hestim
Engineering & Business School

Filière : MGO

Initiation aux bases de données

Professeur:

Korota Arsène COULIBALY

kcoulibaly@hestim.ma

Année Universitaire: 2022-2023

PLAN DU COURS

BASES DE DONNÉES ET SYSTÈMES DE GESTION DE BASES DE DONNÉES

1. Définition d'une base de données
2. Modèles de base de données
3. Systèmes de gestion de bases de données(SGBD)
4. Domaines d'utilisation des bases de données

RÔLES D'UNE BASE DE DONNÉES

DES SYSTÈMES DE GESTION DE FICHIERS AUX SGBD

LANGAGES DE MANIPULATION DES DONNÉES

1. Algèbre relationnelle
2. Langage SQL

ORGANISATION DU COURS

- 7 séances CM+TD+TP
- 1 examen écrit(70%) + un TP(30%)

OBJECTIFS

- Savoir définir une base de données
- Connaitre l'utilité d'un système de gestion de base de données
- Savoir pourquoi et quand utiliser une base de données
- Avoir une idée de comment interroger et manipuler une base de donnée

BASES DE DONNÉES ET SYSTÈMES DE GESTION DE BASES DE DONNÉES

hestim
Engineering & Business School

1- Définitions :

Abréviation : **DB**(en anglais) pour Data Base et **BD**(en français) pour Base de Données.

Système d'Information(SI) : Un système d'information désigne la structure regroupant les moyens mis en œuvre pour pouvoir partager des données.

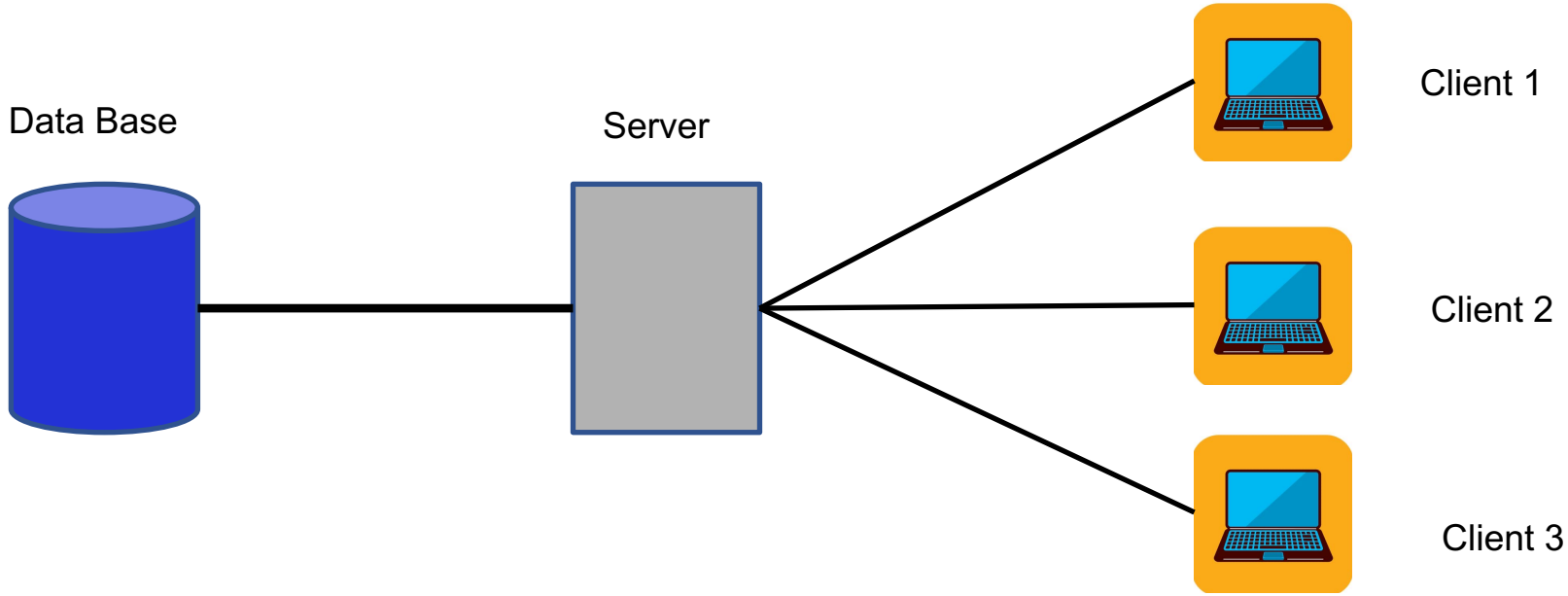
Base de Données(1) : est une entité dans laquelle il est possible de stocker des informations de manière structurée et avec le moins de redondance possible.

Base de Données(2) :

- Un ensemble structuré
- Organisé
- Permettant le stockage de grandes quantités de d'informations afin d'en faciliter l'exploitation : **Ajout**, **Mise à jour**, **recherche de données**

1- Définitions :

Base de Données(3) : Physiquement une base de donnée constitue un ensemble de fichiers présents sur une mémoire(généralement un disque) et accessibles simultanément par plusieurs utilisateurs ou clients .

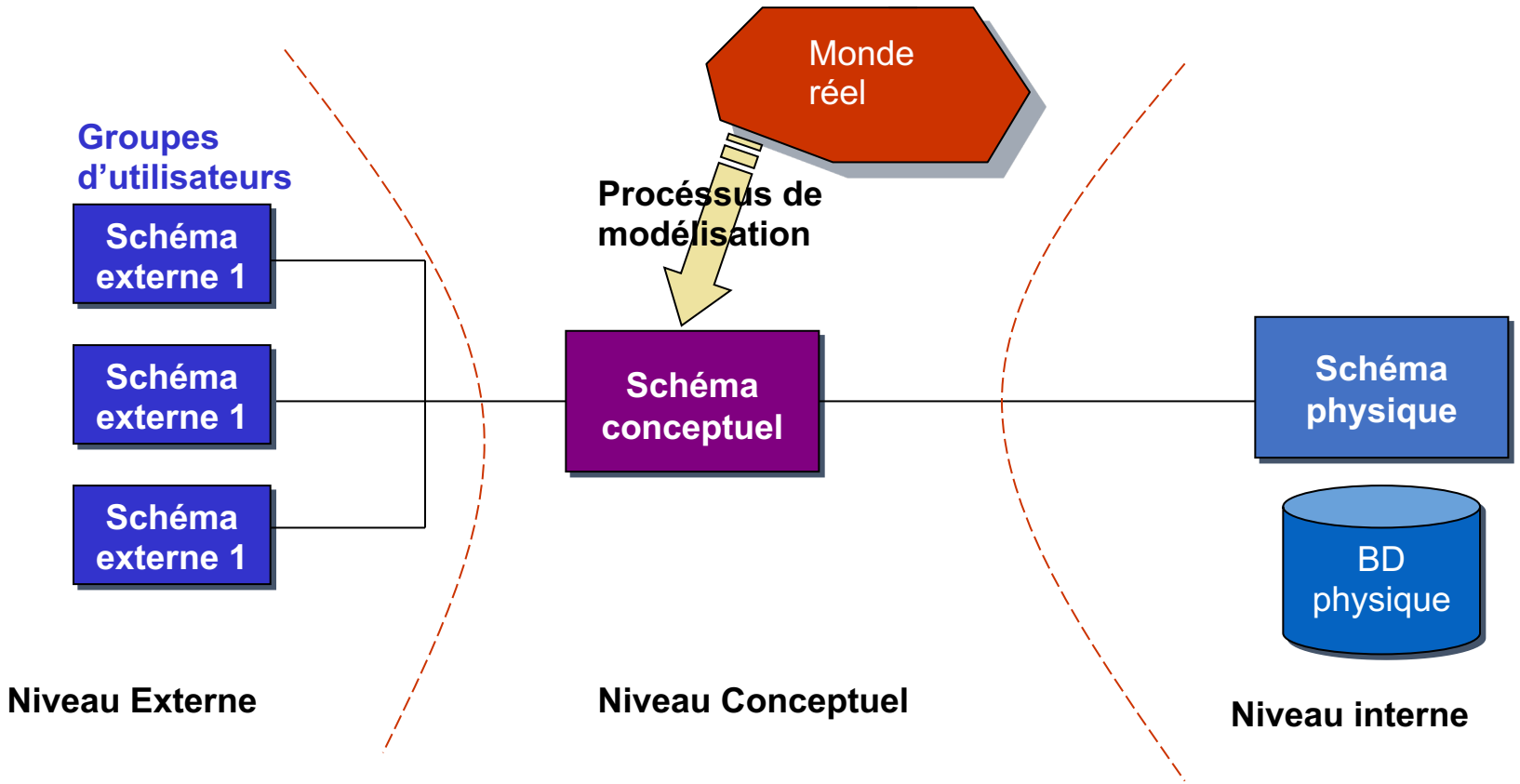


2- Les niveaux de représentations d'une base données :

a- Le niveau externe

b- Le niveau conceptuel

c- Le niveau interne



Le modèle Modèle conceptuel de données(MCD) Entités-Associations (E-A)

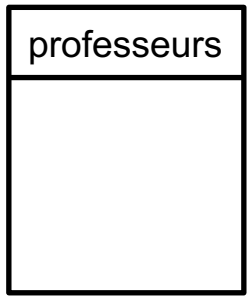
1- Définitions :

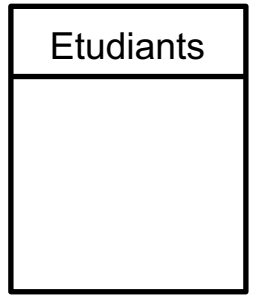
E-A : est une méthode de modélisation intermédiaire permettant une description naturelle du monde réel à partir des concepts d'entités et d'associations. Le modèle est basé sur la théorie des ensembles et des relations.

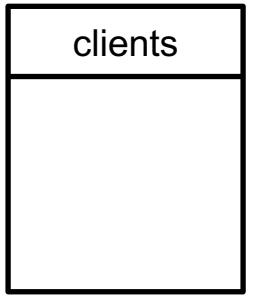
Le modèle Modèle conceptuel de données(MCD) Entités-Associations (E-A)

2- MCD Entité :

Entité : Une Entité est une population d'individus homogènes. Par exemple, les produits ou les articles vendus par une entreprise peuvent être regroupés dans une même entité articles, car d'un article à l'autre, les informations ne changent pas de nature (a chaque fois, il s'agit de la désignation, du prix unitaire, etc.).



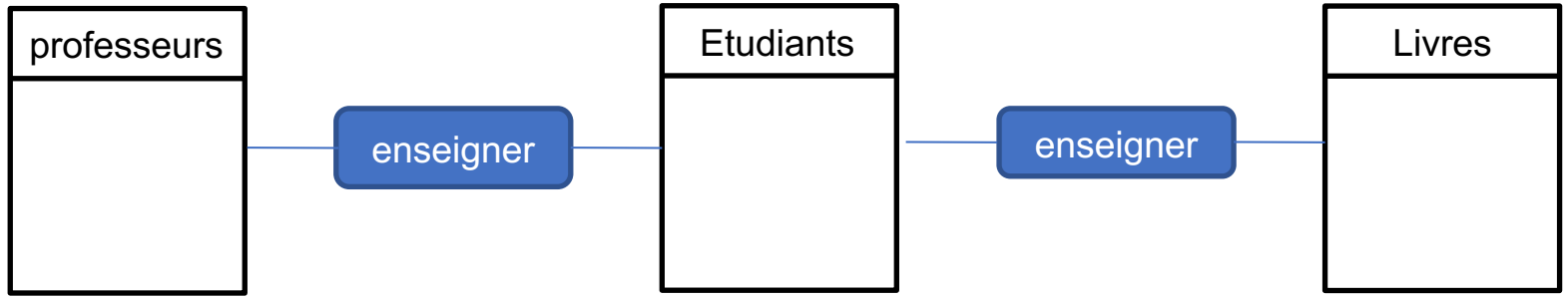




Le modèle Modèle conceptuel de données(MCD) Entités-Associations (E-A)

3- MCD Association :

Association : Une Association est une liaison qui a une signification précise entre plusieurs entités. Dans notre exemple, l'association enseigner est une liaison évidente entre les entités professeurs et Etudiants, tandis que l'association Lire établit le lien sémantique entre les entités articles et fournisseurs.

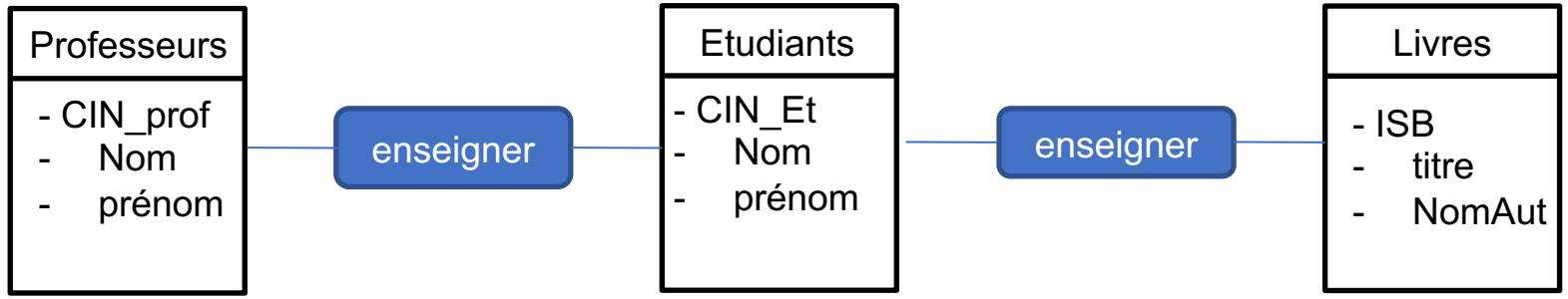


Le modèle Modèle conceptuel de données(MCD) Entités-Associations (E-A)

4- MCD Attribut :

Attribut : Un Attribut est une propriété d'une entité ou d'une association.

Exemple : le **nom de famille** est un attribut de l'entité **Professeurs**, le **prénom** est un attribut de l'entité Professeurs, l'ISBN, le titre et l'auteur d'un livre sont des attributs de l'entité **Livres.**



Le modèle Modèle conceptuel de données(MCD) Entités-Associations (E-A)

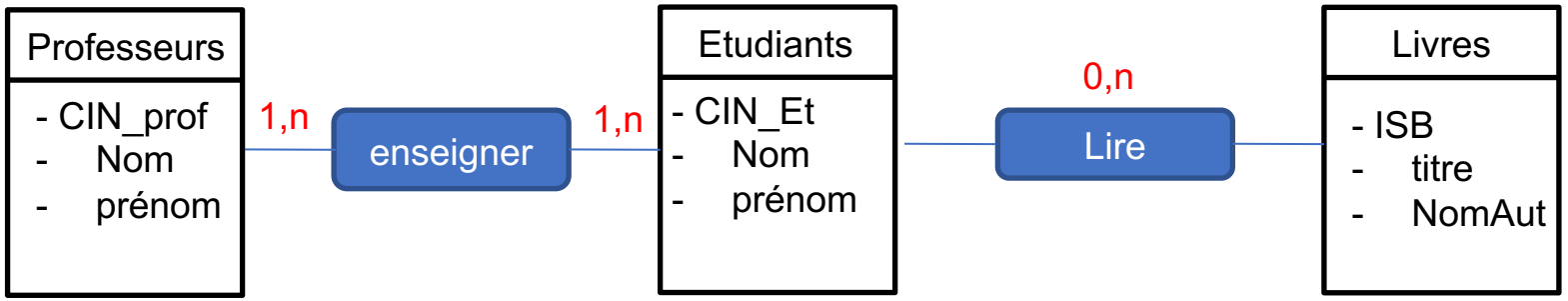
5- MCD cardinalité :

cardinalité: La Cardinalité d'un lien entre une entité et une association précise le minimum et le maximum de fois qu'un individu de l'entité peut être concerné par l'association.

- Une cardinalité minimale est toujours 0 ou soit 1 et une cardinalité maximale est 1 ou n
- Si une cardinalité maximale est connue et vaut 2, 3 ou plus alors on considère qu 'elle est indéterminée et vaut n.

Exemple : Un Professeur peut enseigner 1 ou plusieurs(n) étudiants.

Un étudiant est peut lire soit 0 ou n livres et un livre est soit lu par 0 ou n étudiants



hestim
Engineering & Business School

NIVEAU CONCEPTUEL

a- Entité ou objet : l'entité est définie comme un objet de gestion considéré d'intérêt pour représenter l'activité à modéliser (exemple : entité Professeur). A son tour, chaque entité (ou objet) est porteuse d'un ou plusieurs attributs simples, (exemples : CIN_prof, nom, ...) dont l'un de ses attributs doit être unique et désigné comme identifiant (exemple : CIN_prof).

b- Association : l'association est un lien sémantique entre entités :

- 1 entité reliée à elle-même : la relation est dite réflexive,
- 2 entités : la relation est dite association binaire
- Plus rarement 3 ou plus :
  - Une association entre trois entités est appelée association ternaire
  - Une association entre n entités est appelée association n-aire
  - Une association peut également être porteuse d'une ou plusieurs propriétés.

NIVEAU CONCEPTUEL

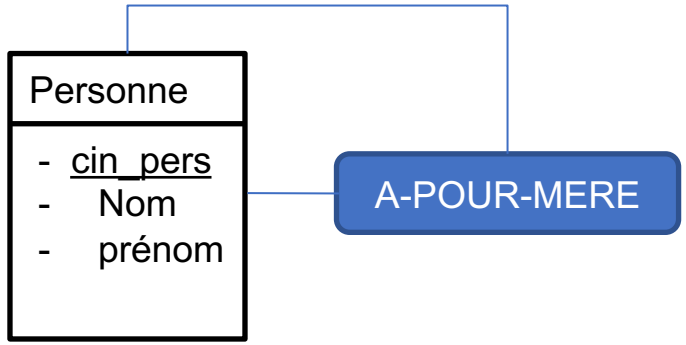
c- cardinalité: indique le nombre minimum (0 ou 1) et maximum (1 ou n) de fois où une occurrence quelconque d'une entité peut participer à une association (ex: un Professeur enseigne un (card. Min=1) ou plusieurs (card. Max=N) étudiants et réciproquement un étudiant est enseigné par 1 Professeur (card. Min=1) ou plusieurs (card. Max=N)). On a donc les combinaisons suivantes :

- (0,1)
- (0,n)
- (1,1)
- (1,n)

NIVEAU CONCEPTUEL

d- exemple d'une association réflexive: C'est une association d'une entité sur elle-même. En effet, il est parfaitement possible d'établir une association entre une entité et elle-même, définissant par là une association cyclique.

Pour traduire le fait que Irène Curie est la fille de Marie Curie on pourra utiliser une association **A-POUR-MERE** entre les deux entités représentant ces personnes.



NIVEAU CONCEPTUEL

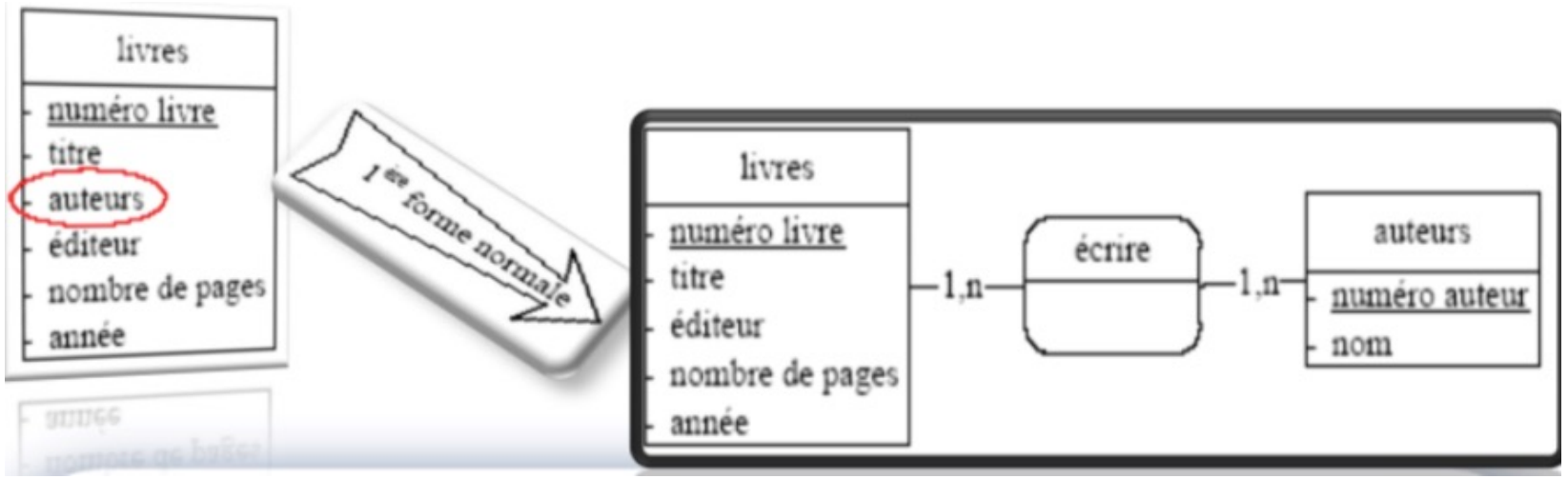
d- règles de normalisation : Un schéma entité-association doit obéir à 9 règles dont 6 règles de normalisation plus les 3 formes normales :

1. **Normalisation des entités (importante)** : toutes les entités qui sont remplaçables par une association doivent être remplacées.
2. **Normalisation des noms** : le nom d'une entité, d'une association ou d'un attribut doit être unique.
3. **Normalisation des identifiants** : chaque entité doit posséder un identifiant.
4. **Normalisation des attributs (importante)** : remplacer les attributs en plusieurs exemplaires en une association supplémentaire de cardinalités maximales et ne pas ajouter d'attribut calculable à partir d'autres attributs.
5. **Normalisation des associations (importante)** : il faut éliminer les associations fantômes redondantes ou en plusieurs exemplaires .
6. Normalisation des cardinalités : une cardinalité minimale est toujours 0 ou 1 (et pas 2, 3 ou n) et une cardinalité maximale est toujours 1 ou n (et pas 2, 3, ...).

NIVEAU CONCEPTUEL

d- règles de normalisation : 1ère Forme Normale

A un instant donné dans une entité, pour un individu, un attribut ne peut prendre qu'une valeur et non pas, un ensemble ou une liste de valeurs. Si un attribut prend plusieurs valeurs, alors ces valeurs doivent faire l'objet d'une entité supplémentaire, en association avec la première.



hestim
Engineering & Business School

NIVEAU CONCEPTUEL

d- règles de normalisation : 2ième Forme Normale

L'identifiant peut être composé de plusieurs attributs mais les autres attributs de l'entité doivent dépendre de l ' identifiant en entier (et non pas une partie de cet identifiant).

NIVEAU CONCEPTUEL

d- règles de normalisation : 3ième Forme Normale

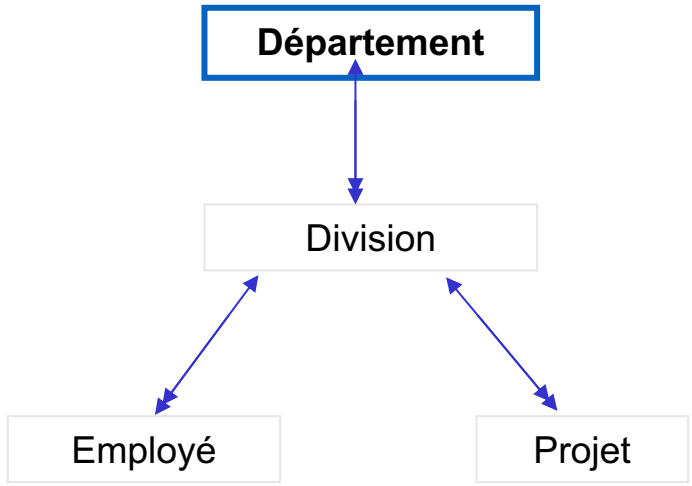
Modèle Hiérarchique

1- Définitions :

Un modèle de données hiérarchique : est un *ensemble de définitions d'arborescence*

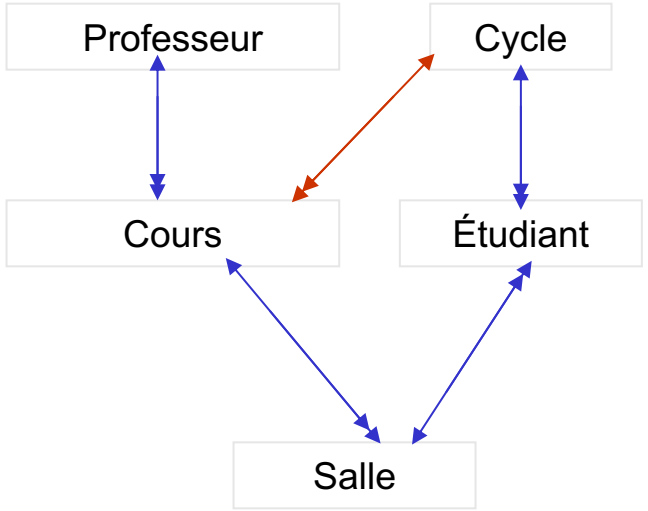
Une définition d'arborescence : est un *diagramme* de données dans lequel chaque entité, sauf la racine, a un *seul arc incident* de type (1:N) et *0, 1 ou plusieurs arcs émergeant* de type (1:N).

Exemple d'une définition d'arborescence :



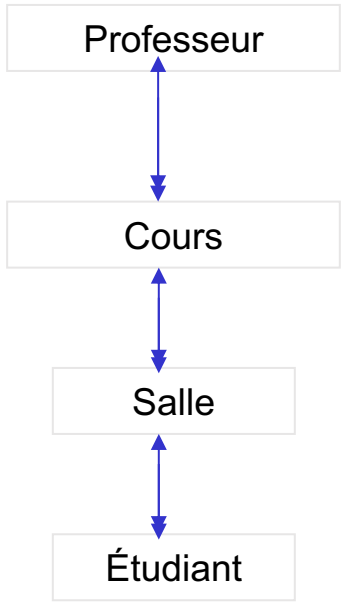
Modèle Hiérarchique

Exemple d'un modèle qui n'est pas hiérarchique :

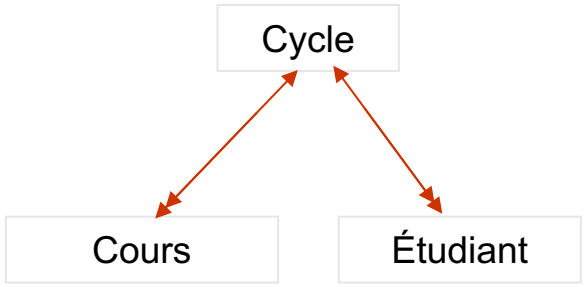


Modèle Hiérarchique

le modèle hiérarchique correspondant est constitué de 2 définitions d'arborescence :



1ère définition d'arborescence

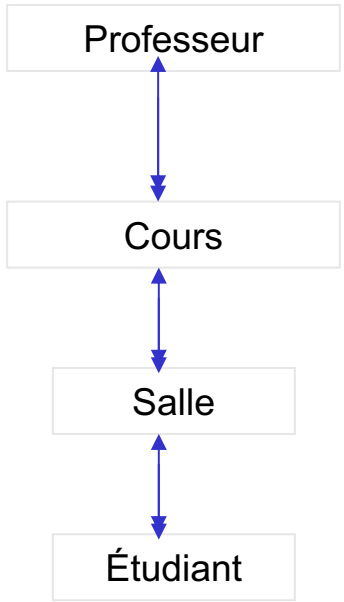


1éme définition d'arborescence

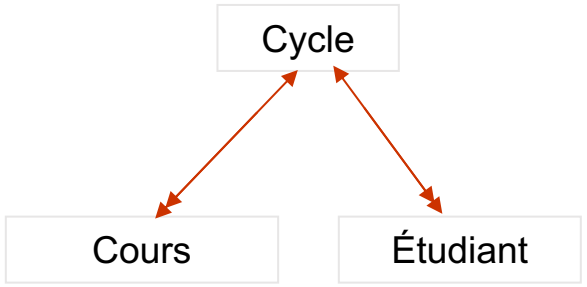
hestim
Engineering & Business School

Modèle Hiérarchique

le modèle hiérarchique correspondant est constitué de 2 définitions d'arborescence :



1ère définition d'arborescence



1éme définition d'arborescence

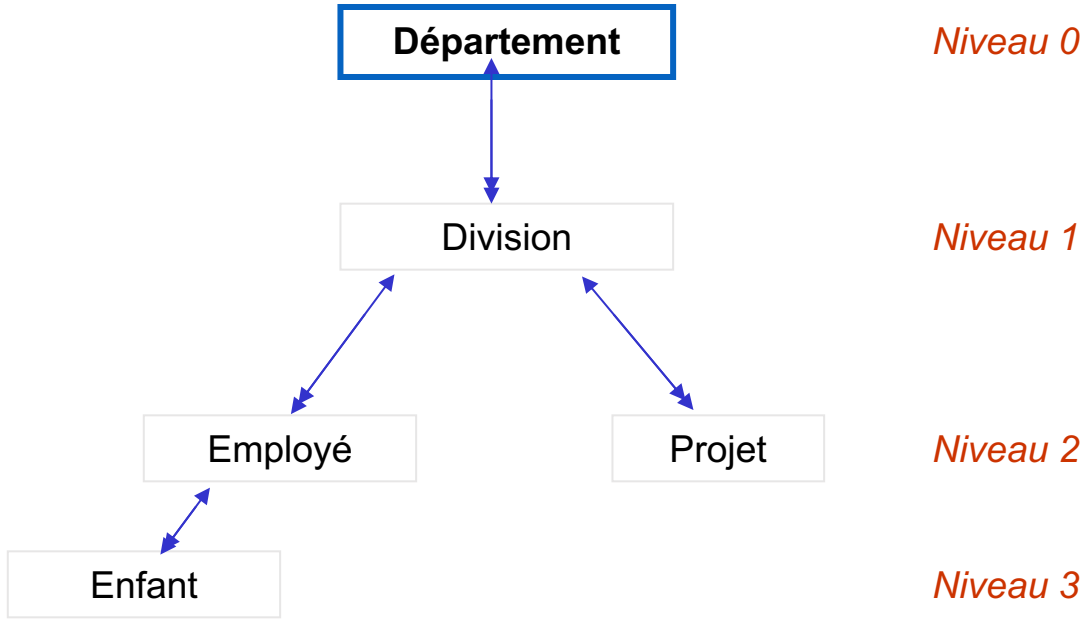
hestim
Engineering & Business School

Modèle Hiérarchique

2- Caractéristiques du modèle :

a- La relation Parent-Enfant : une occurrence d'entité d'un niveau i est dite parent, si elle est associée avec au moins une occurrence d'entité de niveau i+1

Exemple : soit le modèle hiérarchique suivant



Modèle Hiérarchique

2- Caractéristiques du modèle :

b- Élimination des occurrences d'entités : une occurrence d'une entité doit être reliée à une occurrence Parent et ainsi de nœud en nœud jusqu'à l'occurrence racine.

→ l'élimination d'une occurrence d'entité donnée provoque l'élimination de tous ses descendants

c- Les liens maillés dans le modèle hiérarchique : Le modèle hiérarchique ne peut représenter directement un lien (N : M) entre deux entités.

Solution : duplication des occurrences ou création d'une nouvelle entité permettant l'éclatement d'un lien (N : M) en deux lien (1 : N)

Modèle réseau

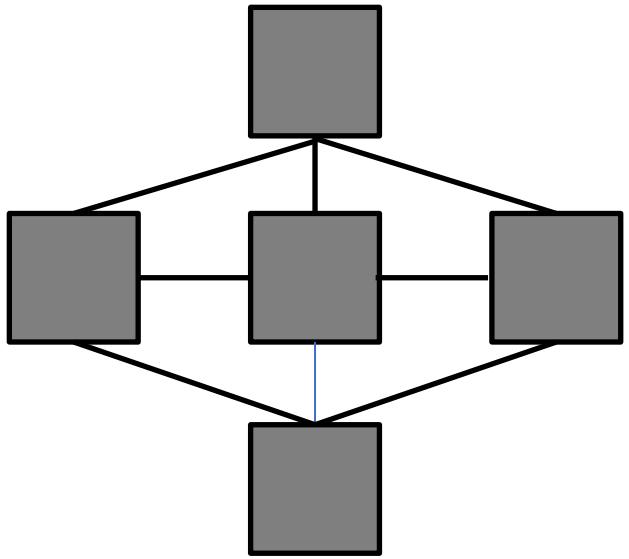
Définitions :

Un modèle de données réseau : est un *modèle hiérarchique dans laquelle les documents n'ont pas de relation parent-enfant stricte. Chaque enregistrement de données peut avoir plusieurs prédécesseurs.*

Une définition d'arborescence : est un *diagramme* de données dans lequel chaque Entité a un ou plusieurs *arcs incidents* et *0, 1 ou plusieurs arcs émergeant* de type (1:N).

On parle ici d'un modèle qui a la structure d'un réseau. Il n'existe pas de chemin d'accès unique à un enregistrement de données.

Exemple d'une définition d'arborescence :



hestim
Engineering & Business School

Modèle relationnel

1-Introduction :

Une base de données relationnelle : est une *base de donnée structuré suivant les principes de l'algèbre relationnelle.*

- *Edgard Frank Codd est le père des bases de données relationnelles*
- *En 1970 à travers un article, Edgar proposa de stocker des données hétérogènes dans des tables, permettant d'établir des relations entre elles. Son article ne connu pas un succès.*
- *Depuis les années 80, cette technologie a mûri et a été adoptée par l'industrie*
- *En 1987, le langage SQL, qui étend l'algèbre relationnelle, a été standardisé.*

Modèle relationnel

1- Introduction :

Une base de données relationnelle : De façon informelle, on peut définir le modèle relationnel de la manière suivante :

- les données sont organisées sous forme de tables à deux dimension, encore appelées relations, dont les lignes sont appelées n-uplet ou tuple en anglais;
- Les données sont manipulées par des opérateurs de l'algèbre relationnelle ;
- l'état cohérent de la base est défini par un ensemble de contraintes d'intégrité.

Modèle relationnel

2- Les concepts du modèle relationnel :

- ***attribut*** : Un attribut est un identificateur (un nom) décrivant une information stockée dans une base.

Exemples d'attribut : l'âge d'une personne, le nom d'une personne, le numéro de sécurité sociale.

- ***Domaine*** : Le domaine d'un attribut est l'ensemble, fini ou infini, de ses valeurs possibles.

Exemples de domaine: l'attribut numéro de sécurité sociale a pour domaine l'ensemble des combinaisons de quinze chiffres et nom a pour domaine l'ensemble des combinaisons de lettres

Modèle relationnel

2- Les concepts du modèle relationnel :

Exemple d'un schéma de relation :

CNE_etudiant	Nom	Prenom
1500000029	DODO	Mamane
1800000033	MANSOURI	Marouane
200000045	KONKOBO	Axel
1700000050	DURANT	Xavier

Relation de schéma Etudiant(CNE_étudiant, Nom, Prénom)

Modèle relationnel

2- Les concepts du modèle relationnel :

- ***Dégré*** : Le degré d'une relation est son nombre d'attributs.
- ***Occurrences ou n-uplets ou tuples*** : Une occurrence, ou n-uplets, ou tuples, est un élément de l'ensemble figuré par une relation. Autrement dit, une occurrence est une ligne du tableau qui représente la relation.
- ***Cardinalité*** : La cardinalité d'une relation est son nombre d'occurrences.

Modèle relationnel

2- Les concepts du modèle relationnel :

- ***Clé candidate***: Une clé candidate d'une relation est un ensemble minimal des attributs de la relation dont les valeurs identifient à coup sûr une occurrence.

La valeur d'une clé candidate est donc distincte pour tous les tuples de la relation. La notion de clé candidate est essentielle dans le modèle relationnel.

Règle : Toute relation a au moins une clé candidate et peut en avoir plusieurs.

Modèle relationnel

2- Les concepts du modèle relationnel :

- ***Clé primaire*** : La clé primaire d'une relation est une de ses clés candidates. Pour signaler la clé primaire, ses attributs sont généralement soulignés.
- ***Clé étrangère*** : Une clé étrangère dans une relation est formée d'un ou plusieurs attributs qui constituent une clé primaire dans une autre relation.
- ***Schéma relationnel*** : Un schéma relationnel est constitué par l'ensemble des schémas de relation.
- ***Base de données relationnelle*** : Une base de données relationnelle est constituée par l'ensemble des n-uplets des différentes relations du schéma relationnel.

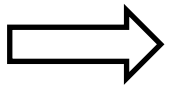
Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

- ***Règle1*** : Chaque entité donne naissance à une relation(table) Chaque attribut de l'entité devient un attribut de la relation. L'identifiant est conservé en tant que clé de la relation.

Etudiant
- CNE - Nom - Prénom - classe



Etudiant

CNE	Nom	Prénom	Classe

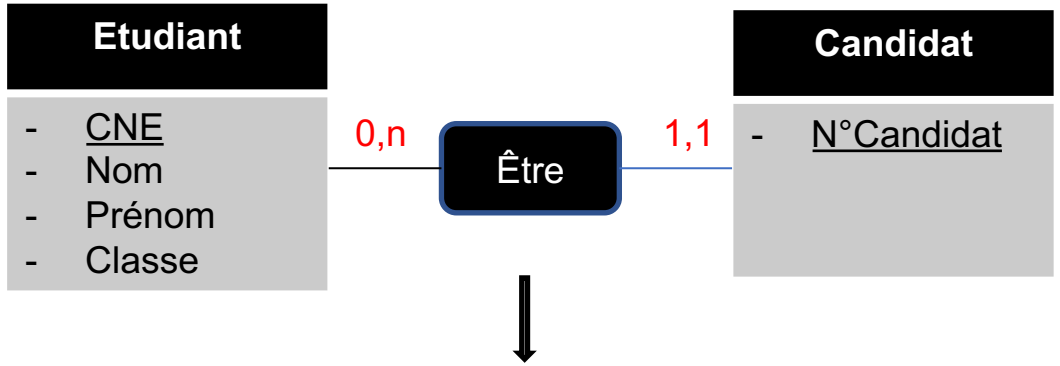
Etudiant(CNE, Nom, Prénom, Classe)

Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

- ***Règle 2 (Association binaire hiérarchique (x,1)->(x,n))***: Toute association hiérarchique (de type [1, n]) se traduit par une clé étrangère. La clé primaire correspondant à l'entité père (côté n) migre comme clé étrangère dans la relation correspondant à l'entité fils (côté 1).



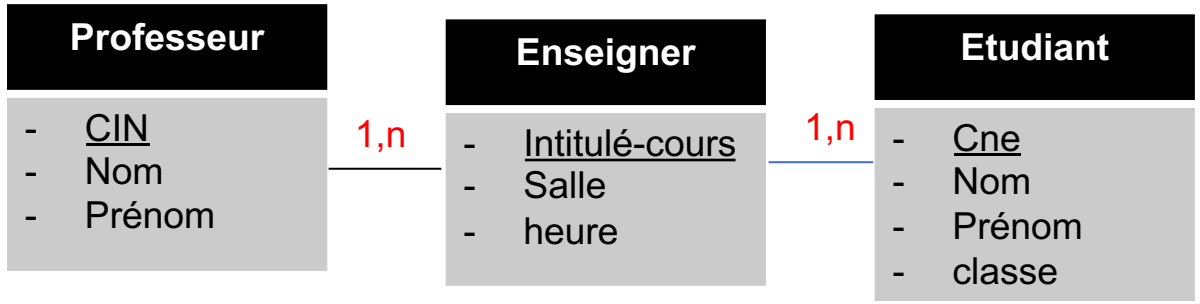
Etudiant(CNE, Nom, Prénom, Classe)
Candidat(N°Candidat, #CNE)

Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

- ***Règle 3(Association binaire hiérarchique (x,n)->(x,n))***: une association binaire de type n : m devient une table supplémentaire. La clé primaire est formée par la concaténation l'ensemble des identifiants des entités reliées



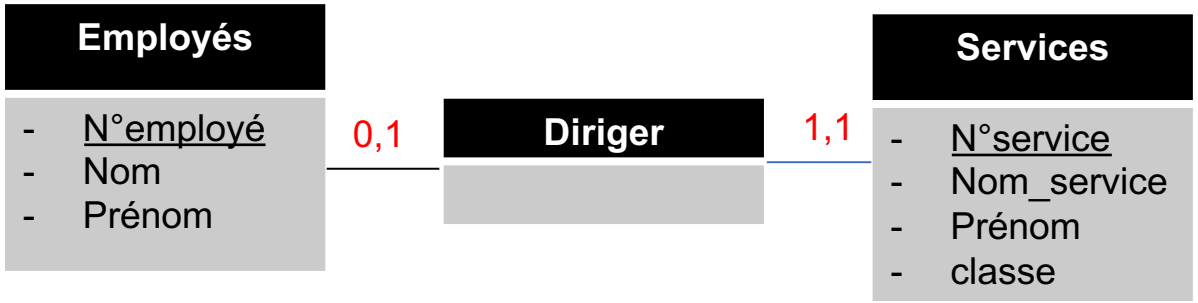
Professeur(CIN, Nom, Prénom)
Etudiant(cne, Nom, Prénom, classe)
Enseigner(#CIN, #cne, Intitulé-cours, salle, heure)

Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

- ***Règle 4(Association binaire hiérarchique (0,1)->(1,1))***: On duplique la clé de la table basée sur l'entité à cardinalité (0,1) dans la table basée sur l'entité à cardinalité (1,1).



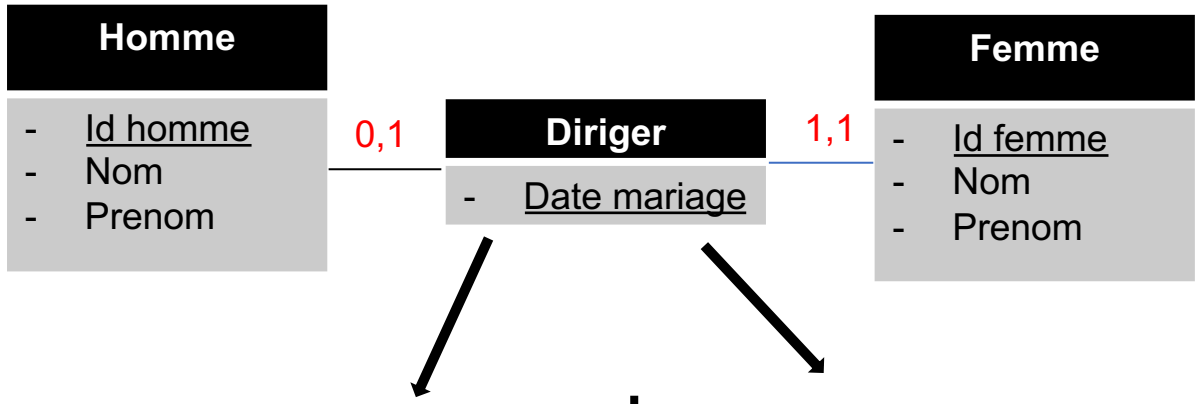
Employés(N°employé, Nom, Prénom)
Services(N°service, Nom_service, #N°employé)

Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

- ***Règle 4(Association binaire hiérarchique (0,1)->(0,1))***: On duplique la clé de la table basée sur l'entité à cardinalité (0,1) dans la table basée sur l'entité à cardinalité (1,1).



Homme(Id homme, Nom, Prenom)
Femme(Id femme, Nom, Prenom,
#Id Homme, #Date mariage)

Femme(Id femme, Nom, Prenom)
Homme(Id homme, Nom, Prenom,
#Id femme,)

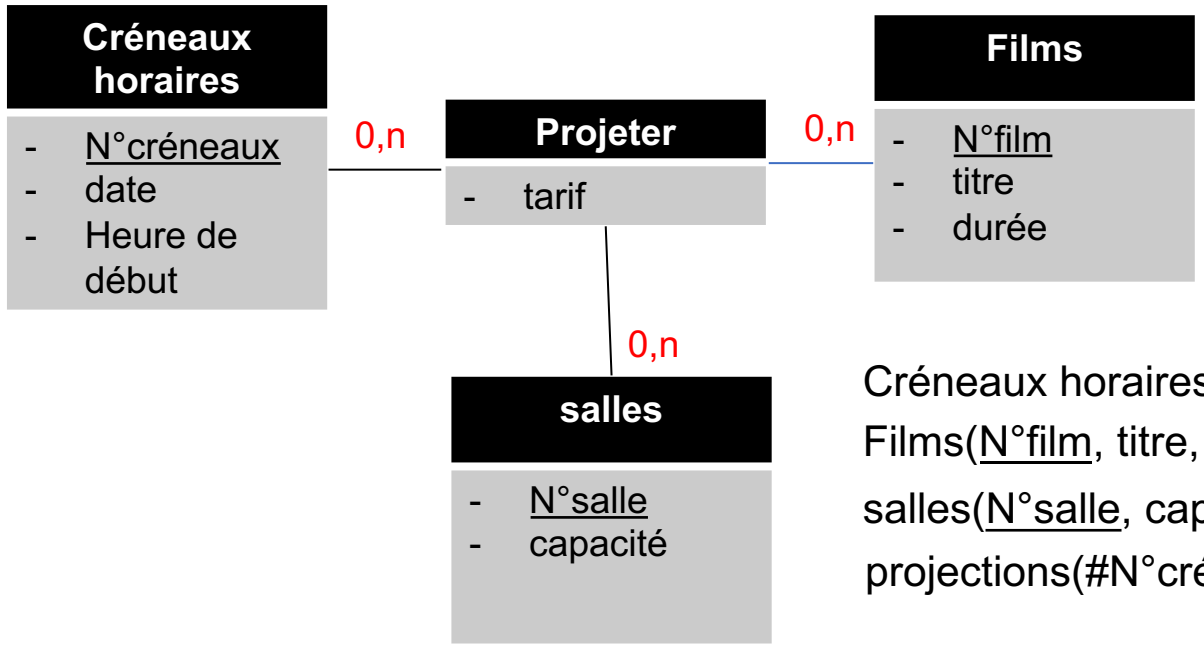
hestim
Engineering & Business School

Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

- ***Règle 4***: une association non binaire est traduite par une table supplémentaire dont la clé primaire est composée d'autant de clés étrangères que d'entités en association



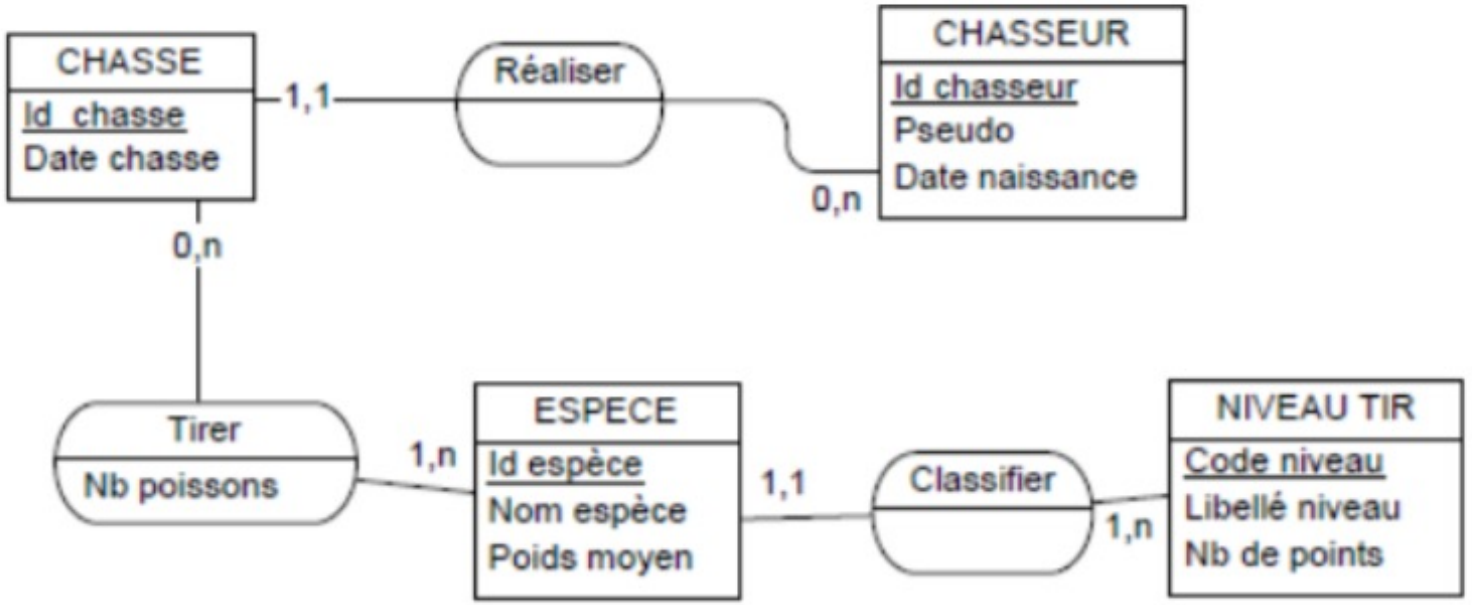
Créneaux horaires(N°créneaux, date, heure de début)
Films(N°film, titre, durée)
salles(N°salle, capacité)
projections(#N°créneaux, #N°film, #N°salle, capacité)

Modèle relationnel(MR)

2- Passage du Modèle E-A vers le MR :

2-1- Règles de passage :

Etablir le modèle relationnel du modèle MCD suivant :



SGBD

1- Définitions:

SGBD : **Un SGBD** est un logiciel qui permet à un utilisateur d'interagir avec une BD (stocker, accéder, mettre à jour des données).

SGBD

2-Fonctions d'un SGBD

Un SGBD permet principalement d'organiser les données sur les supports, et fournit les procédures de manipulation de ces mêmes données (recherche, sélection, mise à jour..)
Mais il a d'autres fonctions:

1. Description: le SGBD doit mettre à la disposition de l'utilisateur un outils pour décrire l'ensemble de données qui sont stockées dans la BD. Il y a différents niveaux de description de ces données:

-description logique: perception de la BD par l'utilisateur
-description physique: organisation des données sur les supports physiques

La description se fait par un langage de définition de données (**LDD**) propre à chaque SGBD

SGBD

2- Fonctions d'un SGBD

2. Utilisation: offrir à l'utilisateur une interaction avec la BD (rechercher, sélectionner et modifier) .

Il existe 2 façon pour interroger la BD:

- par programmes d'application (informaticiens)
- avec langage de requêtes (non informaticien)

Cette utilisation se fait par un langage de manipulation de données (**LMD**)

3. Intégrité: offrir à l'utilisateur la possibilité de définir des règles qui permettent de maintenir l'intégrité de la BD (règles ou contraintes d'intégrité: propriétés qui devront être toujours vérifiées).

Exemple: 0<Note<20 ou note = -1 (absent)

SGBD

2- Fonctions d'un SGBD

4. Confidentialité : puisqu'une BD est partagée par plusieurs utilisateurs, certains sous ensembles ne doivent être accessibles que par des personnes autorisées.
Le SGBD doit offrir des mécanismes permettant de vérifier les droits d'accès des utilisateurs (mots de passe, rôle..)
Exemple: …….

5. Synchronisation d'accès ou concurrence d'accès : lorsque les programmes d'application accèdent aux même informations au même temps, le SGBD doit offrir les mécanismes pour détecter les cas où il y aurait conflit d'accès et de les traiter correctement (problème de lecture fantôme…)

SGBD

3- Exemples de SGBD

- PostgreSQL: http://www.postgresql.org/
- MySQL : http://www.mysql.org/
- Oracle : http://www.oracle.com/
- IBM DB2 : http://www-306.ibm.com/software/data/db2/
- Microsoft SQL : http://www.microsoft.com/sql/
- Sybase : http://www.sybase.com/linux

RÔLES D’UNE BASE DE DONNÉES

hestim
Engineering & Business School

Pourquoi une BD ?

Exemple: Considérons une compagnie d'assurance vendant trois sortes d'assurances: Assurance vie, automobile et local)

-La compagnie est organisée en trois département, chacun a ses propres fichiers (traitement traditionnel). L'information est utilisée pour réaliser certain travaux.

Travail 1: gérer les assurances, Pour cela on a besoin :

-d'un fichier F1 des assurés : Numéro, nom, prénom, date de naissance, sexe, adresse..

-des programmes d'applications: P1→ calcul de prime

P2 → attestations

P3 → mise à jour

Pourquoi une BD ?

Exemple (suite):

Travail 2: gérer les sinistres, Pour cela on a besoin :

-d'un fichier F2 des sinistres : Numéro, date du sinistre, lieu, détail du sinistre..

-des programmes d'applications: P4→ Ajout d'un sinistre

P5 → mise à jour

Travail 3: règlement des sinistres, Pour cela on a besoin :

-des fichiers F1 et F2.

-d'un programme d'applications: P6→ calcul et production du règlement

Chaque département a son propre système d'information semblable à celui-ci

Pourquoi une BD ?

Exemple (suite):

Travail 2: gérer les sinistres, Pour cela on a besoin :

-d'un fichier F2 des sinistres : Numéro, date du sinistre, lieu, détail du sinistre..

-des programmes d'applications: P4→ Ajout d'un sinistre

P5 → mise à jour

Travail 3: règlement des sinistres, Pour cela on a besoin :

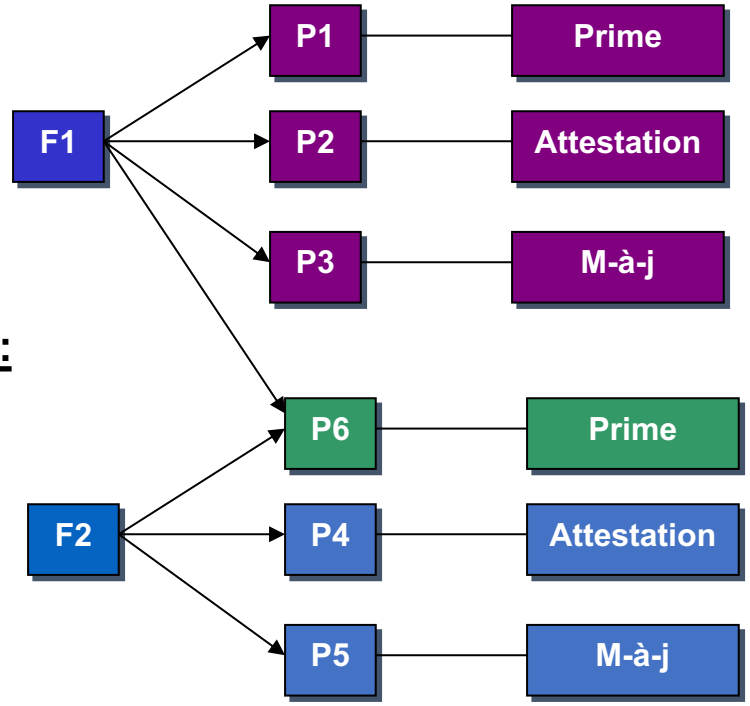
-des fichiers F1 et F2.

-d'un programme d'applications: P6→ calcul et production du règlement

Chaque département a son propre système d'information semblable à celui-ci

Pourquoi une BD ?

Récapitulons:



Département vie :

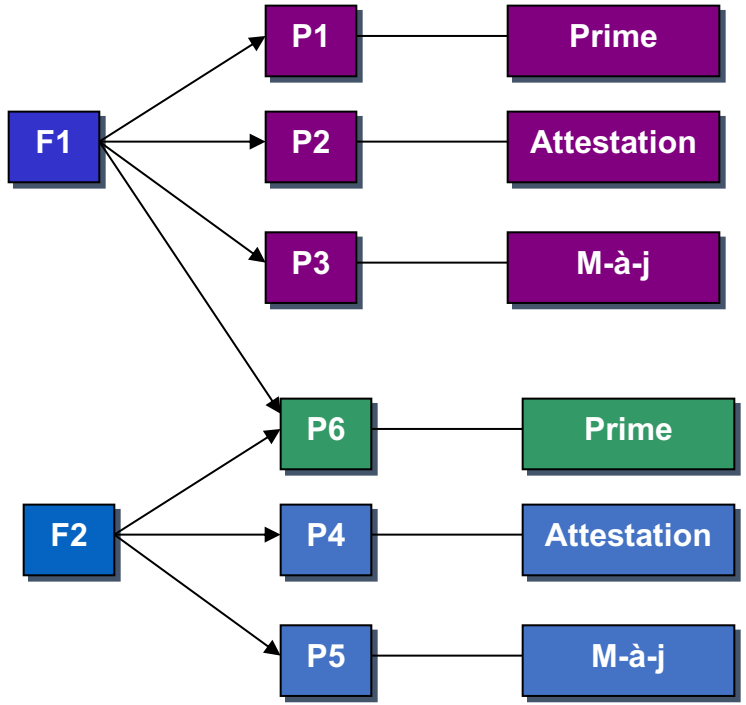
Chaque département a son propre système d'information semblable à celui-ci

Pourquoi une BD ?

Problèmes avec l'utilisation des fichiers

Problème 1: un adhérent X a souscrit l'assurance vie et auto. Son identifiant (nom, prénom, age, adresse…) va figurer dans deux fichiers

Défaut 1:
Redondance d'information → espace perdu



Pourquoi une BD ?

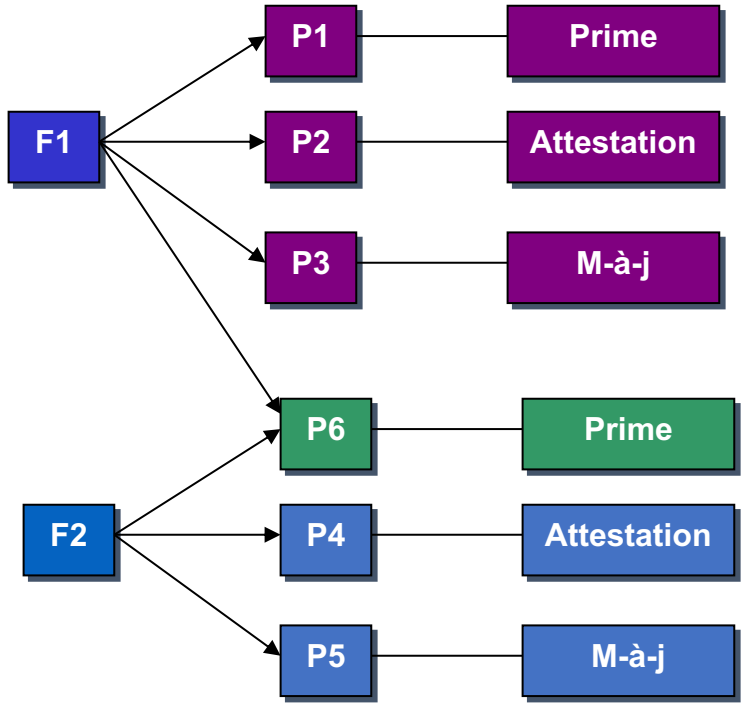
Problème 2: L'adhérent X déménage et envoie l'avis de changement d'adresse qui aboutit au département auto lequel met son fichier à jour mais ne transmet pas l'information au département vie

Défaut 2:

Redondance d'information → inconsistance d'information

ou

coût élevé de modification



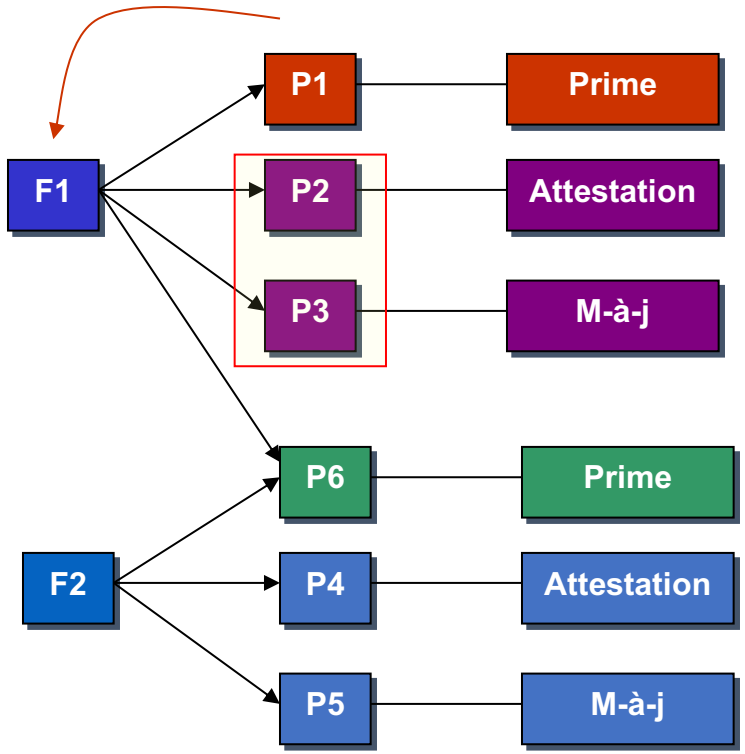
Pourquoi une BD ?

Problèmes avec l'utilisation des fichiers

Problème 3: un changement dans le calcul des primes exige une nouvelle donnée dans F1 (la profession par exemple). Les programmes P2 et P3 bien que non concernés par cette nouvelle information sont à corriger, de plus le fichier F1 est à refaire

Défaut 3:

les programmes sont dépendant des données

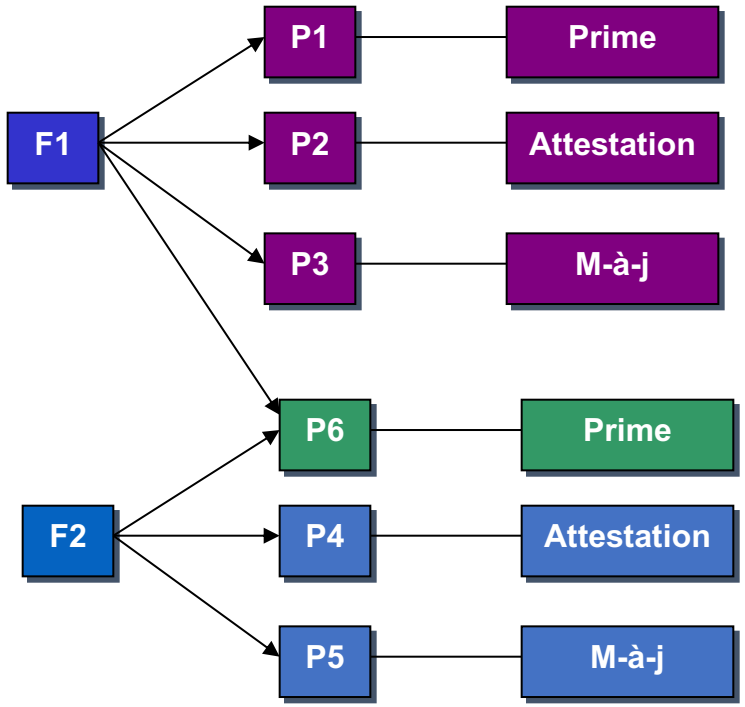


Pourquoi une BD ?

Problème 4: on veut savoir combien de femmes de 25 à 40 ans ont souscrit une assurance vie. Il faut alors écrire tout un programme pour le savoir

Défaut 4:

les données sont accessibles seulement à travers les programmes d'application

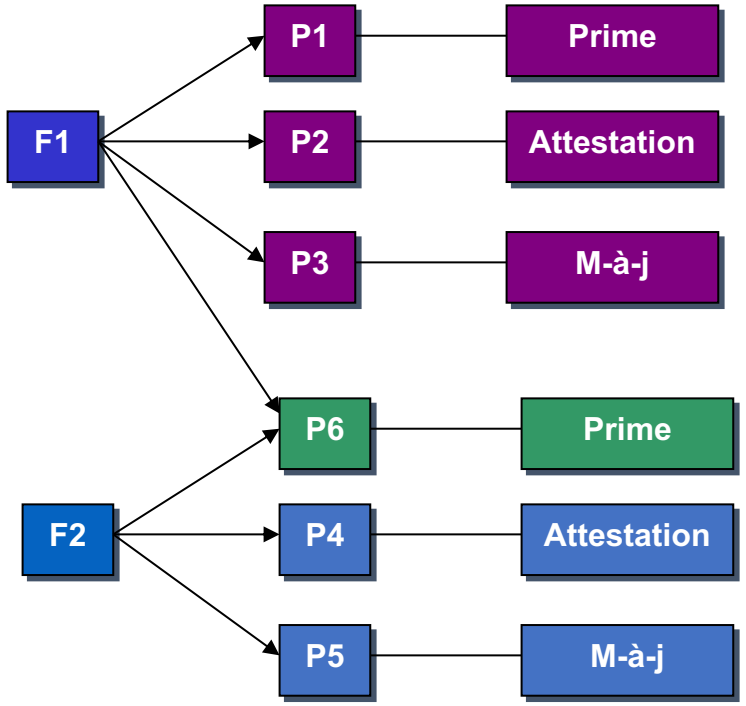


Pourquoi une BD ?

Problème 4: on veut savoir combien de femmes de 25 à 40 ans ont souscrit une assurance vie. Il faut alors écrire tout un programme pour le savoir

Défaut 4:

les données sont accessibles seulement à travers les programmes d'application



Pourquoi une BD ?

Récapitulatif → objectifs des BD

Défauts avec les fichiers	Objectifs des BD
Redondance d'information	Unicité de l'information
Dépendance des données à l'égard des programmes	Indépendance des programmes d'applications
Données accessibles à travers les programmes d'applications	Accès par : programmes et langages des requêtes
Données dispersées et incohérentes	Intégrité des données
Chacun a ses propres fichiers	Partage des données par plusieurs utilisateurs

DES SYSTÈMES DE GESTION DE FICHIERS AUX SGBD

hestim
Engineering & Business School

hestim
Engineering & Business School

LANGAGES DE MANIPULATION DES DONNÉES

hestim
Engineering & Business School

OPERATIONS ALGEBRIQUES APPLIQUEES AUX TABLES

- **UNAIRES (Un opérande : sélection, projection) :** ce sont les opérateurs les plus simples, ils permettent de produire une nouvelle table à partir d'une autre table
- **BINAIRES (Deux opérandes : Union, Intersection, différences) :** ces opérateurs permettent de produire une nouvelle relation à partir de deux relations de même degré et de même domaine.
- **BINAIRES ou N-AIRES(Produit cartésien, jointure, division) :** ils permettent de produire une nouvelle table à partir de deux ou plusieurs autres tables.

5 OPERATIONS ALGEBRIQUES ELEMENTAIRES

–RESTRICTION

–PROJECTION

–PRODUIT

–DIFFERENCE

–UNION

OPERATION COMPLEMENTAIRE (l'une des plus utilisées)

–JOINTURE

hestim
Engineering & Business School

1- Sélection ou restriction

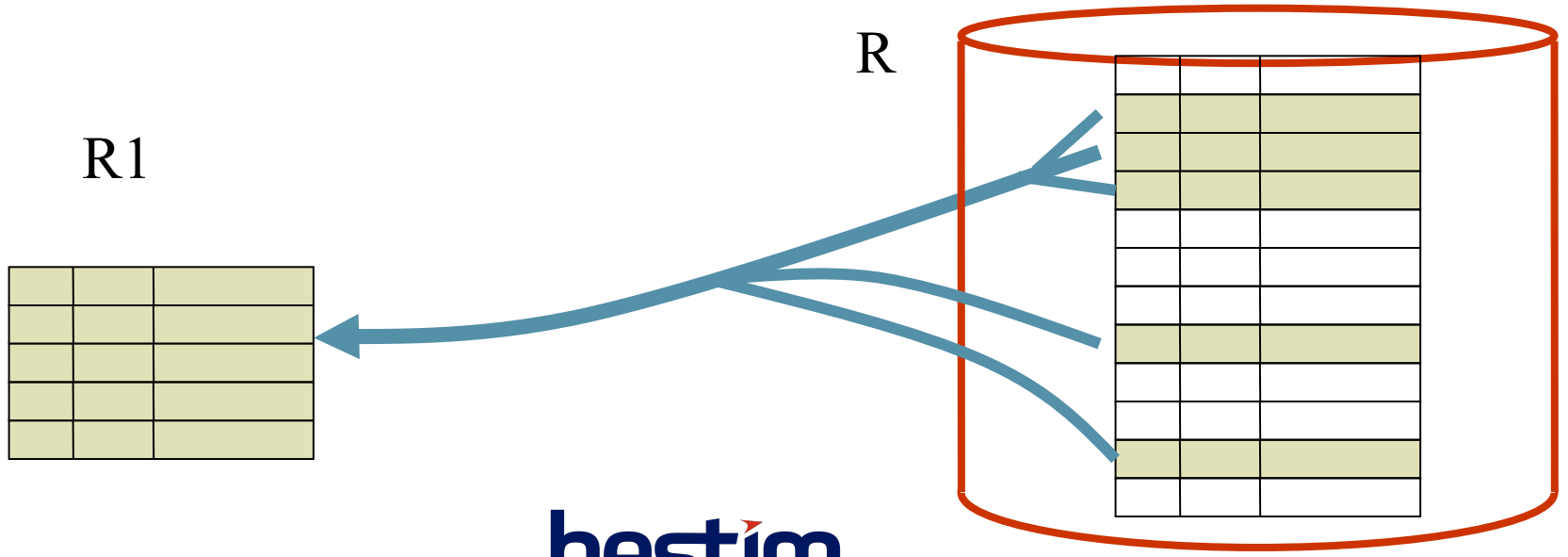
R1 = RESTRICTION (R; <condition_de_restriction>)

condition_simple : opérande1 □ opérande2. □ est un opérateur à prendre parmi {=, □, >, <, >=, <=}

condition_composée : conditions simples liées par les opérateurs logiques **et**, **ou**, **non**

R1 contient les lignes de R qui vérifient la condition de restriction

$$R1 = \square_{condition}\ (R)$$



1- Sélection ou restriction

Livre

N°Livre	TitreLivre
10	La monnaie
25	La finance

LivreAuteur

N°Livre	N°Auteur
10	1
25	1
25	3

Auteur

N°Auteur	NomAuteur
1	Dupont
2	Durand
3	Martin

1- Sélection ou restriction

- Soit la table livre(N°Livre, Titre, Année)

  – On veut savoir quels sont les livres qui sont sortis en 2000

  – R1=Restriction(Livre; Année=2000)

  – R1= $\sigma_{Année=2000}$(Livre)

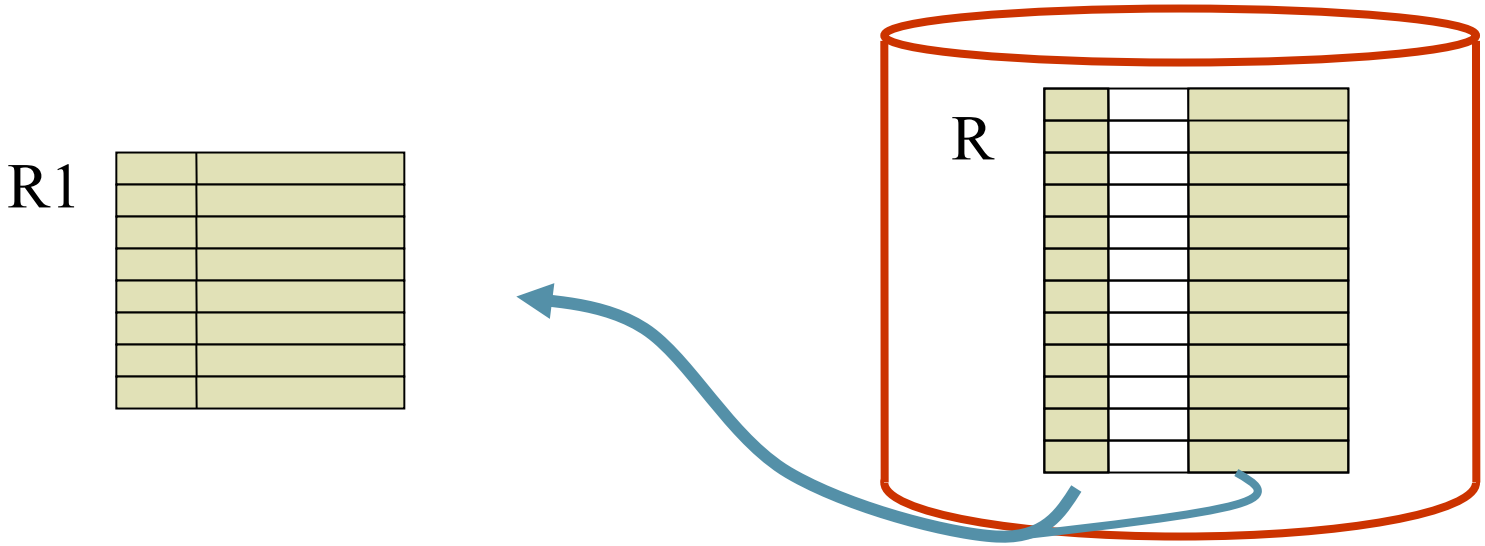
2- Projection

R1 = PROJECTION (R; <liste_attributs_projection>)

<liste_attributs_projection> : Attributs Ai, avec Ai □ R

R1 est une table qui contient les lignes de R où on ne garde que les champs spécifiés dans (liste_attributs_projection)

R1 = □ $_{\text{Liste des attributs}}$ (R)



2- Projection

- Soit la table Livre(N°Livre, Titre, Année)

  – On veut savoir quels sont les titres des livres

  – R1=Projection(Livre; Titre)

  – **R1=** Π_{Titre} **(**Livre)

3- Union

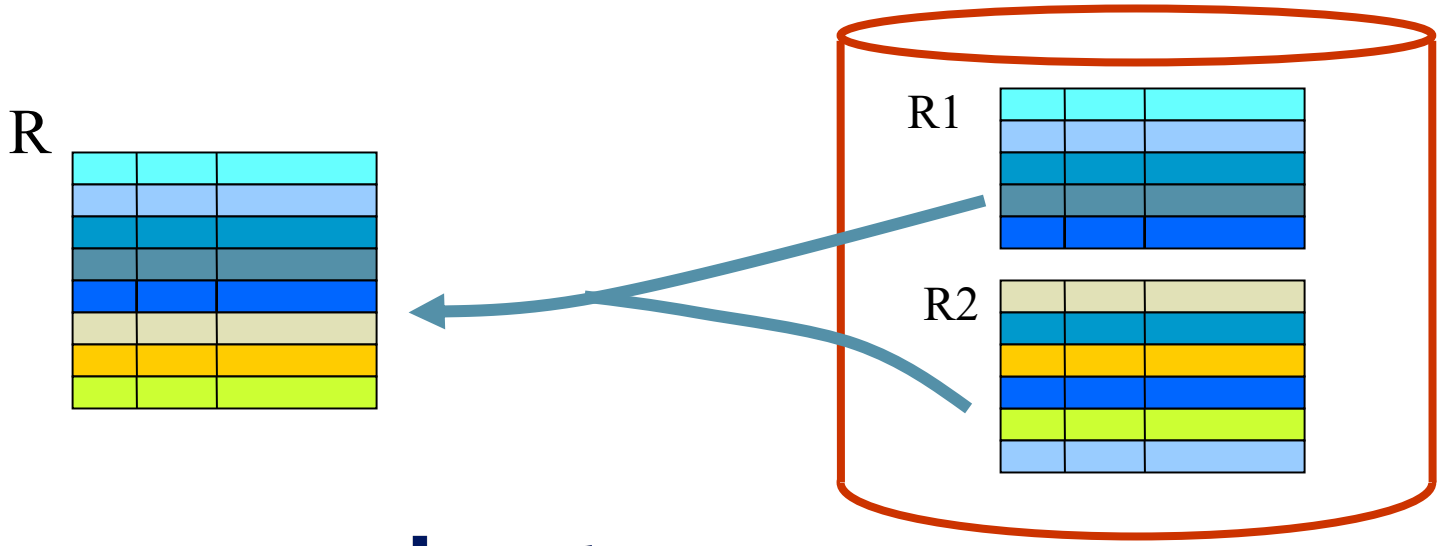
R = UNION (R1 ; R2)

R contient les lignes de R1 et celles de R2

Les doublons sont éliminés.

R1 et R2 doivent avoir les mêmes champs.

R = R1 □ R2



3- Union

Soit les tables:
Etudiant(NSS, Nom, Adresse)
Employé(NSS, Nom, Adresse)

On veut avoir la liste des personnes qui accèdent à l'université (étudiants ou employés)

Union(Etudiant; Employé)

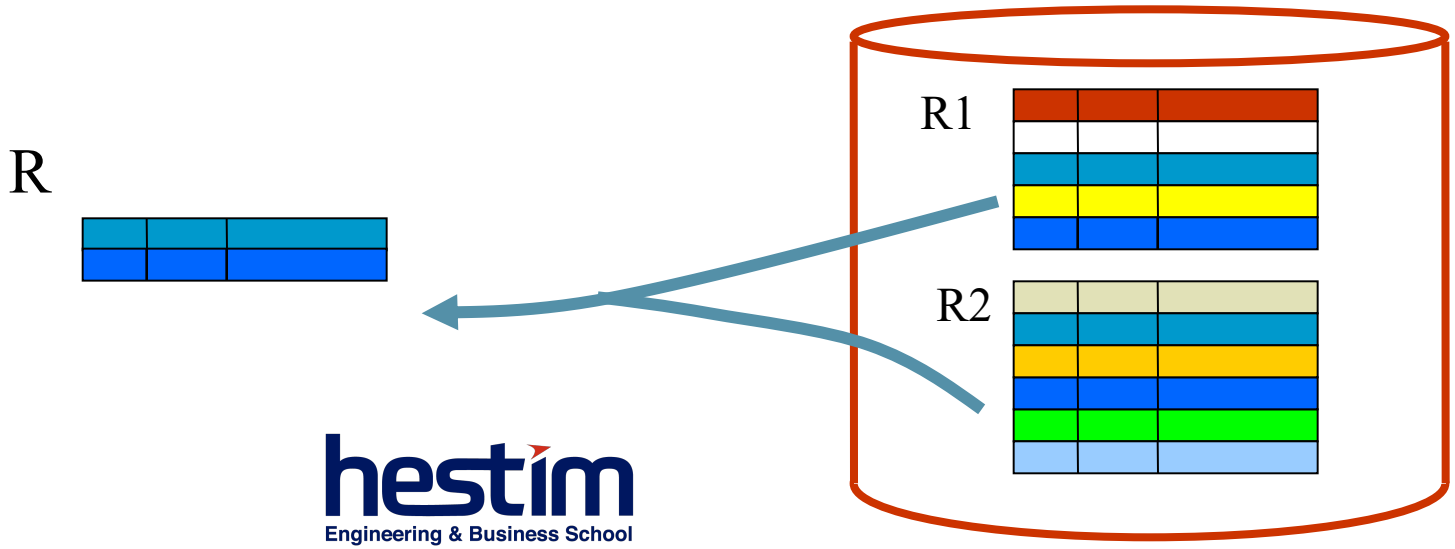
4- Intersection

R = Intersection(R1; R2)

R contient les lignes de R1 qui sont dans R2

R1 et R2 doivent avoir les mêmes champs.

R = R1 □ R2



hestim
Engineering & Business School

4- Intersection

- Soit les tables:
  - Etudiant(NSS, Nom, Adresse)
  - Employé(NSS, Nom, Adresse)

- On veut avoir la liste des personnes qui sont en même temps étudiantes et employées à l'université

- Intersection(Etudiant ; Employé)

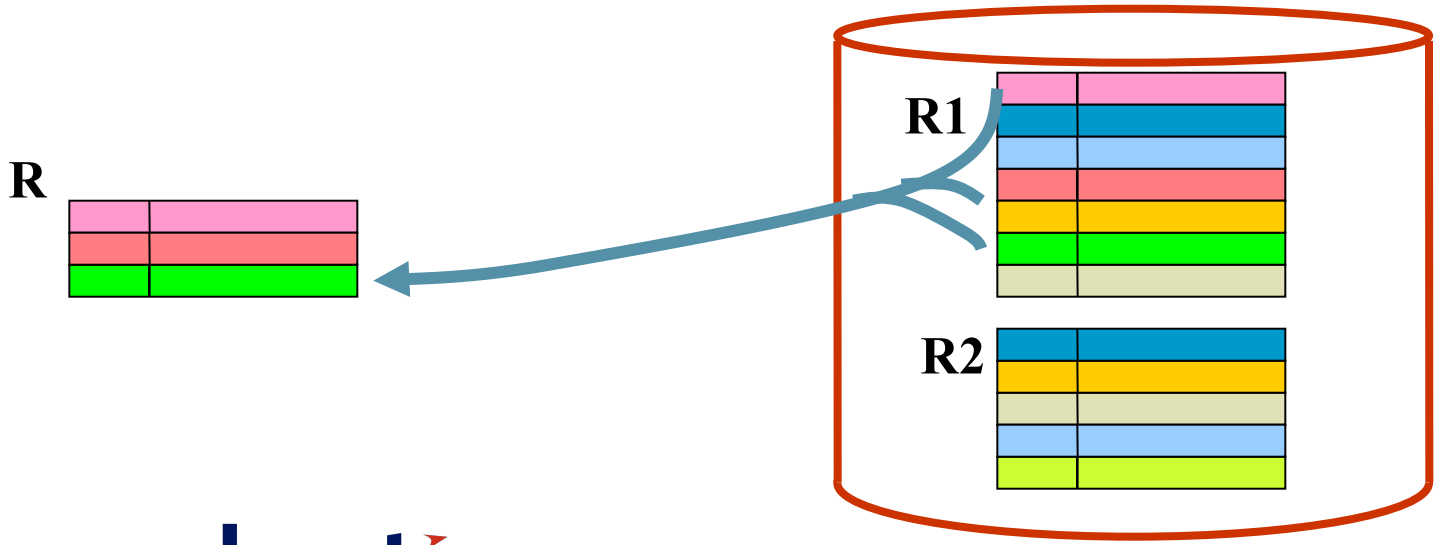
5- Différence

R = DIFFERENCE (R1; R2)

R contient les lignes de R1 qui ne sont pas dans R2.

R1et R2 doivent avoir le même schéma.

R =R1 - R2



5- Différence

- Soit les tables:
  - Etudiant(NSS, Nom, Adresse)
  - Employé(NSS, Nom, Adresse)

- On veut avoir la liste des personnes qui sont employées à l'université et qui ne sont pas en même temps des étudiants

- Différence(Employé; Etudiant)

6- Produit cartésien

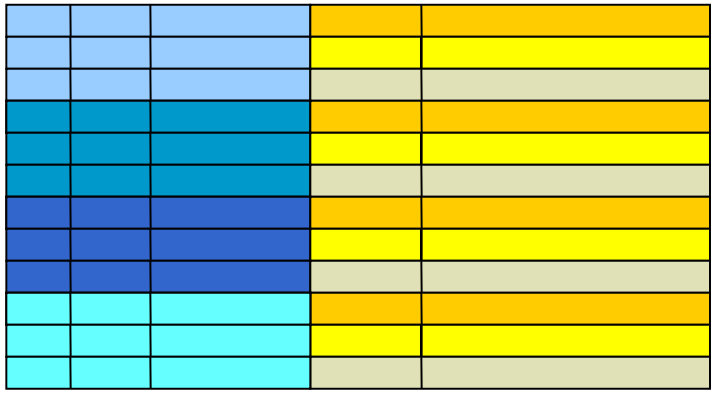
R = PRODUIT (R1;R2)

R contient le produit cartésien de R1 et R2, i.e toutes les combinaisons des lignes de R1 avec les lignes de R2

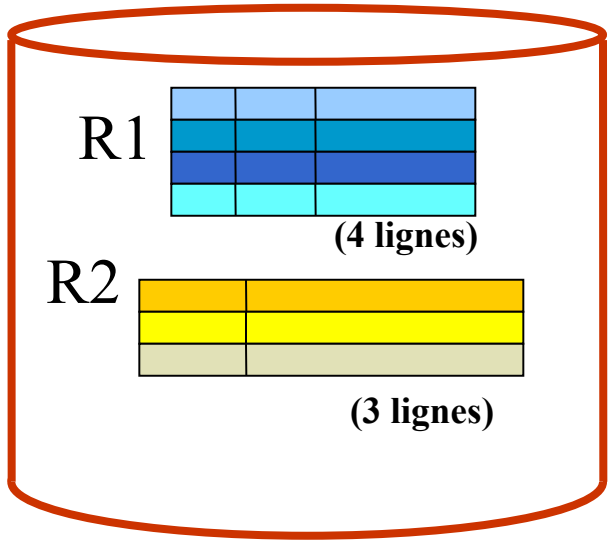
Les colonnes de R sont celles de R1 AINSI que celles de R2

R =R1 * R2

R



(3x4 = 12 lignes)



6- Produit cartésien

Soient les tables:

Livre(N°Livre, Titre, Année) contient *n* lignes

Auteur(N°Auteur, Nom, AnnéeNaissance) *m* lignes

On veut savoir quels sont les affectations possibles qu'on peut réaliser entre les livres et les auteurs

R1=Produit(Livre; Auteur)

R1(N°Livre, Titre, Année, N°Auteur, Nom, AnnéeNaissance)

R1 contient *n***m* lignes

6- Produit cartésien

Exemple combinant produit, restriction et projection

Soient les tables:

Livre(N°Livre, Titre, Année)

Auteurs(N°Auteur, Nom, AnnéeNaissance)

On veut avoir les couples de la forme (N°Livre, N°Auteur) où l'année de sortie du livre N°Livre correspond à l'année de naissance de l'auteur N°Auteur

R1=Produit(Livre; Auteur)

R2=Restriction(R1; Année=AnnéeNaissance)

R3=Projection(R2; N°Livre, N°Auteur)

6- Produit cartésien

Produit : Cas particulier

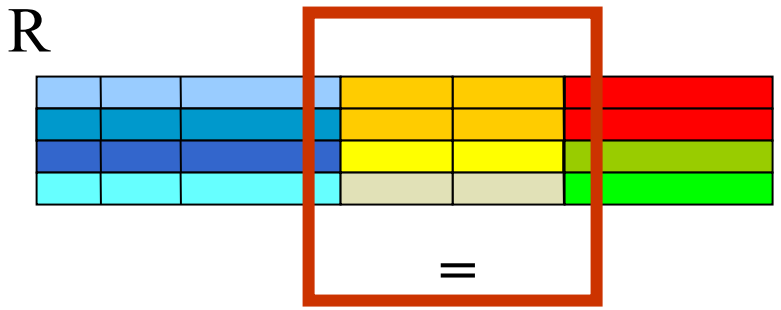
- Que se passe-t-il si les deux tables ont des champs en commun ?
  - Ex: R(A, B) et S(B, C)
  - R * S donne une Table qui a 4 champs :
    - A, R.B, S.B, C
  - Les champs communs sont précédés du nom de la table d'où ils proviennent

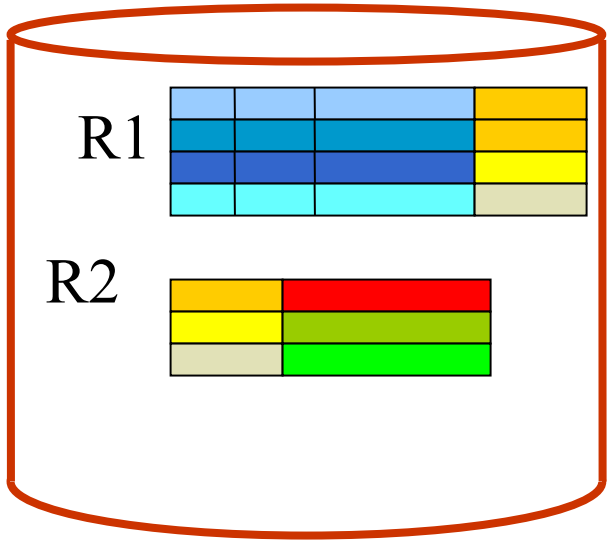
7- Jointure

R = JOINTURE (R1; R2)

R contient les combinaisons des lignes de R1 avec les lignes de R2 qui ont la même valeur sur les champs communs

R =R1 ⊗ R2





7- Jointure

Exemple de jointure

- Soit les tables
  - Livre(N°Livre, Titre, Année, N°Auteur)
  - Auteur(N°Auteur, Nom, AnnéeNaissance)
- On veut savoir pour chaque livre, l'auteur qui l'a écrit (on suppose qu'il n'y a qu'un auteur par livre)
- Jointure(Livre; Auteur)

SQL: les parties (les 2 grandes parties)

- Data-definition language (DDL): fournit des commandes pour définir des schémas de relation, supprimer des relations et modifier des schémas de relation.
- Data-manipulation language (DML): offre la possibilité d'interroger les informations de la base de données et d'insérer des tuples, de supprimer des tuples et de modifier des tuples dans la base de données.

SQL: les parties (sous parties)

- Integrity: DDL SQL comprend des commandes permettant de spécifier les contraintes d'intégrité auxquelles les données stockées dans la base de données doivent satisfaire.
  - Astuce: Les mises à jour qui violent les contraintes d'intégrité sont interdites.
- View definition. Le DDL SQL inclut des commandes pour la définition des vues.
- Transaction control. SQL inclut des commandes pour spécifier les points de début et de fin des transactions.
- Embedded SQL and dynamic SQL. Le langage SQL intégré et dynamique définit la façon dont les instructions SQL peuvent être intégrées dans des langages de programmation à usage général, tels que C, C ++ et Java.
- Authorization. Le DDL SQL comprend des commandes pour spécifier les droits d'accès aux relations et aux vues.

SQL DDL - Data Definition Language

SQL DDL pour la BD university

department(dept_name, building, budget)
course(course_id, title, dept name, credits)
instructor(ID, name, dept name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)

DDL

- SQL DDL permet de spécifier non seulement un ensemble de relations, mais également des informations sur chaque relation, notamment:
  - ✓ Le schéma de chaque relation.
  - ✓ Les types de valeurs associés à chaque attribut.
  - ✓ Les contraintes d'intégrité.
  - ✓ L'ensemble d'indices à maintenir pour chaque relation.
  - ✓ Les informations de sécurité et d'autorisation pour chaque relation.
  - ✓ La structure de stockage physique de chaque relation sur le disque.

DDL – les types de base (atomique) par le standard SQL

- char(n): une chaîne de caractères de longueur fixe n.
- varchar(n): une chaîne de caractères de longueur variable avec une longueur maximale n.
- int: un entier (un sous-ensemble fini d'entiers qui dépend de la machine).
- smallint: un petit entier (un sous-ensemble dépendant de la machine de type entier).
- numeric(p, d) : Un nombre à virgule fixe avec une précision spécifiée par l'utilisateur. Le nombre est composé de p chiffres (plus un signe), et d des p chiffres se trouvent à droite de la virgule décimale. **numeric((3,1)** permet de stocker exactement 44,5, mais ni 444,5 ni 0,32 ne peuvent être stockés exactement dans un domaine de ce type.
- real, double precision: nombres à virgule flottante et double précision à virgule flottante avec précision dépendante de la machine.
- float(n) : nombre à virgule flottante avec une précision d'au moins n chiffres.

CREATE TABLE

- Syntaxe

```
CREATE TABLE r
(A1 D1,
 A2 D2,
 ...,
 An Dn,
⟨integrity-constraint1⟩,
…,
⟨integrity-
constraintk⟩);
```

```
CREATE TABLE department
(dept_name      varchar (20),
 building  varchar (15),
 budget    numeric (12,2),
primary key (dept_name));
```

CREATE TABLE

- Syntaxe

```
CREATE TABLE r
(A1 D1,
 A2 D2,
 ...,
 An Dn,
⟨integrity-constraint1⟩,
…,
⟨integrity-
constraintk⟩);
```

```
CREATE TABLE department
(dept_name      varchar (20),
 building  varchar (15),
 budget    numeric (12,2),
primary key (dept_name));
```

CREATE TABLE

- utilisée pour créer les tables de base qui forment le cœur d'une base de données relationnelle.
- utilisé à tout moment pendant le cycle de vie du système,
  - le développeur de la base de données peut commencer avec un petit nombre de tables et les ajouter à mesure que des applications supplémentaires.
- Comme dans le modèle relationnel abstrait, les lignes sont considérées comme non ordonnées.
- Cependant, les colonnes sont ordonnées de gauche à droite, pour correspondre à l'ordre des définitions de colonnes dans la commande CREATE TABLE.

CREATE TABLE et contraintes d intégrité

- primary key (Aj1, Aj2,…, Ajm): les attributs Aj1, Aj2,…, Ajm forment la clé primaire de la relation.
  - Les attributs de clé primaire doivent être non nuls et uniques (nonnull and unique);

- foreign key (Ak1, Ak2,…, Akn) references s: les valeurs des attributs (Ak1, Ak2,…, Akn) pour tout tuple dans la relation doit correspondre aux valeurs des attributs de clé primaire de certains tuple dans la relation s.

CREATE TABLE et contraintes d intégrité

- table *course* : "foreign key (dept_name) references department"
  - pour chaque tuple de course, le nom de département spécifié dans le tuple doit exister dans l'attribut de clé primaire (dept_name) de la relation de département.
- Q: Si cette contrainte n"existe pas! Résultat?
  - un cours dans un département inexistant.
- Q: Discuter les autres contraintes PK FK de l"exemple.
- Remarque: Certains SGBD, y compris MySQL, nécessitent une syntaxe alternative, "foreign key (dept_name) references department(dept_name)", (où les attributs référencés dans le référencé sont répertoriés explicitement).

CREATE TABLE et contraintes d intégrité

- not null: cette contrainte pour un attribut spécifie que la valeur nulle n'est pas autorisée pour cet attribut, la contrainte exclut la valeur nulle du domaine de cet attribut.

Contraintes d'intégrité et violation !

- Important: SQL empêche toute mise à jour de la base de données qui viole une contrainte d'intégrité.
- SQL signale une erreur et empêche la mise à jour.
- Donner quelques exemples de violation pour la BD university.

- Une relation nouvellement créée est initialement vide.
- L'insertion, la mise à jour et la suppression des tuples dans une relation, leur mise à jour et leur suppression sont effectuées par les instructions de manipulation de données **insert, update et delete**

Contraintes d'intégrité

- Pour supprimer une relation d'une base de données SQL:

  drop table r;

- supprime toutes les informations sur la relation supprimée de la BD.
- dropt table r; est une action plus drastique que delete from r;
- delete from r; : conserve la relation r, mais supprime tous les tuples.
- drop table r; supprime tous les tuples de r et le schéma de r.

ALTER TABLE

- ALTER TABLE pour ajouter des attributs à une relation existante. Tous les tuples de la relation ont la valeur null comme valeur pour le nouvel attribut:

alter table r add A D;

  - où A: nom de l'attribut à ajouter et D: le type de l'attribut ajouté.

- Nous pouvons supprimer des attributs d'une relation par la commande:

alter table r drop A;

- La structure de base d'une requête SQL se compose de trois clauses: select, from et where.

- Une requête prend en entrée:
  - les relations répertoriées dans la clause from,
  - les opère comme spécifié dans les clauses where et select,
- puis produit une relation comme résultat.

- Requêtes sur une seule relation
  - Requête 1: "Trouvez les noms de tous les instructeurs."

```
select name from instructor;
```

  - Requête 2: «Trouver les noms de département de tous les instructeurs», qui peut être écrite comme suit:

```
select dept name from instructor;
```

    - un nom de département peut apparaître plusieurs fois.

- Dans la définition formelle et mathématique du modèle relationnel, une relation est un ensemble. Ainsi, les tuples en double n'apparaîtraient jamais dans les relations.

- Pour forcer l'élimination des doublons: mot-clé distinct

```
select distinct dept name from
instructor;
```

- Requêtes sur une seule relation
  - mot clé all: pour spécifier explicitement que les doublons ne sont pas supprimés:

```
select all dept name from instructor;
```

- La clause select peut également contenir des expressions arithmétiques impliquant les opérateurs +, -, ∗ et / opérant sur des constantes ou des attributs de tuples. Par exemple, la requête:

```
select ID, name, dept_name, salary
  *1.1 from instructor;
```

□10% d'augmentation .....

- Requêtes sur une seule relation (fin)
- La clause where pour satisfaire un prédicat spécifié.
  - Requête 3: "Trouvez les noms de tous les instructeurs du département d'informatique qui ont un salaire supérieur à 70 000 $."

```
select  name
from    instructor
where     dept_name = 'Comp. Sci.' and
          salary >
 70000;.
```

- SQL permet l'utilisation des connecteurs logiques AND, OR, et NOT dans la clause where.
- Les opérandes des connecteurs logiques peuvent être des expressions impliquant les opérateurs de comparaison <, <=,>,> =, = et <>.
- SQL nous permet d'utiliser les opérateurs de pour comparer des chaînes et des expressions arithmétiques, acomparaisoninsi que des types spéciaux, tels que les types de date.

- Requêtes sur les relations multiples
  - Requête 4: "Récupérer les noms de tous les instructeurs, ainsi que leurs noms de département et le nom du bâtiment de département."
  - En SQL, pour répondre à des requêtes de ce genre: répertorer les relations auxquelles il faut accéder dans la clause from et spécifions la condition de correspondance dans la clause where. (jointure AR)

    select name, instructor.dept_name, building from instructor, department

    where instructor.dept_name= department.dept_name;.
  - Notez que l'attribut dept_name dans les 2 relations.
  - Utiliser le nom de relation comme comme un préfixe (dans le nom de l'instructeur.dept_name et department.dept_name) pour indiquer clairement à quel attribut nous nous référons.

- Requêtes sur les relations multiples
  - Requête 4: "Récupérer les noms de tous les instructeurs, ainsi que leurs noms de département et le nom du bâtiment de département."
  - En SQL, pour répondre à des requêtes de ce genre: répertorer les relations auxquelles il faut accéder dans la clause from et spécifions la condition de correspondance dans la clause where. (jointure AR)

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name=
department.dept_name;.
```

- Notez que l'attribut dept_name existe dans les 2 relations.
- Utiliser le nom de relation comme comme un préfixe (dans le nom de l'instructeur.dept_name et department.dept_name) pour indiquer clairement à quel attribut nous nous référons. enlever l'ambigüité

- Requêtes sur les relations multiples
  - Syntaxe de SELECT ... FROM... WHERE ...

```
select A1, A2,…, An
from r1, r2,…, rm
where P;
```

- Requêtes sur les relations multiples
  - En général, pour un SGBD, la signification d'une requête SQL peut être comprise comme suit:

1. Générez un produit cartésien des relations répertoriées dans la clause from.
2. Appliquez les prédicats spécifiés dans la clause where sur le résultat de l'étape 1.
3. Pour chaque tuple du résultat de l'étape 2, sortez les attributs (ou résultats d'expressions) spécifiés dans la clause select.

Ordonner l'affichage des tuples

- SQL offre à l'utilisateur un certain contrôle sur l'ordre dans lequel les tuples d'une relation sont affichés.
- clause order by:
  - fait apparaître les tuples dans le résultat d'une requête dans l'ordre trié.
  - Requête 8: « Pour lister par ordre alphabétique tou les instructeurs du département de physique »

```
select name
from   instructor
where  dept_name = 'Physics'
order by     name;
```

Ordonner l'affichage des tuples

- `order by` par défaut répertorie les éléments dans l'ordre **croissant**.
- Pour spécifier l'ordre de tri: nous pouvons spécifier:
  - **`desc`** pour l'ordre décroissant
  - **`asc`** pour l'ordre croissant.
- Order by peut être effectuée sur plusieurs attributs.
- Requête 9: « répertorier la relation instructor par ordre décroissant de salaire. Si plusieurs instructeurs ont le même salaire, nous les classons par ordre croissant de nom ».

```
Select *
from  instructor
order by    salary desc, name
asc;
```

Les prédicats dans la clause where

- Between: opérateur de comparaison de la norme SQL.
  - simplifier les clauses where qui spécifient qu'une valeur doit être inférieure ou égale à une certaine valeur et supérieure ou égale à une autre valeur.
- Requete 10: trouver les noms des instructeurs dont le salaire se situe entre 90 000 et 100 000

```
select name  from
instructor
where salary between 90000 and
100000;
```

```
select name  from
instructor  where salary
<=
100000 and salary
>= 90000;
```

- `Not between` : De même, nous pouvons utiliser cet opérateur pour la comparaison

Les prédicats dans la clause where

- Le standar SQL nous permet d'utiliser la notation (v1, v2,..., vn) pour dénoter un tuple contenant les valeurs v1, v2,..., vn;
  - la notation est appelée constructeur de lignes.
- Les opérateurs de comparaison peuvent être utilisés sur des tuples, et l'ordre est défini lexicographiquement.
- Par exemple, *(a1, a2) <= (b1, b2)* est vrai si a1 <= b1 et a2 <= b2;
- Exemple requête SQL: (les deux possibilités)

```
select name, course_id  from
instructor, teaches
where instructor.ID= teaches.ID and dept_name =
'Biology';
```

```
select name, course_id  from
instructor, teaches
where (instructor.ID, dept_name)= (teaches.ID,
'Biology');
```

union, intersect et except

- Ces opérations opèrent sur des relations et correspondent aux opérations de l'ensemble mathématique U, ∩ et −.
- Exemple

L'ensemble de tous les cours enseignés au semestre d'automne 2017=C1

```
select course_id
from section
where semester = 'Fall' and year= 2017;
```

course_id
CS-101
CS-347
PHY-101

L'ensemble de tous les cours enseignés au semestre de printemps 2018 = C2

```
select course id
from section
where semester = 'Spring' and year= 2018;
```

course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

union

Trouver l'ensemble de tous les cours enseignés à l'automne 2017 ou au printemps 2018, ou les deux.

➔ les parenthèses autour de chaque instruction de sélection sont facultatives ➔ pour faciliter la lecture;

➔ certaines SGDB ne permettent pas l'utilisation des parenthèses.

```
(select course_id from section where semester = 'Fall'
and year= 2017)
union
(select course_id from section where semester = 'Spring'
and year= 2018);
```

- L'opération d'union élimine automatiquement les doublons (contrairement à la clause select). CS-101 et CS - 319 n'apparaît qu'une seule fois dans le résultat

c1 union c2 ➔

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

intersect

Pour trouver l'ensemble de tous les cours enseignés à l'automne 2017 et au printemps 2018

```
(select course_id from section where semester = 'Fall'
and year= 2017)
intersect
(select course_id from section where semester = 'Spring'
and year= 2018);
```

- Résultat: ne contient qu'un seul tuple avec CS-101.
- L'opération d'intersection élimine automatiquement les doublons.

c1 intersect c2 →

course_id
CS-101

NB. MySQL n'implémente pas l'opération d'intersection