relation binaire



Il faut noter que, pour les habitués du modèle entité/relation, les multiplicités sont, en UML, « à l'envers ». En revanche il est possible de spécifier le rôle de chaque classe vis à vis de la ou les classe(s) en relation. Enfin, on retrouver également le rôle de l'association.
On peut aussi définir la **navigabilité**



Quelques généralités



La navigabilité indique s'il est possible de traverser une association. On représente graphiquement la navigabilité par une flèche du côté de la terminaison navigable et on empêche la navigabilité par une croix du côté de la terminaison non navigable.

La terminaison du côté de la classe Commande n'est pas navigable : cela signifie que les instances de la classe Produit ne stockent pas de liste d'objets du type Commande. Inversement, la terminaison du côté de la classe Produit est navigable : chaque objet commande contient une liste de produits.

Pour finir

Il y aurait plein d'autres choses à aborder, comme les notions de **paquetages** de **qualificationn** de **dépendance**, etc.

De même, l'approche orientée objet peut également faire référence à une structure de *code* qu'il serait bon d'adopter ou encore à des nomenclatures spécifiques, ...

Mais vous verrez cela l'année prochaine ! Retour aux principes de base !

Modèle Relationnel

Objectifs des SGBD relationnels

- **Independance physique :** un remaniement de l'organisation physique des données n'entraîne pas de modification dans les programmes d'applications
- **Indépendance logique :** un remaniement de l'organisation logique des fichiers n'entraîne pas de modifications dans les programmes d'applications concernés
- **Manipulation facile des données :** un utilisateur néophyte ou profane (*i.e.* non informaticien) doit pouvoir manipuler simplement les données (interrogation et éventuellement mise à jour de la base de données)
- **Administration facile des données :** un SGBD doit fournir des outils pour décrire les données, permettre le suivi de ces structures et autoriser leur évolution (tâche concernant plutôt l'administrateur des bases de données).

Objectifs des SGBD relationnels

- **Efficacité des accès aux données :** garantie d'un bon débit (nombre de transactions tâches effectuées par seconde) et d'un bon temps de réponse (temps moyen d'attente pour effectuer une tâche)
- **Redondance contrôlée des données :** diminution du volume de stockage, pas de mise à jour multiple, ni d'incohérence
- **Cohérence des données :** l'âge d'une personne doit être un entier strictement positif. le SGBD doit vérifier que les applications respectent certaines règles (contraintes d'intégrités)
- **Partage des données :** utilisation simultanée des données par différentes applications
- **Sécurité des données :** les données doivent être protégées contre les accès non autorisés ou en cas de panne

Schéma relationnel

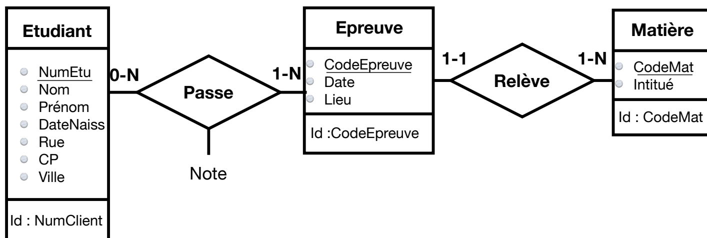
Il est important de garder le formalisme suivant en tête :

- une base de données est constitué d'un ensemble de **relations**
- le schéma d'une base de données relationnelle est donc un **ensemble de schémas de relation que l'on notera** R_i
- le schéma d'une relation est constitué **du nom de la relation** et **d'un ensemble d'attributs** formant un $n - uplet$ (ici $n = r_i$)

$$R_i = (A_1, A_2, \ldots, A_{r_i})$$

Ces attributs doivent être **simples** et **monovalués** et la **structure doit être plate**.

Schéma relationnel : exemple



- Le relation EPREUVE est l'ensemble des attributs { Code Epreuve, Date, Lieu}
- Chaque attribut A_i prend ses valeurs dans un certain domaine

  Note $\in [0, 20]$
  Lieu $\in$ {'Amphi 136', 'K073', 'Salle 201', 'Salle 01', ... }

Quelques règles

Des contraintes semblables à celle exposées lors de la présentation du modèle E/A

- **Toute relation possède un identifiant (clé primaire)** : ensemble d'attributs dont les permettent de distinguer les n-uplets les uns des autres. Il ne peut en aucun admettre de valeurs nulles
  Ex : CodeEpreuve est une primaire de la relation EPREUVE.
- **Identifiant externe (clé étrangère)** : il s'agit d'un attribut qui est clé primaire d'une autre relation.

Quelques règles

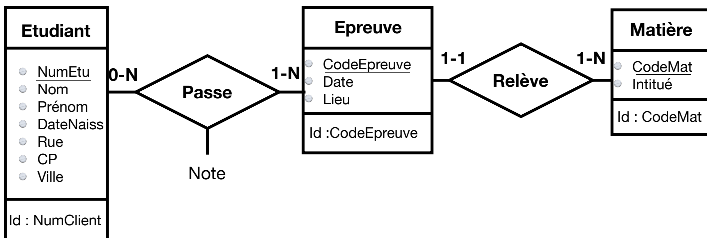
Une relation est définie par :

- son nom
- son $n - uplets$, *i.e.* sa liste d'attributs qui doit respecter des contraintes de domaines (on ne les mentionne pas explicitement dans le schéma)
- son ou ses identifiants (clé primaire, étrangère). Il est d'usage de souligner la clé primaire et de faire précéder d'un # la ou les clé(s) étrangère(s)

Ex : EPREUVE (<u>CodeEpreuve</u>, Date, Lieu, #CodeMat)

Passage au modèle relationnel

Repartons de notre modèle E/A concernant les examens que passent des étudiants et regardons ensemble comme passer de ce modèle à un modèle relationnel



Passage au modèle relationnel : règle 1

Chaque **entité** devient une **relation**.
Les **attributs** de l'entité deviennent **attributs** de la relation.
Enfin, **l'identifiant** de l'entité devient **clé primaire** de la relation

Exemple :

ETUDIANT (NumEtu, Nom, Prénom, DateNaiss, Rue, CP, Ville)

Passage au modèle relationnel : règle 2

Chaque **association avec cardinalités maximales 1-1** est prise en compte en incluant la **clé primaire** d'une des relations comme clé étrangère dans l'autre relation.

Exemple : Si un étudiant peut posséder une seul et unique carte étudiant, on aura :

CARTE(NumCarte, Crédit, ...)
ETUDIANT(<u>NumEtu</u>, Nom, Prénom, DatNaiss, Rue, CP, Ville, #NumCarte)

Passage au modèle relationnel : règle 3

Chaque **association avec cardinalités maximales 1-N** est prise en compte en incluant la **clé primaire** de la relation dont la cardinalité maximale est **N** comme **clé étrangère** dans l'autre relation.

Exemple : Entre les relations EPREUVE et MATIERE, on aura :

EPREUVE (CodeEpreuve, Date, Lieu, #CodeMat)
MATIERE (CodeMat, Intitulé)

Passage au modèle relationnel : règle 4

Dans le cas d'associations de cardinalités maximales **plusieurs - plusieurs,** ***i.e.*** **N-N**, cette dernière est prise en compte dans le schéma relationnel de la façon suivante

- **création d'une nouvelle relation** dont la **clé primaire** est la concaténation des clés primaires des relations participantes
- les attributs de l'association sont insérés dans cette nouvelle relation si nécessaire

Exemple : L'association PASSE devient alors

PASSE(#<u>NumEtu</u>,# <u>CodeEpreuve</u>, Note)

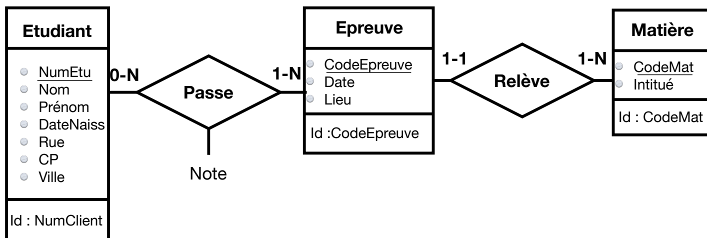
Passage au modèle relationnel : règle 5

Dans le cas d'associations de types **cycliques**, il suffit d'appliquer les règles précédentes pour établir le schéma relationnel.

Il faudra donc créer une nouvelle relation à partir de l'association quand cela est nécessaire et bien mentionner les identifiants primaires de l'entité en association cyclique dans les attributs et en clé étrangère de l'association transformée en relation.

Passage au modèle relationnel : exemple

Repartons de notre modèle E/A concernant les examens que passent des étudiants et regardons comment passer de ce modèle à un modèle relationnel



Passage au modèle relationnel : exemple

Le modèle relationnel associé serait le suivant :

ETUDIANT (NumEtu, Nom, Prénom, DateNaiss, Rue, CP, Ville)

PASSE(#NumEtu,# CodeEpreuve, Note)

EPREUVE(CodeEpreuve, Date, Lieu, # CodeMat)

MATIERE(CodeMat, Intitulé)

Qu'est-ce qu'une Base de données correcte ?

Il s'agit d'un ensemble de relations tel que :

- Chaque relation décrit un fait élémentaire avec les seuls attributs qui lui sont directement liés
- Il n'y a pas de redondance d'information, génératrices de problèmes lors des mises à jour
- Il n'y a pas de perte d'information
  PERSONNE(Nom, Prénom)
  ADRESSE(No, Rue, Ville)
  Qui habite ou ? Impossible de répondre.

Normalisation

Qu'est-ce que la normalisation ?

- C'est un processus de transformation d'une relation qui pose des problèmes lors des mises à jour en des relations ne posant pas de problèmes
- On mesure la qualité d'une relation par son degré de normalisation
- Diverses formes normales

    Première forme normale
    Deuxième forme normale
    Troisième forme normale
    FNBC
    Quatrième forme normale

Qu'est-ce qu'une Base de données incorrecte ?

Une relation n'est pas correcte si :

- elle implique des répétitions au niveau de sa population
- elle pose des problèmes lors des mises à jour (insertions, modifications, suppressions)
- les conditions pour qu'une relation soit correcte peuvent être définies formellement :

$\rightarrow$ **Règles de normalisation**

Vérification du modèle relationnel

Première forme normale

- Une relation est en première forme normale si tous les attributs ne sont pas décomposables (on dit aussi que tous les attributs sont **atomiques**, donc pas **multi-valués**

- **Exemple :**

  La relation DEPARTEMENT(Nom, Adresse, Tel) n'est pas en première forme normale si les attributs Nom et Adresse peuvent être du type [*Jean Paul*] ou [*Rue de Marseille, 69003 Lyon*].

Vérification du modèle relationnel

Deuxième forme normale

- Une relation est en deuxième forme normale si elle est en première forme normale et si tout attribut non clé primaire dépend entièrement de la clé primaire (à vérifier dans le cas ou l'identifiant est composé)

Exemples :

La relation CLIENT(NumCli, Nom, Prenom, DateNaiss, Rue, Cp, Ville) est en deuxième forme normale.
La relation COMMANDE(NumProd, NumFour, Quantite, VilleFour) n'est pas en deuxième forme normale car seul $NumFour \rightarrow Ville$

La décomposition suivante donne deux relations en deuxième forme normale :
COMMANDE (NumProd, NumFour, Quantité) FOURNISSEUR(NumFour, Ville Four)

Vérification du modèle relationnel

Troisième forme normale

- Une relation est en troisième forme normale si elle est en deuxième forme normale et s'il n'existe aucune dépendance fonctionnelle entre deux attributs non clé primaire.

Exemples :

La relation MUSIQUE(<u>NoChanson</u>, # NoChanteur, NomChanteur) n'est pas en troisième forme normale
En effet, NoChanteur $\rightarrow$ Nom La décomposition suivante donne deux

relations en troisième forme normale :
R1(<u>NoChanson</u>, # NoChanteur) R2(<u>NoChanteur</u>, Nom)

Dépendances fonctionnelles

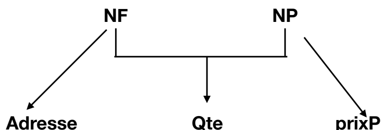
- Soit R(X,Y,Z) une relation où X,Y, Z sont des ensembles d'attributs. Z peut être vide.
- **Définition :** X est en dépendance fonctionnelle avec Y (X$\rightarrow$ Y) si pour une valeur de X on détermine une seule de valeur de Y dans la relation R.
- **Exemple :** PRODUIT(NumProd, Dési, Prix)

  Deux dépendances fonctionnelles possibles :
  NumProd $\rightarrow$ Dési NumProd $\rightarrow$ Prix

Graphe des dépendances fonctionnelles

Il est intéressant, pour chaque relation, de connaitre les dépendances fonctionnelles. On peut alors les représenter sous forme de graphes. On considère la relation LIVRAISON(NF,Adresse, NP, PrixP, Qte)

- NF —> Adresse : l'adresse du fournisseur ne dépend que du fournisseur
- NP —> PrixP : le prix d'un produit ne dépend que du produit
- (NF, NP) —> Qte : la quantité livrée dépend du produit et du fournisseur
- [faux : NF —> Qté, NP —> Qte]



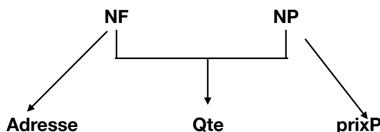
Dépendance fonctionnelle et identifiants

- Le graphe minimum des dépendances fonctionnelles permet de trouver les identifiants de la table
- L'identifiant d'une table est l'ensemble (minimal) des nœuds du graphe minimum à partir desquels on peut atteindre tous les autres nœuds (via les dépendances fonctionnelles)
- Pour que cela soit faux, il faudrait qu'il y ait deux lignes avec la même valeur de l'identifiant et des valeurs différentes pour les autres attributs, ce qui est en contradiction avec les dépendances fonctionnelles.

Ainsi, dans l'exemple précédent, on remarque la relation n'est pas normalisée, deux identifiants : NF et NP. Il va donc falloir la décomposer, *i.e.* la remplacer par un ensemble de relations.

Méthode pragmatique

Repartons de notre relation LIVRAISON(NF, Adresse, NP, prixP, Qte) et du graphe des dépendances fonctionnelles associé.



On a deux identifiants : *NF* et *NP* avec les dépendances décrites dans le graphe ci-dessus.
Une bonne décomposition se ferait alors de la façon suivante :

- NP —> prixP se traduit par PRODUIT(NP, prixP)
- NF —> Adresse se traduit par FOURNISSEUR(NF, Adresse)
- NP,NF —> Qte se traduit par LIVRAISON(NF, NP, Qte)

Méthode formelle de décomposition

Théorème de Heath

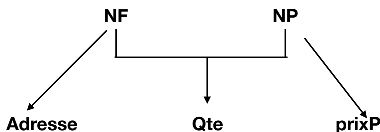
T(X,Y,Z) est décomposable sans perte d'information en

T1(X,Y) et T2(X,Z)

si X est en dépendance fonctionnel avec Y.

Application de Heath

Reprenons notre relation LIVRAISON(NF, NP, Adresse, prixP, Qte)



- NF —> Adresse, on peut décomposer LIVRAISON en
  L1 (NF, Adresse)
  L2 (NF, NP, prixP, Qte)
- NP —> prixP, on peut décomposer L2 en
  L3 (NP, prixP) L4 (NP, NF, Qte)
- et on a bien la dépendance fonctionnelle (NP, NF) —> Qte.

Langage SQL

Avant propos

- SQL **Structured Query Language** ce qui signifie *Langage de requête structurée* en français
- SQL permet la définition, la manipulation et le contrôle d'une base de données relationnelle. Il se base sur l'algèbre relationnelle (mais on ne parlera pas d'algèbre relationnel dans ce cours).
- SQL est un standard depuis la fin des années 80 (1986) mais qui fut développé en 1974 chez **IBM** par Astrahan & Chamberlin

Avant propos

Le langage SQL se décompose essentiellement en deux parties :

- **Le langage DDL (Data Description Language) :** est la partie qui est consacrée à *la définition ou la modification de structure*. C'est donc la partie qui va nous permettre de définir des tables, de la modifier, de définir la nature des colonnes de notre table et éventuellement les contraintes qui y sont associées.
- **Le langage DML (Data Manipulation Language) :** est la partie qui consacrée à la *manipulation des données* (extraction et modification). C'est très certainement la partie qui est susceptible d'intéresser toute personne amenée à travailler avec des données.

Le langage DDL

Cette partie est à la base d'un SGBD et va permettre de mettre en place **toute la structure de la base de données** :

- définir une table
- définir la nature des éléments de cette table, *i.e.* le type des attributs
- modifier une table existante en ajoutant, supprimant des colonnes
- supprimer une table
- prendre en compte les différentes contraintes d'intégrités
- ajouter des structures physiques
- ...

Création d'une table

La création d'une table se fait à l'aide de CREATE TABLE. La commande générique est la suivante

```
CREATE TABLE nom-table(
   id0 type-données PRIMARY KEY,
   colonne1 type-données DEFAULT val,
   colonne2 type-données
   colonne3 type-données,
   ...
)
```

On peut aussi créer une table à partir d'une table existante

```
CREATE TABLE Table2 AS (
SELECT Attr1, Attr2,  ...
FROM Table1
)
```

Nature des attributs ou type des données

Des types **numériques**

- **NUMERIC**
- **DECIMAL** ou DECIMAL (M,D) M chiffres au total
- **INTEGER** : tinyint (1 octet) -smallint (2 octets) - mediumint (3 octets) - int (4 octets) - bigint (8 octets)
- **FLOAT** 4 octets par défaut
- **REAL** 8 octets par défauts
- **DOUBLE PRECISION** 8 octets

Nature des attributs ou type des données

Des types **Date et Heure**

- **DATETIME** : AAAA-MM-JJ HH:MM:SS
  de 1000-01-01 00:00:00 à 9999-12-31 23:59:59
- **DATE** : AAAA-MM-JJ
  de 1000-01-01 à 9999-12-31
- **TIMESTAMP** : Date sans séparateur AAAAMMJJHHMMSS
- **TIME** : HH:MM:SS
- **YEAR** : YYYY
  de 1901 à 2155

Nature des attributs ou type des données

Des types **Chaînes** comme avec CHAR(n) et VARCHAR(n) où $1 \leq n \leq 255$

- **TINYTEXT** taille $\leq 2^8$ caractères
- **TEXT** taille $\leq 2^{16}$ caractères
- **MEDIUMTEXT** taille $\leq 2^{24}$ caractères
- **LONGTEXT** taille $\leq 2^{32}$ caractères

Contraintes sur les colonns et tables

Sur les **colonnes** ou **attributs**

- NOT NULL
- UNIQUE
- PRIMARY KEY
- REFERENCES nom-table
- CHECK (condition)

Sur une **table**

- UNIQUE(nom-col)
- PRIMARY KEY (nom-col)
- FOREIGN KEY (nom-col) REFERENCES nom-table nom-col
- CHECK (condition)

Quelques exemples de création de tables

- **Exemple 1 :** créer une table *client* avec les attributs *NumCli*, *Nom*, *DateNaiss*, *telephone*, *solvable* en attribuant les types adéquats à chacun d'eux et la clef primaire au bon attribut.

Quelques exemples de création de tables

- **Exemple 1 :** créer une table *client* avec les attributs *NumCli*, *Nom*, *DateNaiss*, *telephone*, *solvable* en attribuant les types adéquats à chacun d'eux et la clef primaire au bon attribut.

```
CREATE TABLE CLIENT (
    NumCli NUMBER(8),
    Nom VARCHAR(30),
    DateNaiss DATE,
    Telephone CHAR(10),
    Solvable BOOLEAN,

    PRIMARY KEY (NumCli)
)
```

Quelques exemples de création de tables

- **Exemple 2 :**

On considère une première table *service* avec les attributs *NumService* et *Nom Service*.
On considère également la table *employé* avec les attributs *mat, nomEmploye, fonction, dateEmbauche, salaire, commission, numService*

Ecrire les requêtes permettant de définir chacune de ces tables en prenant soit d'indiquer le type de chaque attribut, les clefs primaires de chaque table ainsi que les références étrangères s'il y a lieu.

Quelques exemples de création de tables

- **Exemple 2 :**

On considère une première table *service* avec les attributs *NumService* et *Nom Service*.
On considère également la table *employé* avec les attributs *mat, nomEmploye, fonction, dateEmbauche, salaire, commission, numService*

Ecrire les requêtes permettant de définir chacune de ces tables en prenant soit d'indiquer le type de chaque attribut, les clefs primaires de chaque table ainsi que les références étrangères s'il y a lieu.

```
CREATE TABLE SERVICE (
    NumService INTEGER,
    NomService VARCHAR(30),

    PRIMARY KEY(NumService),
)
```

Quelques exemples de création de tables

- **Exemple 2 (suite) :**

```
CREATE TABLE EMPLOYE (
    mat INTGER(3),
    nomEmploye VARCHAR(15),
    fonction VARCHAR(15),
    dateEmbauche DATE,
    salaire DECIMAL(7,2),
    comission DECIMAL(7,2),
    numService INTEGER,

    PRIMARY KEY(mat),
    FOREIGN KEY(NomService) REFERENCES Service(numService)
)
```

Modification structure existante

- **Ajout d'attributs**

```
ALTER TABLE nom_table ADD attribut type (taille), ...
```

- **Exemple 1**

```
ALTER TABLE Client ADD telephone CHAR(8);
```

- **Exemple 2**

```
ALTER TABLE Produit ADD CodeCategorie INTEGER;
ALTER TABLE Produit ADD FOREIGN KEY (CODECategorie)
REFERENCES Categorie (CodeCatégorie);
```

Modification structure existante

- **Modification d'attributs**

```
ALTER TABLE nom_table MODIFY attribut type (taille),...
```

- **Exemple 1**

```
ALTER TABLE client MODIFY telephone CHAR(10);
```

- **Exemple 2**

```
ALTER TABLE Employe MODIFY Salaire MONEY;
```

Modification structure existante

- **Suppression d'attributs**

```
ALTER TABLE nom_table DROP attribut
```

- **Exemple 1**

```
ALTER TABLE Client DROP DateNaissance;
```

- **Exemple 2**

```
ALTER TABLE Employe DROP NumService;
```

Attention ! Cette suppression est possible uniquement si la relation entre la table Employé et la table Service est cassée. Dans le cas contraire, la SGBD vous empêchera de faire cette manipulation.

Modification structure existante

- **Suppression d'une table**

```
DROP TABLE nom_table
```

- **Exemple 1**

```
DROP TABLE Services
```

Attention ! Cette suppression est possible uniquement si la table *Services* n'est pas reliée à une autre table de la base de données.

Actions déclenchées

Nous avons vu que des actions menées sur certaines tables peuvent avoir des répercussions sur des autres tables.
Par exemple, que se passerait-il quand on détruit/met à jour une clé primaire ou un attribut de type unique qui est référencé par un attribut FOREIGN KEY d'une autre table ?

Exemple :

```
CREATE TABLE Departement (
   numero_departement INT(4) PRIMARY KEY ,
   numero_manager INT(4) REFERENCES
   EMPLOYE(numero_employe)
```

Quid en cas de mise à jour ?

Actions déclenchées

Deux circonstances

- ON DELETE
- ON UPDATE

Trois possibilités d'actions

- SET NULL
- SET DEFAULT : valeur par défaut si existe, sinon NULL
- CASCADE : on répercute les m.a.j.

Exemple

```
CREATE TABLE Departement (
   numero_departement INT(4) PRIMARY KEY,
   numero_manager INT(4) REFERENCES Employe(numero_emp)
            ON DELETE SET NULL
            ON UPDATE CASCADE
)
```

Retour sur les contraintes –> CHECK

- Permet de définir portant sur chaque ligne d'une table
- Mais également de mettre des contraintes sur les colonnes d'une table ou sur plusieurs colonnes d'une table.

Exemple :

```
CREATE TABLE divisions(
   num_div INT CHECK (num_div between 10 AND 99),
   nom_div VARCHAR(9) CHECK (nom_div = UPPER(nom_div),
   bureau VARCHAR(10) CHECK
   (bureau IN ('Lyon', 'Paris', 'Lille' ))
)
```

Retour sur les contraintes –> CHECK

- Un autre exemple de contraintes portant sur plusieurs colonnes

Exemple :

```
CREATE TABLE Employe(
   numero_employe INT(4) PRIMARY KEY,
   nom_employe VARCHAR(10),
   nom_job VARCHAR(9),
   numero_manager INT(4),
   salaire DECIMAL(7,2),
   commission DECIMAL(7,2),
   numero_departement SMALLINT(2),
   CONSTRAINT salaire_com CHECK (salaire+commission <=5000)
)
```

Statut des contraintes

Comment et quand vérifier une contrainte ? On peut faire ce qui suit pour chaque contrainte

- **DEFFERABLE/NOT DEFFERABLE** : contrainte activée immédiatement ou non
- **ENABLE/DISABLE** : active ou désactive la contrainte
- **VALIDATE/NOVALIDATE** : on valide ou pas les données déjà saisies

ENABLE/DISABLE

- activation/désactivation d'une contrainte dont on a donné le nom dans un CREATE TABLE ou autre.
- si la contrainte n'est pas vérifiée sur certaines valeurs, elle ne peut être activée
- on ne peut pas activer une contrainte de clé étrangère qui référence un attribut auquel est associé une contrainte non active

Statut des contraintes

ENABLE

- **VALIDATE :** la contrainte est activée et contrôle que les données de la table vérifient la contrainte (c'est l(option par défaut)
- **NOVALIDATE :** la contrainte est activée, seules les nouvelles données entrées dans la base de données devront vérifier cette contrainte
-

DISABLE

- **VALIDATE :** on cherche à désactier la contrainte, si les données ne sont pas valides, erreur.
  Après **DISABLE**, on ne peut plus entrer, modifier ou supprimer des données de la table
- on peut faire n'importe quelle opération, y compris entrer des données non conformes à la contrainte

Retour sur les manipulations de tables

```
DROP TABLE nom_table CASCADE CONSTRAINTS
```

CASCADE CONSTRAINTS

- Supprime toutes les contraintes référençant une clé primaire (PRIMARY KEY) ou une clé unique (UNIQUE) de cette table
- Si on cherche à détruire une table dont certaines attributs sont référencés sans spécifier CASCADE CONSTRAINT, on a un message d'erreur.

Retour sur les manipulations de tables

Une structure plus complète

```
ALTER TABLE nom_table{
   ADD contrainte_table
   | DROP {
                  PRIMARY KEY
                  | UNIQUE (nom_colonne)
                  | CONSTRAINT nom_contrainte  }
                  CASCADE CONSTRAINTS
   | RENAME CONSTRAINT ancien TO nouveau
   | MODIFY CONSTRAINT nom_contrainte
                       statut_contrainte
}
```

Ajouter une contrainte

```
... ADD contrainte_table
```

Quelques exemples :

- on veut ajouter une contrainte à la table Employe

```
ALTER TABLE Employe
   ADD CONSTRAINT seuil_commission
   CHECK ((commission/salaire)<=1)
```

- on veut définir une clé primaire dans la table Pays (non définie plus tôt)

```
ALTER TABLE Pays
   ADD CONSTRAINT cle_pays PRIMARY KEY (nom)
```

Supprimer une contrainte

```
DROP
  {PRIMARY KEY
   | UNIQUE nom_colonne
   | CONSTRAINT nom_contrainte}
   CASCADE CONSTRAINTS
```

Exemple :

```
ALTER TABLE Departements
   DROP PRIMARY KEY CASCADE CONSTRAINTS
```

On peut aussi modifier le statut d'une contrainte à l'aide de la commande **MODIFY**.

Le langage DML

Le langage DDL permet donc de bien définir notre bases de données, la structure de nos différentes tables, attributs ainsi que les contraintes relatives à chaque tables et attributs.
Cette partie plutôt technique reste cependant très importante pour tout spécialiste en SGBD qui se respecte mais aussi pour modéliser un problème

Le langage DML

Le langage DDL permet donc de bien définir notre bases de données, la structure de nos différentes tables, attributs ainsi que les contraintes relatives à chaque tables et attributs.
Cette partie plutôt technique reste cependant très importante pour tout spécialiste en SGBD qui se respecte mais aussi pour modéliser un problème

La partie qui va nous intéresser est susceptible d'intéresser un public plus important, même non initié.

Manipulation des données

Trois commandes permettant d'effectuer des manipulations de base sur les données

- **INSERT INTO :** qui permet d'ajouter des valeurs dans une ligne d'une table
- **UPDATE :** qui permet de changer les valeurs dans une ligne d'une table
- **DELETE FROM :** qui permet de supprimer les valeurs d'une ligne d'une table

Insérer des données

Ajout d'un enregistrement

```
INSERT INTO nom_table (A1, A2, ...) VALUES (V1, V2, ...)
```

ou, si l'on souhaite remplir tous les attributs

```
INSERT INTO nom_table VALUES (V1, V2, ...)
```

Exemples

- ```
  INSERT INTO Produit
  VALUES (400, "chemise", 78)
  ```
- ```
  INSERT INTO Vendeur (Nom, Prenom, Comm, Fixe, num_secteur)
  VALUES ("Chalbier", "Germaine", 0.03, NULL, 1)
  ```

Modifier des valeurs existantes

Mettre à jour un enregistrement

```
UPDATE nom_table SET Attribut = valeur ou sous requête
```

ou, si l'on souhaite uniquement modifier les lignes respectant une certaine condition

```
UPDATE nom_table SET Attribut = valeur ou sous requête
WHERE conditions
```

Exemples

- ```
  UPDATE Employe
  SET Salaire = 1000;
  ```
- ```
  UPDATE Etudiant
  SET Note = 15 WHERE Nom = "Girfaut"
  ```

Supprimer des valeurs enregistrées

Supprimer un enregistrement

```
DELETE FROM nom_table WHERE conditions
```

Exemples

- `DELETE * FROM Etudiants`

  ou

  `DELETE ALL FROM Etudiants`

  ou

  `TRUNCATE TABLE  Etudiants`

La différence entre **TRUNCATE** et **DELETE** se fait lors de la présence d'un auto-incrément dans la table. **TRUNCATE** réinitialise l'auto-incrément alors que **DELETE** non.

Suite ...

C'est tout ce qu'il y a besoin de savoir sur la manipulation des données ...

On va maintenant se focaliser sur le plus pertinent :

Les requêtes SQL

Structure générale d'une requête

Considérons une requête formée de trois clauses (il en existe bien sûr bien plus !)

```
SELECT liste_attributs
FROM liste_tables
WHERE condition
```

- **SELECT** définit le format du résultat cherché
- **FROM** définit à partir de quelles tables le résultat est calculé
- **WHERE** définit les prédicats de sélection du résultat

Sélection et projection des données

Sélectionner l'ensemble des valeurs d'une table de données

```
SELECT * FROM Employe
```

On peut également créer une requête qui ne retourne qu'un sous-ensemble des attributs, *i.e.* faire une projection des données

```
SELECT A1, A2 FROM nom_table
```

La requête ne retourne que les attributs *A1* et *A2* de la table *nomtable*

Sélection des lignes

Exemples

```
SELECT * FROM PAYS
```

NOM	CAPITALE	POPULATION	SURFACE
Irlande	Dublin	5	70
Autriche	Vienne	9	83
Angleterre	Londres	53	244
Japon	Tokyo	450	398
USA	Washington	314	441

```
SELECT Nom, Capitale FROM Pays
```

NOM	CAPITALE
Irlande	Dublin
Autriche	Vienne
Angleterre	Londres
Japon	Tokyo
USA	Washington

Sélection des lignes

Sélectionner des lignes respectant une ou plusieurs conditon(s) donnée(s)

```
SELECT * FROM Pays WHERE Surface > 100
```

NOM	CAPITALE	POPULATION	SURFACE
Angleterre	Londres	53	244
Japon	Tokyo	450	398
USA	Washington	314	441

On peut bien évidemment mettre plusieurs conditions afin de créer un filtre plus précis sur les données. On peut également combiner la *projection* avec la *sélection*.

Opérateur de comparaison

On peut comparer des nombres

- "=" pour un test d'égalité
- "<>" pour dire "différent de"
- ">" pour dire "plus grand que"
- "<" pour dire "plus petit que"
- **"BETWEEN v1 AND v2"** ou ">**v1 AND** <**v2**" pour dire "compris entre v1 et v2"
- **"NOT BETWEEN v1 AND v2"** ...

Mais on peut aussi faire des tests pour comparer des chaînes de caractère ou des bouts de chaines de caractères (**LIKE**) ou regarder si elle se trouve dans une liste (**IN**).

Opérateur de comparaison

Comparaison totale des chaînes de caractère

- `... WHERE Pays LIKE ...`
- `... WHERE Pays IN Liste_Pays`

Comparaison partielle des chaînes de caractère

- % : un ou plusieurs caractères

  `... WHERE Pays LIKE '%lan%'`
- un underscore pour remplacer exactement un caractère

  `... WHERE Pays LIKE  'I_lande'`

Notez l'existence de **NOT LIKE** par opposition à **LIKE**

Opérateur de comparaison

Comparaison à un ensemble

On cherche par exemple tous les employés ayant un salaire plus grand que les employés administratifs

```
SELECT * FROM Employe
WHERE Salaire > ALL (SELECT Salaire From Employe
                 WHERE Statut = "Administratif)
```

NOM	PRENOM	SALAIRE	STATUT
Bonnot	Jean	2100	Ingénieur
Smith	John	1800	Administratif
Afeu	Pierre	1500	Adminstratif
Lafleur	Marie	1700	Technicien
Rose	Sylvie	2500	DRH

NOM	PRENOM	SALAIRE	STATUT
Bonnot	Jean	2100	Ingénieur
Rose	Sylvie	2500	DRH

Donnez une requête équivalente qui fournirait la même table de résultats.

Sélection des lignes

Il arrive dans certaines tables que des doublons soient présents (erreur humaine ou technique). Dans certaines études, il est parfois intéressant de savoir combien de clients différents fréquentent le magasin sur une période donnée. Cela peut se faire à l'aide de la commande **DISTINCT**

Exemple :

On veut supprimer tous les doublons de personne (Nom, Prenom) se trouvant dans la table.

```
SELECT DISTINCT (Nom, Prenom) FROM Client
```

Intersection entre deux tables

On peut être amené à étudier deux tables portant sur deux enseignes différentes avec des attributs identiques et on souhaite savoir si ces deux enseignes ont des clients en commun.
Pour cela on peut comparer leur fichier en réalisant une **intersection** entre les deux tables.

```
SELECT * FROM table_1
INTERSECT
SELECT * FROM table_2
```

Remarque : Les tuples en double sont éliminés du résultat (pas de copie dans la nouvelle table)

Union et Différence

De la même façon, on peut réaliser une **union** entre deux tables, ce qui peut se révéler pratique lorsque notre table a été initialement fragmenté en plusieurs tables différentes.

```
SELECT * FROM table_1
UNION
SELECT * FROM table_2
```

On peut aussi **retrancher** des éléments existants à une table selon une autre table

```
SELECT * FROM table_1
MINUS
SELECT * FROM table_2
```

Remarque : Les tuples en double sont éliminés du résultat (pas de copie dans la nouvelle table)

Jointure entre deux tables différentes

Dans le cadre d'un traitement, il est souvent nécessaire de croiser les éléments se trouvant sur deux tables ou plus, c'est ce que l'on appelle une **jointure**.

Considérons les deux tables suivantes :

NOM	CAPITALE	POPULATION	SURFACE
Irlande	Dublin	5	70
Autriche	Vienne	9	83
Angleterre	Londres	53	244
Japon	Tokyo	450	398
USA	Washington	314	441

Année	Lieu	Pays
1896	Athènes	Grèce
1900	Paris	France
1904	St Louis	USA
1908	Londres	Uk

Jointure entre deux tables différentes

La jointure se fait en précisant les deux tables utilisées pour la requête mais aussi **l'attribut commun** sur lequel on réaliste la jointure.

Exemple

```
SELECT Annee, Lieu, Pays, Capitale
FROM JO, Pays
WHERE JO.Pays = Pays.Nom
```

Année	Lieu	Pays	Capitale
1908	Londres	UK	Londres
1904	St Louis	USA	Washington

Jointure sur une même table

On se pose souvent ce genre de question lorsque que l'on souhaite comparer un même attribut de plusieurs lignes différentes.

Exemple : Comment comparer les populations des pays ?

```
SELECT P1.nom, P1.population, P2.nom, P2.population
FROM Pays as P1, Pays AS P2
WHERE P1.population > P2.population
```

Toutes les paires de pays telles que le premier pays a une population plus grande que le deuxième pays

Notez l'utilisation d'alias pour faire appel plusieurs fois à la table Pays.

Jointure sur une même table

P1.NOM	P1.POPULATION	P2.NOM	P2.POPULATION
Autriche	9	Irlande	5
Autriche	9	Suisse	8
UK	53	Irlande	5
UK	53	Autriche	9
UK	53	Suisse	8
...	...	...	...

Quelle valeur d'attribut n'apparaitra jamais dans P1.nom ? Idem dans P2.nom ?

Tri des résultats

Il est possible de trier les résultats d'une requête selon un ou plusieurs attribut(s)

```
SELECT * FROM Pays ORDER BY POPULATION DESC
```

NOM	CAPITALE	POPULATION	SURFACE
Japon	Tokyo	450	398
USA	Washington	314	441
Angleterre	Londres	53	244
Autriche	Vienne	9	83
Irlande	Dublin	5	70

Autre option **ASC**.
On peut ordonner salon plusieurs attributs quand cela est nécessaire et surtout en cas de doublon avec le premier attribut mentionné.

Variables calculées

On souhaite aussi calculer certaines grandeurs en fonction des attributs existants, *i.e.* calculer la valeur d'un nouvel attribut.
Par exemple, imaginons que l'on souhaite déterminer la densité de population des différents pays et l'afficher avec les autres colonnes. Tout cela avec seulement deux chiffres après la virgule

```
SELECT *, ROUND(Population/Surface,2) AS Densite
FROM Pays
```

NOM	CAPITALE	POPULATION	SURFACE	Densité
Irlande	Dublin	5	70	0.07
Autriche	Vienne	9	83	0.11
Angleterre	Londres	53	244	0.22
Japon	Tokyo	450	398	1.13
USA	Washington	314	441	0.71

Les fonctions d'agrégations

Quelques fonctions d'agrégations

- **AVG** pour calculer la moyenne des valeurs d'un attribut
- **VARIANCE** : pour calculer la variance des valeurs d'un attribut
- **STDDEV** : pour calculer l'écart-type des valeurs d'un attribut
- **SUM** pour calculer la somme des valeurs d'un attribut
- **MIN** : valeur minimal d'un attribut
- **MAX** : valeur maximum d'un attribut
- **COUNT** : nombre de valeurs d'un attribut

Les quatre premiers sont spécifiques aux attributs numériques et les trois derniers sont globaux

Fonctions d'agrégations

- **Exemple 1** : Quantité total en stock

```
SELECT SUM (Qte)
FROM Produits
```

- **Exemple 2** : Nombre clients

```
SELECT Count(Num_Client)
FROM Client
```

- **Exemple 3** : Le nombre de clients qui ont passé une commande

```
SELECT COUNT(DISTINCT Num_Client)
FROM Commandes
```

- **Exemple 4** : Salaire moyen dans une entreprise

```
SELECT AVG(salaire)
FROM Entreprises
```

Les fonctions d'agrégations

On utilise souvent ces fonction d'agrégations pour calculer une moyenne ou un max, non pas sur l'ensemble des données, mais sur des groupes de données.
On effectue cela à l'aide de la clause **GROUP BY**.

- **Exemple 1** : Quantité en stock par catégorie

```
SELECT Catégorie, SUM(Qte)
FROM Produits
GROUP BY CATEGORIE
```

- **Exemple 2** : Le salaire moyen selon le poste occupé

```
SELECT Poste, AVG(salaire)
FROM Entreprises
GROUP BY Ville
```

Les fonctions d'agrégations

On considère maintenant l'exemple suivant avec une liste de pays semblables aux exemples utilisés précédemment :

NOM	CAPITALE	POPULATION	SURFACE	CONTINENT
Irlande	Dublin	5	70	Europe
Autriche	Vienne	9	83	Europe
Angleterre	Londres	53	244	Europe
Mexique	Mexico	250	289	Amérique
USA	Washington	314	441	Amérique

Et regardons un exemple d'utilisation d'une fonction d'agrégation sur les continents.
Ecrire une requête qui détermine la plus faible, la plus grande et la valeur moyenne de population sur chaque continent ainsi que la somme des surfaces sur chaque continent et le nombre de pays sur chaque continent

Les fonctions d'agrégations

```
SELECT Continent, MIN(Population), MAX(Population), AVG(Populat
       SUM(Surface), COUNT(*)
FROM Pays
GROUP BY Continent
```

CONTINENT	MIN(Population)	MAX(Population)	AVG(Population)	SUM(Surface)	COUNT(*)
Europe	5	53	22..3	397	3
Amérique	250	314	282	730	2

Les fonctions d'agrégations

Une autre fonction de groupement HAVING

Il s'agit d'une condition a posteriori sur un résultat de groupement contrairement à la clause **WHERE**.
La clause **HAVING** ne s'utilise qu'avec **GROUP BY**.

Exemple : on souhaite retourner uniquement les continents dont la surface excède 500.

```
SELECT Continent, SUM(Surface)
FROM Pays
GROUP BY Continent
HAVING SUM(Surface) > 500
```

CONTINENT	SUM(Surface)
Amérique	730

Remarque : nous aurions également pu utiliser un alias pour effectuer cette requête.

Les requêtes imbriquées/emboîtées

On considère la base de données suivante :

- Produit(np, nomp, couleur, poids, prix) : base de produits
- Usine(nu, nomu, ville, pays) : base des usines
- Fournisseur (nf, nomf, type, ville, pays) : base des fournisseurs
- Livraison(np, nu, nf, quantité) : base des livraisons avec les références associées

Les requêtes imbriquées/emboîtées

On souhaite déterminer le nom et la couleur des produits livrés par le fournisseur 1. On a deux solutions possibles.

Les requêtes imbriquées/emboîtées

On souhaite déterminer le nom et la couleur des produits livrés par le fournisseur 1. On a deux solutions possibles.

Solution 1 : une jointure déclarative

```
SELECT nomp, couleur FROM Produit, Livraison
WHERE (Livraison.np = Produit.np) AND nf = 1
```

Les requêtes imbriquées/emboîtées

On souhaite déterminer le nom et la couleur des produits livrés par le fournisseur 1. On a deux solutions possibles.

Solution 1 : une jointure déclarative

```
SELECT nomp, couleur FROM Produit, Livraison
WHERE (Livraison.np = Produit.np) AND nf = 1
```

Solution 2 : une jointure procédurale (emboîtement)

```
SELECT nomp, couleur FROM Produit
WHERE np IN
   (SELECT np FROM Livraison WHERE nf =1)
```

Les requêtes imbriquées/emboîtées

Retour sur la requête une jointure procédurale (emboîtement)

```
SELECT nomp, couleur FROM Produit
WHERE np IN
   (SELECT np FROM Livraison WHERE nf =1)
```

- **IN** compare chaque valeur de *np* avec l'ensemble (ou multi-ensemble) de valeurs retournés par la sous-requête
- **IN** peut aussi comparer des tuples de valeurs

```
SELECT nu FROM Usine
WHERE (ville, pays) IN
   (SELECT ville, pays, FROM Fournisseur)
```

Les requêtes imbriquées/emboîtées

Un autre exemple une jointure procédurale (emboîtement)

```
SELECT nomf
FROM Produit, Usine, Fournisseur, Livraison
WHERE Produit.couleur = 'rouge'
   AND (Usine.ville = 'Londres' OR Usine.ville = 'Paris')
   AND Livraison.np = Produit.np
   AND Livraison.nf = Fournisseur.nf
   AND Produit.np = Livraison.np
```

Que fait la requête ci-dessus ?

Les requêtes imbriquées/emboîtées

Un autre exemple une jointure procédurale (emboîtement)

```
SELECT nomf
FROM Produit, Usine, Fournisseur, Livraison
WHERE Produit.couleur = 'rouge'
   AND (Usine.ville = 'Londres' OR Usine.ville = 'Paris')
   AND Livraison.np = Produit.np
   AND Livraison.nf = Fournisseur.nf
   AND Produit.np = Livraison.np
```

Que fait la requête ci-dessus ?

Elle nous renseigne sur le nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge

Les requêtes imbriquées/emboîtées

Exercice

Ecrire une requête qui nous renseigne sur le nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge. La requête doit se faire sous forme de requêtes emboîtées !

```
SELECT nomf
FROM Fournisseur
WHERE nf IN
   (SELECT nf FROM Livraison
   WHERE np IN
      (SELECT np FROM Produit
      WHERE couleur = 'rouge')
   AND nu IN
      (SELECT nu FROM Usine
      WHERE ville = 'Londres' OR ville = 'Paris'))
```

Les requêtes imbriquées/emboîtées

Quantificateur ALL

On souhaite maintenant avoir le numéro des fournisseurs qui ne fournissent **que des produits rouges**

Les requêtes imbriquées/emboîtées

Quantificateur ALL

On souhaite maintenant avoir le numéro des fournisseurs qui ne fournissent **que des produits rouges**

```
SELECT nf FROM Fournisseur
WHERE 'rouge' = ALL
   (SELECT couleur FROM Produit
   WHERE np IN
      (SELECT np FROM Livraison
      WHERE Livraison.nf = Fournisseur.nf))
```

- La requête imbrique est ré-évaluée pour chaque tuple de la requête (ici pour chaque *nf*
- tous les éléments de l'ensemble doivent vérifier la condition

Les requêtes imbriquées/emboîtées

Quantificateur EXISTS

On souhaite maintenant avoir le nom des fournisseurs qui fournissent au moins un produit rouge

Les requêtes imbriquées/emboîtées

Quantificateur EXISTS

On souhaite maintenant avoir le nom des fournisseurs qui fournissent au moins un produit rouge

```
SELECT nomf FROM Fournisseur
WHERE EXISTS
   (SELECT *
   FROM Livraison, Produit
   WHERE Livraison.nf = Fournisseur.nf
      AND Livraison.np = Produit.np
      AND Produit.couleur = 'rouge')
```

- **EXISTS** test si l'ensemble n'est pas vide.

Les jointures - JOIN

Nous avojs déjà vu comment faire des jointures entre des tables à l'aide de la clause **WHERE** à l'aide des clés primaires. C'est ce que l'on appelle une jointure **interne**, *i.e.* qui ne sélectionne que les données qui ont une correspondance entre les deux tables.
Il existe une autre façon de formuler une jointure **interne** à l'aide de la fonction clause **JOIN**.

Les jointures - JOIN

Nous avojs déjà vu comment faire des jointures entre des tables à l'aide de la clause **WHERE** à l'aide des clés primaires. C'est ce que l'on appelle une jointure **interne**, *i.e.* qui ne sélectionne que les données qui ont une correspondance entre les deux tables.
Il existe une autre façon de formuler une jointure **interne** à l'aide de la fonction clause **JOIN**.

A l'image des jointures **internes**, il existe également des jointures **externes** qui, elles, sélectionnent toutes les données, même si certaines n'ont pas de correspondance dans l'autre table.

Les jointures - JOIN

Quelques clauses pour effectuer une jointure

- **INNER JOIN** : retourne les enregistrements lorsque la condition est vérifiée dans les deux tables
- **CROSS JOIN** : effectue le produit cartésien entre deux tables
- **LEFT JOIN** : retourne tous les éléments de la table de gauche même si la condition n'est pas vérifiée dans l'autre table
- **RIGHT JOIN** : même chose mais à droite
- **FULL JOIN** : retourne les résultats quand la condition est vraie dans au moins une des deux tables.
- **SELF JOIN** : permet d'effectuer la jointure d'une table avec elle même
- **NATURAL JOIN** : jointure naturelle entre deux tables s'il y a au moins une colonne qui porte le même nom

On se focalisera uniquement sur les clauses **INNER JOIN**, **LEFT JOIN** and **RIGHT JOIN**.

Les jointures - JOIN

On considère les deux tables suivantes : une table *Jeux* et une table *Joueurs* contenant différentes informations relatives à des jeux vidéos et aux propriétaires des jeux vidéos.

ID	Nom	ID_proprietaire	Console	Prix
1	Super Mario Bros	1	NES	4
2	Sonic	2	Megadrive	2
3	Zelda : Ocarina of time	1	Nintendo 64	15
4	Mario Kart 64	1	Nintendo 64	25
5	Super Smash Bros Melee	3	GameCube	55
6	Bomberman	6	NES	10

ID	Prenom	Nom	Téléphone
1	Florent	Dugommier	01 44 77 21 33
2	Patrick	Lejeune	03 22 17 41 22
3	Michel	Doussand	04 11 78 02 00
4	Romain	Vipelli	01 21 98 51 01

Les jointures Internes

Partant des deux tables précédentes, je souhaite associer retrouver le nom du propriétaire de chaque jeu.
Il s'agit ici d'une jointure **interne** que l'on peut effectuer avec la clause **WHERE** :

Les jointures Internes

Partant des deux tables précédentes, je souhaite associer retrouver le nom du propriétaire de chaque jeu.
Il s'agit ici d'une jointure **interne** que l'on peut effectuer avec la clause **WHERE** :

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Jeux as j, Joueurs as p
WHERE j.ID_proprietaire = p.ID
```

Les jointures Internes

Partant des deux tables précédentes, je souhaite associer retrouver le nom du propriétaire de chaque jeu.
Il s'agit ici d'une jointure **interne** que l'on peut effectuer avec la clause **WHERE** :

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Jeux as j, Joueurs as p
WHERE j.ID_proprietaire = p.ID
```

Mais on peut aussi le faire à l'aide de la clause **INNER JOIN**

Les jointures Internes

Partant des deux tables précédentes, je souhaite associer retrouver le nom du propriétaire de chaque jeu.
Il s'agit ici d'une jointure **interne** que l'on peut effectuer avec la clause **WHERE** :

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Jeux as j, Joueurs as p
WHERE j.ID_proprietaire = p.ID
```

Mais on peut aussi le faire à l'aide de la clause **INNER JOIN**

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Joueurs as p
INNER JOIN jeux as j
ON j.ID_proprietaire = p.ID
```

Les jointures internes

Nom_jeu	Nom_proprietaire
Super Mario Bros	Florent
Sonic	Patrick
Zelda : Ocarina of time	Florent
Mario Kart 64	Florent
Super Smash Bros Melee	Michel

Les jointures externes

Ces jointures permettent de récupérer toutes les données, même celles qui n'ont pas de correspondance.
Exemple : une jointure externe à gauche, on remplace **INNER** par **LEFT**, ce qui nous donne

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Joueurs as p
LEFT JOIN jeux as j
ON j.ID_proprietaire = p.ID
```

Les jointures externes

Ces jointures permettent de récupérer toutes les données, même celles qui n'ont pas de correspondance.
Exemple : une jointure externe à gauche, on remplace **INNER** par **LEFT**, ce qui nous donne

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Joueurs as p
LEFT JOIN jeux as j
ON j.ID_proprietaire = p.ID
```

Nom_jeu	Nom_Proprietaire
Super Mario Bros	Florent
Zelda : Ocarina of Time	Florent
Mario Kart 64	Florent
Sonic	Patrick
Super Smash Bros	Michel
NULL	Romain

Les jointures externes

Remarques :

Dans ce premier exemple la table *Joueurs* sert de table de référence et toute les lignes de joueurs apparaissent bien dans la table de résultat (même pour les joueurs qui ne possèdent pas de jeux vidéos !!!)
Dans ce cas, un joueur qui ne possède pas jeu vidéo se verra associer la valeur **NULL**

On associe ensuite à chaque joueur les jeux vidéos qu'il possède.

In fine, un joueur apparaîtra autant de fois dans la table qu'il possède de jeux vidéos.

Les jointures externes

Exemple :

Une jointure externe à droite, on remplace **INNER** par **RIGHT**, ce qui nous donne

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Joueurs as p
RIGHT JOIN jeux as j
ON j.ID_proprietaire = p.ID
```

Les jointures externes

Exemple :

Une jointure externe à droite, on remplace **INNER** par **RIGHT**, ce qui nous donne

```
SELECT j.nom AS nom_jeu, p.prenom as Nom_proprietaire
FROM Joueurs as p
RIGHT JOIN jeux as j
ON j.ID_proprietaire = p.ID
```

Nom	ID_proprietaire
Super Mario Bros	Florent
Sonic	Patrick
Zelda : Ocarina of time	Florent
Mario Kart 64	Florent
Super Smash Bros Melee	Michel
Bomberman	NULL

Contrôle d'accès aux données

- Les BDD comptent souvent plusieurs utilisateurs, notamment lorsqu'elles sont partagées en réseau et tous, n'ont pas nécessairement des besoins identiques.

- Tous les utilisateurs ne sont donc pas habilités à consulter et modifier toutes les données d'une base. SQL offre des fonctions de règlementation des droits d'accès aux données sous la forme de privilèges.

- Seul l'administrateur qui a créé un élément (une table ou une vue) a la possibilité d'accorder ou de retirer des droits sur cet élément.
Ex : accorder le droit de visionner le contenu d'une table, de modifier une table partiellement ou totalement, supprimier une table, ...

Contrôle d'accès aux données : Principe

Un privilège est l'autorisation qui est accordée à un utilisateur d'effectuer une opération sur un objet. Un privilège concerne donc, et **doit spécifier**, une opération, un objet (ressource) de la base de données ou de son environnement, l'utilisateur qui accorde le privilège (c'est celui qui exécute la requête de création ou retrait du privilège) et celui qui le reçoit.

Privilèges sur une table

Les principales opérations admises sur le contenu des tables sont les suivantes :

- **SELECT** : extraction des données
- **INSERT** : ajout de lignes
- **DELETE** : suppression de lignes ou tables
- **UPDATE** : modification des valeurs de certaines colonnes

La commande ci-dessous permet aux utilisateurs nommés de consulter le contenu d'une table *Produit* et de modifier les valeurs de *Qstock* et *Prix* *(ex : des droits accordés aux personnels en gestion des stocks)*

```
GRANT select, update(Qstock, Prix)
ON Produit
TO P_Mercier, S_Financiers
```

Privilèges sur une table

Il est également possible de transmettre au destinataire d'un privilège, le privilège de transmettre celui-ci à d'autres utilisateurs. La commande ci-dessous permet aux utilisateurs aux utilisateurs nommés d'effectuer n'importe quelle manipulation sur la table *Produit* mais aussi de transmettre ce privilège s'il juge cela nécessaire

```
GRANT ALL
ON Produit
TO P_Mercier, S_Financiers
WITH grant option
```

Un privilège peut aussi être accorder à l'ensemble des utilisateurs

```
GRANT select
ON Produit
TO public
```

Privilèges sur une table

Il existe d'autres privilèges liés notamment à l'administration de la base de données, en particulier à la définition et à la modification de structure de données : **CREATE TABLE**, **ALTER TABLE**, **DROP INDEX** ou encore **DELETE**.

Tout comme il est possible d'accorder des droits/privilèges à des utilisateurs, il est également possible de leur **retirer des droits** : ce qui est plutôt pratique lorsque certains utilisateurs, extérieurs à une entreprise, on fini de travailler sur un projet et qu'ils n'ont donc plus besoin d'avoir accès à une base de données.

Privilèges sur une table : REVOKE

La commande **REVOKE** permet de retirer un privilège préalablement accorder à un utilisateur

```
REVOKE update(Prix)
ON Produit
TO P_Mercier
```

```
REVOKE run
ON Compta
FROM P_Mercier
```

Remarques :

- lorsqu'un même privilège fut accordé à un même utilisateur par plusieurs personnes indépendantes, alors cet utilisateur disposera de ce privilège tant que chacune des personnes ne le lui aura pas retiré.
- il est aussi possible de retirer la faculté de transmettre un privilège par une commande telle que :

```
REVOKE grant option for update (Compte)
ON Client
FROM P_Mercier
```

Exemples :

Ecrire les commandes permettant d'effectuer les demandes suivantes :

1) Le droit de d'ajouter et de consulter des enregistrements de la table Clients est accorder à tous les utilisateurs
2) Pierre obtient tous les privilèges sur la table Client et a le droit d'accorder des privilèges à d'autres utilisateurs
3) Tous les utilisateurs ont le droit de consulter toutes les tables
4) Marine a le droit d'ajouter des lignes à la table Produit
5) Marine a le droit de modifier le contenu de la table Catalogue et peut déléguer ce droit
6) Thomas et Pierre peuvent mettre à jour les prix proposés par les fournisseurs

Exemples :

1)
```
GRANT SELECT, INSERT
ON Clients
TO PUBLIC
```

2)
```
GRANT ALL
ON Client
TO Pierre
WITH Grant Option
```

3)
```
GRANT SELECT
ON ALL
TO PUBLIC
```

Exemples :

4)
```
GRANT INSERT
ON Produit
TO Marine
```

5)
```
GRANT UPDATE
ON Catalogue
TO Marine
WITH GRANT OPTION
```

6)
```
GRANT UPDATE(Prix)
ON Fournisseurs
TO Thomas, Pierre
```

Exemples :

Il nous reste encore à regarder comment créer des utilisateurs.
Cela se fait à l'aide de la commande **CREATE USER**.

Création d'utilisateurs

```
CREATE USER Nom_User WITH PASSWORD = '******' MUST_CHANGE
```

Exemple :

```
CREATE USER Guillaume WITH PASSWORD = '1234' MUST_CHANGE
```

Supprimer des utilisateurs

```
DROP USER Nom_User
```

Etudes de cas

Objectifs

- On se propose maintenant de mettre en pratique tout ce que nous avons vu (enfin pas tout en globalité non plus !). Plus précisément partant d'un énoncé on va voir comment construire le schéma entité-association et ensuite construire le schéma relationnel ou directement le schéma de la base de données.
- Le travail sera effectué sur deux exemples :

  Les animaux d'un zoo : que l'on va effectuer ensemble

  Les voyages en avion : que vous allez faire ... tout seul !

Process

- Il va déjà falloir effectuer une lecture attentive de l'énoncé et décomposer cet énoncé afin d'identifier les informations les plus importantes et qui seront nécessaires à l'élaboration du schéma entité/association. Cela constituera la première mais aussi une étape fondamentale pour la suite
- Représenter le schéma EA en affichant les entités et les associations ainsi que les liens entres les objets. Vous prendrez soins de bien préciser les identifiants de chacune de entités.
- Enfin, vous présenterez le schéma relationnel ou directement le schéma de la base de données avec les différentes références étrangères.
- Et enfin (last but not least !), vous écrirez les requêtes SQL permettant de créer votre base de données (*i.e.* les différentes tables).

Enoncé : Les animaux d'un zoo

Les animaux d'un zoo suivent chacun un régime alimentaire. Un régime est constitué d'un mélange d'ingrédients, chacun en quantité déterminée. Le régime d'un animal peut varier d'un jour à l'autre. Chaque animal est caractérisé, en fonction de son espèce, par ses besoins minima et maxima en nutriments (calcium, protéines, etc ...), exprimés en mg par unité de poids de l'animal. Ces besoins sont fonctions de l'espèce de l'animal. On connaît la teneur de chaque ingrédient en nutriments, exprimée en mg par kg d'ingrédient. Chaque ingrédient a un coût unitaire. Chaque animal requiert des soins qui sont évalués en francs par jour. Ces soins peuvent varier d'un jour à l'autre.

Etape 1 : Décortiquer l'énoncé

1) *un zoo a des animaux*
   Le zoo est le domaine d'application, qu'on ne représente pas explicitement pour le moment. On revanche, on peut définir le type d'entité **ANIMAL**
2) *un animal suit un régime*
   Un nouveau type d'entité **REGIME** est défini et relié à **ANIMAL** par l'association *suit*. On peut ensuite considérer qu'un régime est propre à un animal.
3) *un régime est constitué/composé d'ingrédients*
   On définit une nouvelle entité **INGREDIENT**, relié à **REGIME** par l'association composé de. On prendra garde qu'un ingrédient peut entrer dans la composition de plusieurs régimes.

Etape 1 : Décortiquer l'énoncé

4) *un ingrédient entre dans la composition d'un régime en une quantité déterminée*
   Cette composition est caractérisée par une quantité. Donc pas d'associations pour *composé de* —> entité **COMPOSITION** qui va être lié à **REGIME** et **INGREDIENT**.
5) *le régime que reçoit un animal dépend du jour*
   Un animal peut donc avoir plusieurs régimes, ce qui nous renseigne sur le type d'associations. On ajoute un attribut *DateRegime* à l'entité **REGIME**. Cette phrase nous renseigne aussi sur les identifiants de **REGIME** : *DateRegime* et **Animal**.
6) *un animal est d'une espèce*
   On affecte à **ANIMAL** un attribut *Espèce*

Etape 1 : Décortiquer l'énoncé

7) *une espèce a des besoins en nutriments* Nouvelle entité **NUTRIMENT**. Il devrait être relié à **ESPECE** par un type d'associations, ce qui n'est possible que si l'on transforme cet attribut en type d'entités **ESPECE**, relié à **ANIMAL**. Il ne reste plus qu'à déterminer le type précis des associations (plusieurs-plusieurs) et lui donner un nom.
8) *calcium, protéines sont des exemples de nutriments*
   Cela suggère un attribut *NomNutriment*, identifiant de **NUTRIMENT**.
9) *chaque besoin d'un animal en un ingrédient est caractérisé par une quantité minimale ...*
   Besoin **ANIMAL** = besoin **ESPECE** donc pas de lien supplémentaire. On voit aussi la nécessité de transformer l'association *besoin de* en une entité et d'y ajouter l'attribut *minimale*

Etape 1 : Décortiquer l'énoncé

10) *et une quantité maximale*
    On ajoute l'attribut *maximale* à l'entité nouvelle créée **BESOIN DE**.
11) *un animal à un poids*
    ANIMAL reçoit l'attribut *Poids*
12) *ces besoins dépendent de l'espèce animal*
    Cette propriété a déjà été exprimée
13) *un ingrédient contient des nutriments*
    On définit une association *contient* entre **INGREDIENT** et **NUTRIMENT** sachant qu'un nutriment peut intervenir dans plusieurs ingrédients.

Etape 1 : Décortiquer l'énoncé

14) *chacun en une teneur déterminée*
Une teneur est une quantité qui caractérise un nutriment en tant que composant d'un ingrédient. Un attribut *quantité* est défini, mais il ne peut être affecté à **NUTRIMENT**. En effet, il y autant de teneurs qu'il y a d'ingrédients où un nutriment apparaît), pas plus qu'à **INGREDIENT** (il y a autant de teneurs qu'il y a de nutriments qui composent l'ingrédient). Il doit donc être affecté au lien qui unit ces deux entités, le type d'associations *contient*. Ce dernier doit donc être transformé en une entité **TENEUR**.

15) *un ingrédient a un coût unitaire*
On affecte un attribut *coût* unitaire à **INGREDIENT**.

16) *un animal requiert des soins*
Chaque prestation de soin à un animal est représentée par une entité **SOINS**. On a donc une association entre **SOINS** et **ANIMAL** qu'il reste à déterminer.

Etape 1 : Décortiquer l'énoncé

17) *les soins d'un animal ont un coût*
    On affecte un attribut *coût* au type d'entités **SOINS**
18) *les soins d'un animal dépendent du jour*
    Le type d'entités **SOINS** reçoit un attribut *DateSoin*. Etant donné l'unité de coût, une entité de **SOINS** représente l'ensemble des prestations destinées à un animal, un jour déterminé —> nouvel identifiant de **SOINS**

Une dernière chose ... rien ne vous perturbe concernant **ANIMAL** ?

Etape 1 : Décortiquer l'énoncé

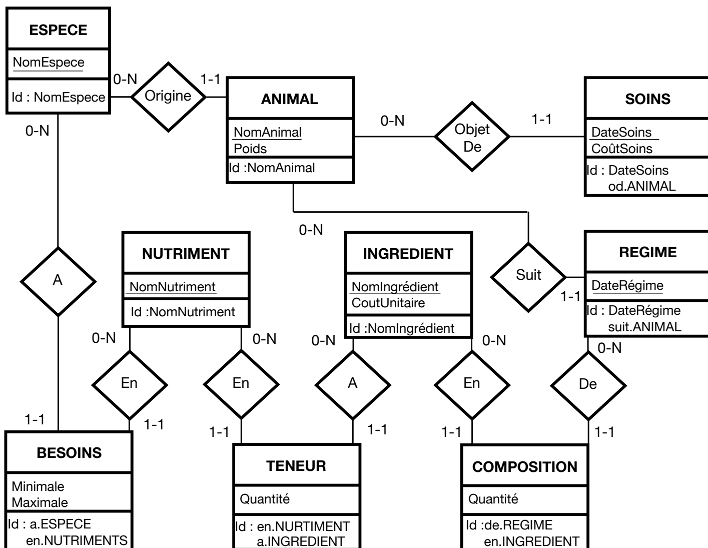
17) *les soins d'un animal ont un coût*
On affecte un attribut *coût* au type d'entités **SOINS**

18) *les soins d'un animal dépendent du jour*
Le type d'entités **SOINS** reçoit un attribut *DateSoin*. Etant donné l'unité de coût, une entité de **SOINS** représente l'ensemble des prestations destinées à un animal, un jour déterminé —> nouvel identifiant de **SOINS**

Une dernière chose ... rien ne vous perturbe concernant **ANIMAL** ? Il faut lui ajouter un attribut *NomAnimal* pour des raisons de complétudes (oui oui ... un poids n'est pas suffisant pour caractériser l'animal !)

Plus qu'à représenter le schéma !

Etape 2 : Schéma E/A



Etape 3 : Schéma relationnel

Finalement, une fois que le travail du schéma est effectué, il s'agit là de la partie la plus simple, il faut simplement faire attention à bien écrire les clés primaires et indiquer les clés étrangères.

ESPECE(NomEspece)
ANIMAL(NomAnimal, Poids, # NomEspece)
SOINS(DateSoins, CoutSoins, #NomAnimal)
NUTRIMENT(NomNutriment)
INGREDIENT(NomIngredient, CoutUnitaire)
REGIME(DateRegime, #NomAnimal)
BESOINS(#NomEspece, #NomNutriments, Minimale, Maximale)
TENEUR(#NomNutriments, #NomIngredient, Quantite)
COMPOSITION(#DateRegime, #NomIngredient, Quantite)

Etape 4 : Ecriture des tables

Il ne nous reste plus qu'à écrire les différentes requêtes pour créer notre base de données ... Let's Go !

```
CREATE DATABASE Zoo

CREATE TABLE Espece (
   NomEspece char(32) NOT NULL,
   PRIMARY KEY(NomEspece))

CREATE TABLE Animal(
   NomAnimal CHAR(32) NOT NULL,
   Poids NUMERIC(8,4) NOT NULL,
   NomEspece CHAR(32) NOT NULL,
   PRIMARY KEY (NomAnimal),
   FOREIGN KEY(NomEspece) REFERENCES Espece)
```

Etape 4 : Ecriture des tables

```
CREATE TABLE Soins(
   NomAnimal CHAR(32) NOT NULL,
   DateSoins NUMERIC(6) NOT NULL,
   CoutSoins NUMERIC(6,2) NOT NULL,
   PRIMARY KEY(NomAnimal, DateSoins),
   FOREIGN KEY(NomAnimal) REFERENCES Animal)

CREATE TABLE NUTRIMENT(
   NomNutriment CHAR(32) NOT NULL,
   PRIMARY KEY(NomNutriment))

CREATE TABLE INGREDIENT(
   NomIngredient CHAR(32) NOT NULL
   CoutUnitaire numeric(8,2) NOT NULL,
   PRIMARY KEY(NomIngredient))
```

Etape 4 : Ecriture des tables

```
CREATE TABLE Regime(
   NomAnimal CHAR(32) NOT NULL,
   DateRegime NUMERIC(6,2) NOT NULL,
   PRIMARY KEY (NomAnimal, DateRegime),
   FOREIGN KEY (NomAnimal) REFERENCES Animal)

CREATE TABLE Besoins(
   NomEspece CHAR(32) NOT NULL,
   NomNutriment CHAR(32) NOT NULL,
   Minimale NUMERIC(8,4) NOT NULL,
   Maximale NUMERIC(8,4) NOT NULL,
   PRIMARY KEY (NomNutriment, NomEspece)
   FOREIGN KEY (NomEspece) REFERENCES Espece)
   FOREIGN KEY (NomNutriment) REFERENCES Nutriment)
   CHECK (Maximale >= Minimale))
```

Etape 4 : Ecriture des tables

```
CREATE TABLE Teneur(
   NomNutriment CHAR(32) NOT NULL,
   NomIngredient CHAR(32) NOT NULL,
   Quantite NUMERIC(8,4) NOT NULL,
   PRIMARY KEY(NomIngredient, NomNutriment),
   FOREIGN KEY(NomNutriment) REFERENCES Nutriment,
   FOREIGN KEY(NomIngredient) REFERENCES Ingredient)

CREATE TABLE Composition(
   DateRegime NUMERIC(6) NOT NULL,
   NomAnimal CHAR(32) NOT NULL,
   NomIngredient CHAR(32) NOT NULL,
   PRIMARY KEY(NomIngredient, NomAnimal),
   FOREIGN KEY(NomAnimal) REFERENCES Animal,
   FOREIGN KEY(NomIngredient) REFERENCES Ingredient)
```

A vous de jouer : Les voyages en avion

Voir feuille distribuée pour vous entraîner
L'exercice est plutôt complexe

References I

Jean-Luc-Hainaut (2005). *Bases de Données et modèles de calcul : Outils et méthodes pour l'utilisateur*. Dunod.