
Bases de données IN206

Nicolas Anciaux
Nicolas.Anciaux@inria.fr

Lien web : <http://petrus/~anciaux/ENSTA/IN206/>

Nature et objectifs du module

- Un cours d'introduction (aucun pré requis)
- Objectif
 - Acquisition de la culture de base à l'ingénierie du SGBD
 - Suivre du cours BD avancées (ASI13) l'an prochain
- Connaissance
 - Fonctionnalités logicielles et raison d'être d'un SGBD
 - Indépendance physique/logique, vues, langage de manipulations, cohérence (contraintes d'intégrité et triggers), standards
 - Optimisation de questions, concurrence d'accès et gestion des pannes, gestion de la confidentialité
 - Connaissances techniques
 - Conception de BD (modèle EA, modèle relationnel) : 2 séances
 - Utilisation du SGBD (SQL, programmation SQL, JDBC/ODBC) : 3 séances
 - Notions plus avancées (transactions, NoSQL) : 2 séances

Objectifs en termes de compétences

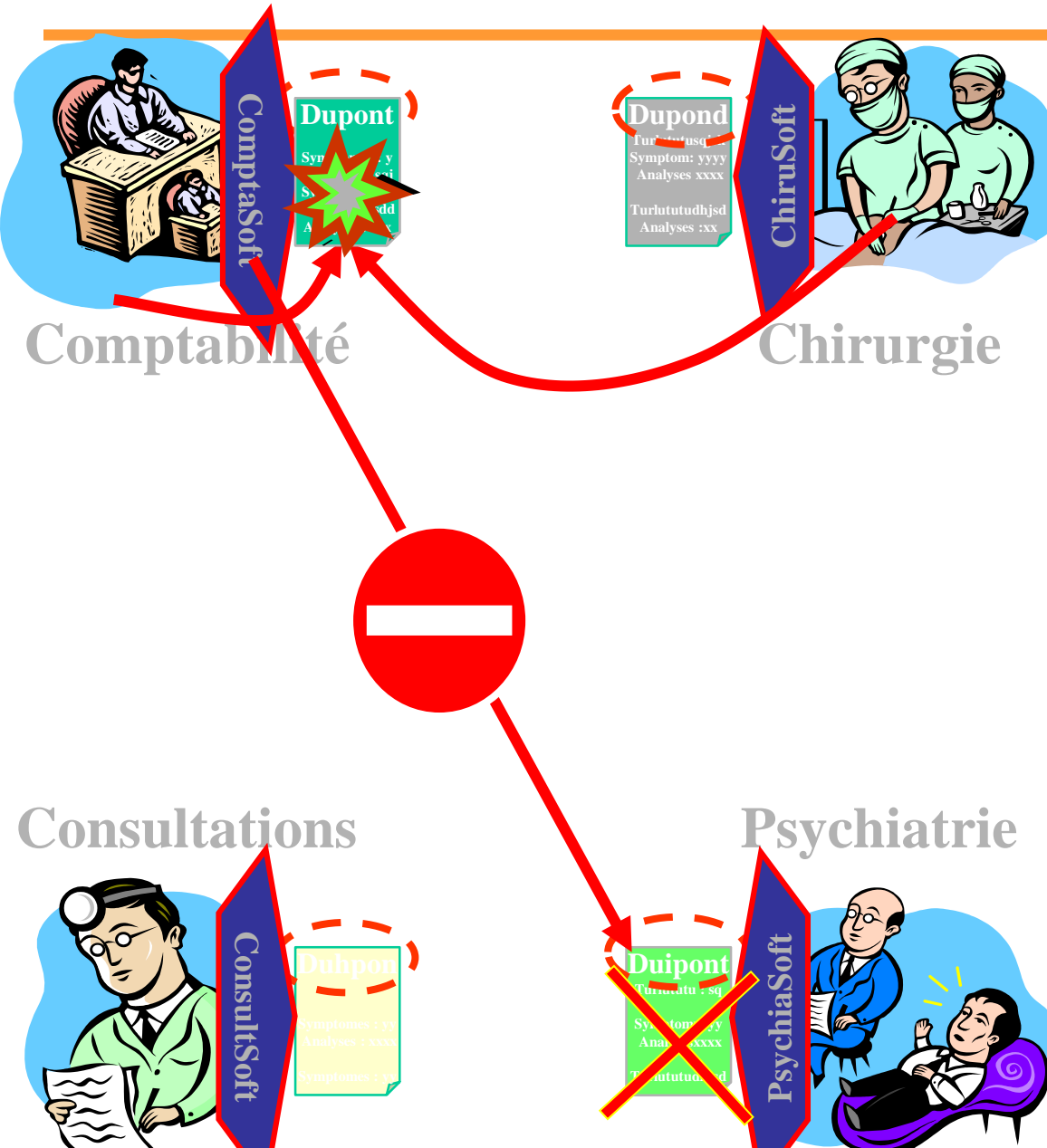
- Concevoir une base de données
 - Réaliser un modèle conceptuel avec le modèle E/A
 - Concevoir un modèle relationnelle de base de données
- Créer une application base de données
 - Créer un schéma relationnel en SQL
 - Ecrire des requêtes SQL d'interrogation/mise à jour
 - Interfacer un programme Java/JDBC avec une base de données
 - Ecrire et invoquer des fonctions et procédure stockées en PL/SQL
- Administrer une base de données en vue d'optimiser les performances
 - Optimiser une base de données multi utilisateurs (gestion de la concurrence)
- Introduction aux systèmes NoSQL

Plan des sessions

16/9	<ul style="list-style-type: none">• Approche BD (vs. fichier)• Conception 1 (modèle EA)	→ TD : Conception (modèle EA) ... et passage au relationnel
23/9	<ul style="list-style-type: none">• Modèle relationnel et algèbre• Conception 2 (relationnel)	→ TD : Exercices d'algèbre relationnelle
30/9	<ul style="list-style-type: none">• Vue d'ensemble SGBD• SQL 1 (LDD)	→ TP (XE) : Création d'une base (SQL) → TP (XE) : Insertion massive (SQL loader)
7/10	<ul style="list-style-type: none">• SQL 2 (LMD) et méthodologie	→ TP (XE) : Interrogation (SQL)
21/10	<ul style="list-style-type: none">• Programmation SQL• PL/SQL, ODBC/JDBC	→ TP (XE) : PL/SQL, OCILIB // CONTRÔLE CONTINU
4/11	<ul style="list-style-type: none">• Propriétés transactionnelles• Concurrency d'accès	→ TP (XE) : Expérience sur la concurrence
8/11	<ul style="list-style-type: none">• Introduction au NoSQL et à la sécurité des SGBD	→ Expériences sur MongoDB → Examen

L'approche bases de données (SGF vs. SGBD)

Confidentialité



Caractéristiques

Plusieurs applications

- plusieurs formats
- plusieurs langages

Redondance de données

Pas de facilité d'interrogation

- Question ⇒ développement

Redondance de code

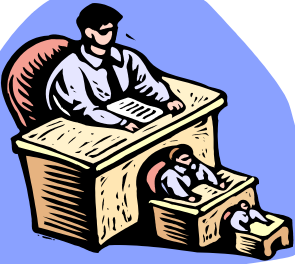
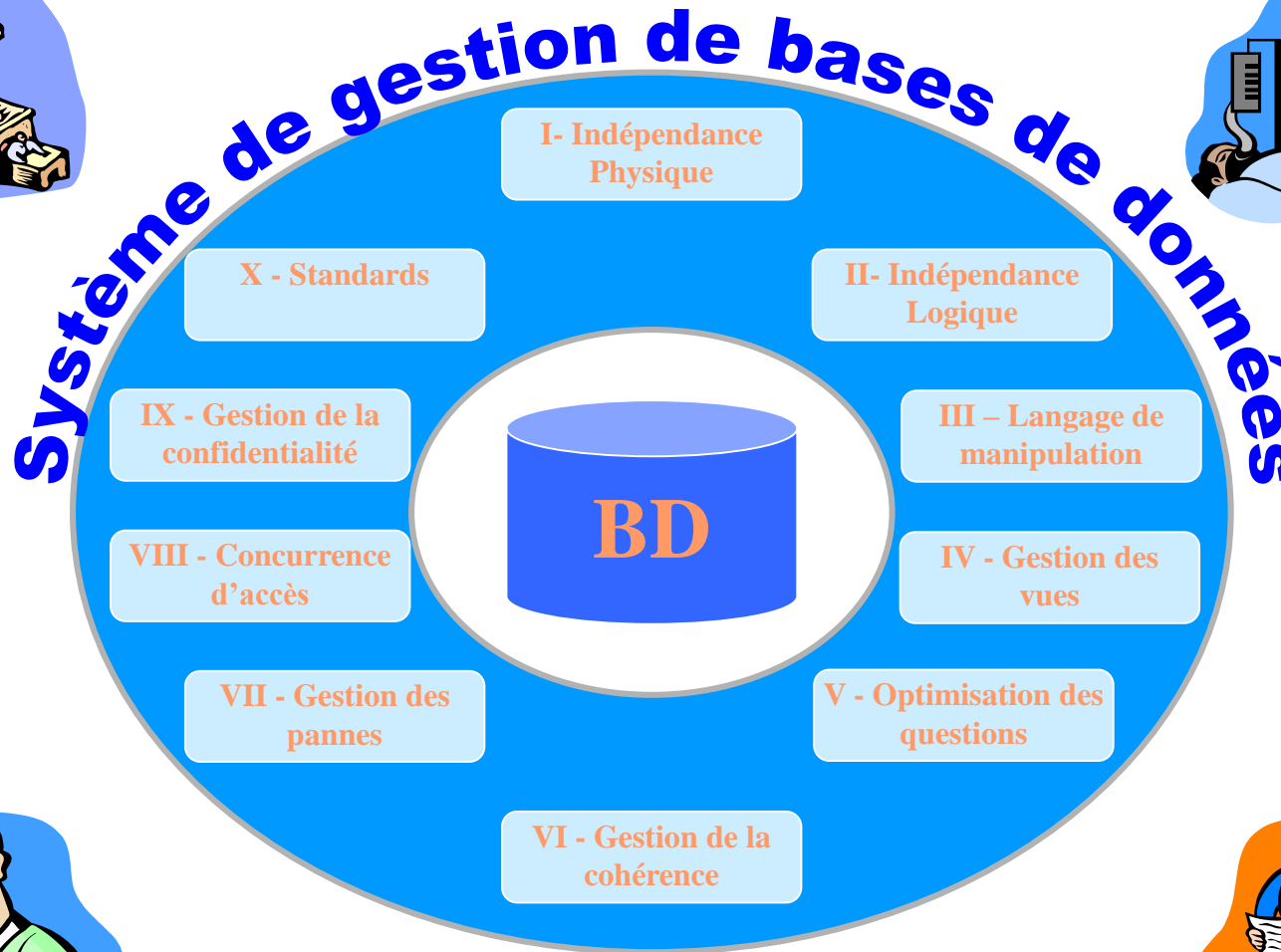
Problèmes

- Difficultés de gestion
- Incohérence des données
- Coûts élevés
- Maintenance difficile
- Gestion de pannes ???
- Partage des données ???
- Confidentialité ???

L'approche “Bases de données” (1)

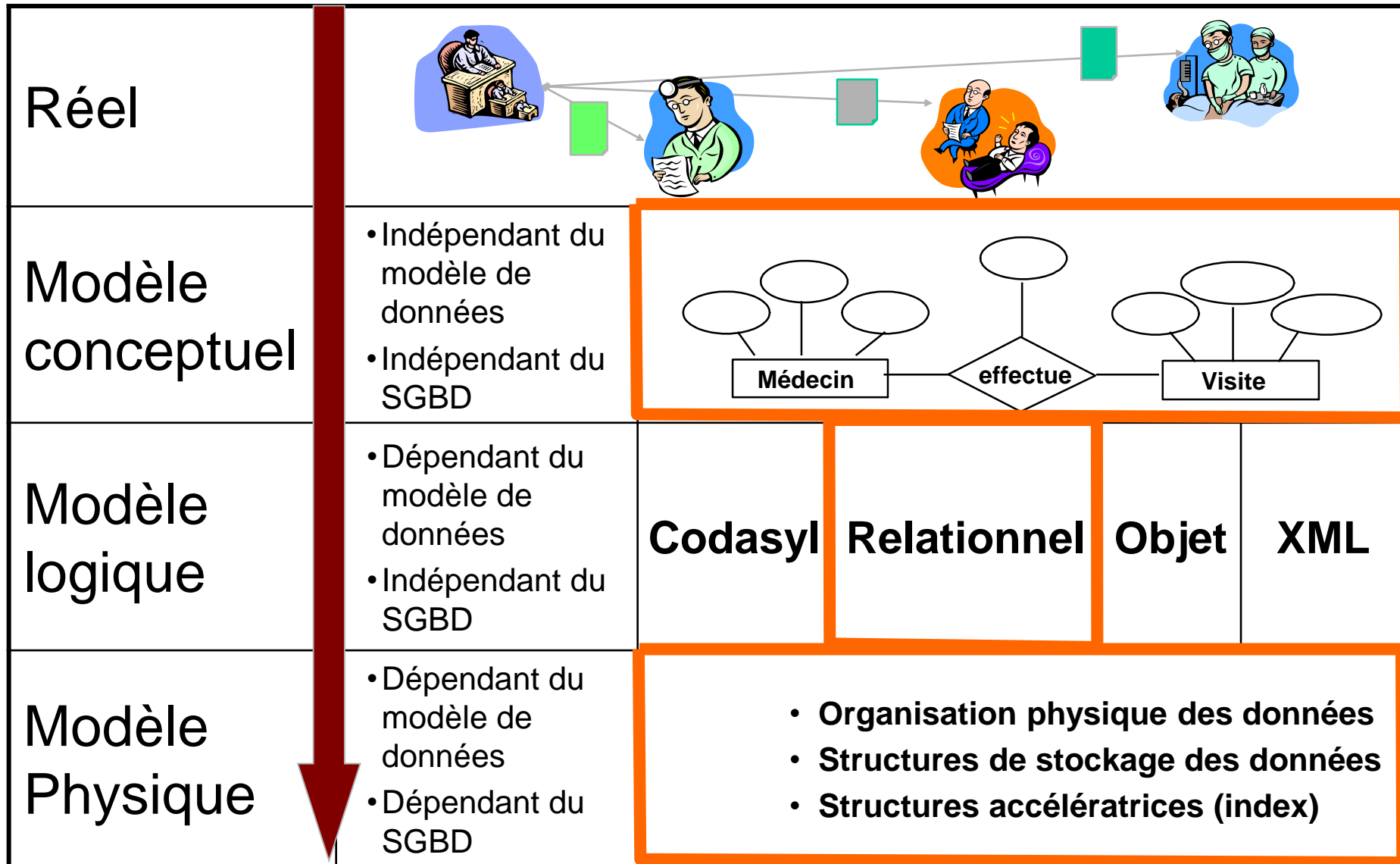
- **Modélisation des données**
 - Éliminer la **redondance** de données
 - **Centraliser** et **organiser** correctement les données
 - Plusieurs niveaux de modélisation
 - Outils de conception
- **Logiciel «Système de Gestion de Bases de Données»**
 - **Factorisation** des modules de contrôle des applications
 - Interrogation, cohérence, partage, gestion de pannes, etc...
 - Administration facilitées des données

L'approche "Bases de données" (2)



Conception de bases de données (Production du modèle conceptuel de données)

Modélisation du réel



Méthode de conception ?

- Plusieurs façons d'aborder la conception d'une BD
 - Intuition + création directe de la BD 😞
 - Suivre une méthode de conception (MCD→MLD→MPD) 😊
 - Entité/Association (E/A) ou Entity/Relationship (E/R)
 - Merise
 - UML
- Suivre son intuition peut conduire à des erreurs
 - Redondances
 - Valeurs nulles
 - Difficulté de gestion
 - Impossibilité de répondre à certaines questions
- Une fois la base de données créée, difficile à modifier... (cf. TP)
- Les outils de conception sont une aide précieuse



Exemple de mauvaise conception (1)

- Stocker des propriétaires de véhicule:

N°	Nom	Prénom	Ville	Pays	Immatriculation	Marque	Couleur
1	Bar	Joe	Paris	France	125 PP 75	Renault	Rouge
2	Dean	Pascal	Vence	France	234 FF 45	Peugeot	Vert
3	Ben	Zoe	Lyon	France	324 DFT 56	Renault	Rouge
4	Bar	Joe	Paris	France	245 FT 75	Renault	Jaune

- Redondance des données et incohérence potentielle
 - Personne répétée pour chaque voiture :
 - ex. Si Joe Bar change de ville et qu'une seule ligne est mise à jour...
 - Redondance Ville/Pays : impact d'une erreur de saisie
- Anomalies de mises à jour et besoin de valeurs nulles.
 - Comment insérer une personne sans voiture ?
 - Sémantique de calculs avec des valeurs nulles...
 - Comment supprimer la dernière voiture d'une personne ?

Exemple de mauvaise conception (2)

- Stocker des personnes qui ont des enfants:

N°	Nom	Prénom	Ville	Pays	Enfant1	Enfant2	Enfant3	NbEnfants
1	Bar	Joe	Paris	France	Paul	Zoe		2
2	Dean	Pascal	Vence	France	Lili			1
3	Ben	Zoe	Lyon	France	Sam	Tor	Tar	3
4	Cat	Tom	Lens	France				0

- Redondance cachée :
 - Nombre d'enfants vs enfants
- Difficulté de gestion
 - Comment gérer les personnes ayant plus de 3 enfants !
 - Comment afficher la liste des enfants ?

Méthodes de conception : Exemple Merise

Réel		
	DONNEES	TRAITEMENT
Modèle conceptuel	MCD Quelles données ? Quelle organisation ?	MCT Quels traitements ?
Modèle logique	MLD Modèle logique (e.g, relationnel)	MLT Structuration en procédure
Modèle Physique	MPD Création de la base de donnée	MPT Description de l'architecture des traitements, algorithmes

Objectif du cours : E/A, Merise, UML ? ➔ E/A light, Merise ultra-light

Approche proposée : orientée données

1/ Définir l'application (~MCT)

- Que veut-on faire exactement, définir les sorties (états)

2/ Définir les données (~MCD)

- quelles sont les données nécessaires ? Comment les organiser ?

3/ Définir les questions nécessaires pour l'application (~MLT)

4/ Validation : Est ce que la structure choisie permet de répondre aux questions ? Sinon, retour en 1/ ou 2/

5/ Passer du MCD au MLD

6/ Définir les requêtes nécessaires pour l'application (~MPT). Normalement, le MLD doit permettre de répondre aux requêtes ?

7/ Passer du MLD au MPD

GENERATION AUTOMATIQUE POSSIBLE !

Déf° (1) : entité / type d'entité

- **Entité** : représentation d'un objet du monde réel ...
 - ... par rapport au problème à modéliser. Une entité peut donc être ...
 - ... concrète : ex. un docteur, un médicament, un client
 - ... abstraite : ex. une visite médicale, une commande
- **Type d'entité** : représentation d'un ensemble d'entités perçues comme similaires et ayant les mêmes caractéristiques
 - Ex. docteurs, patients, médicaments, clients, visites, commandes

<i>Profs</i>	
<i>Bouganim</i>	<i>Profs</i>
<i>Luc</i>	<i>Creenn</i>
....	<i>Isabelle</i>

Entités

Profs
Nom
Prénom
Adresse
...

Type d'entité

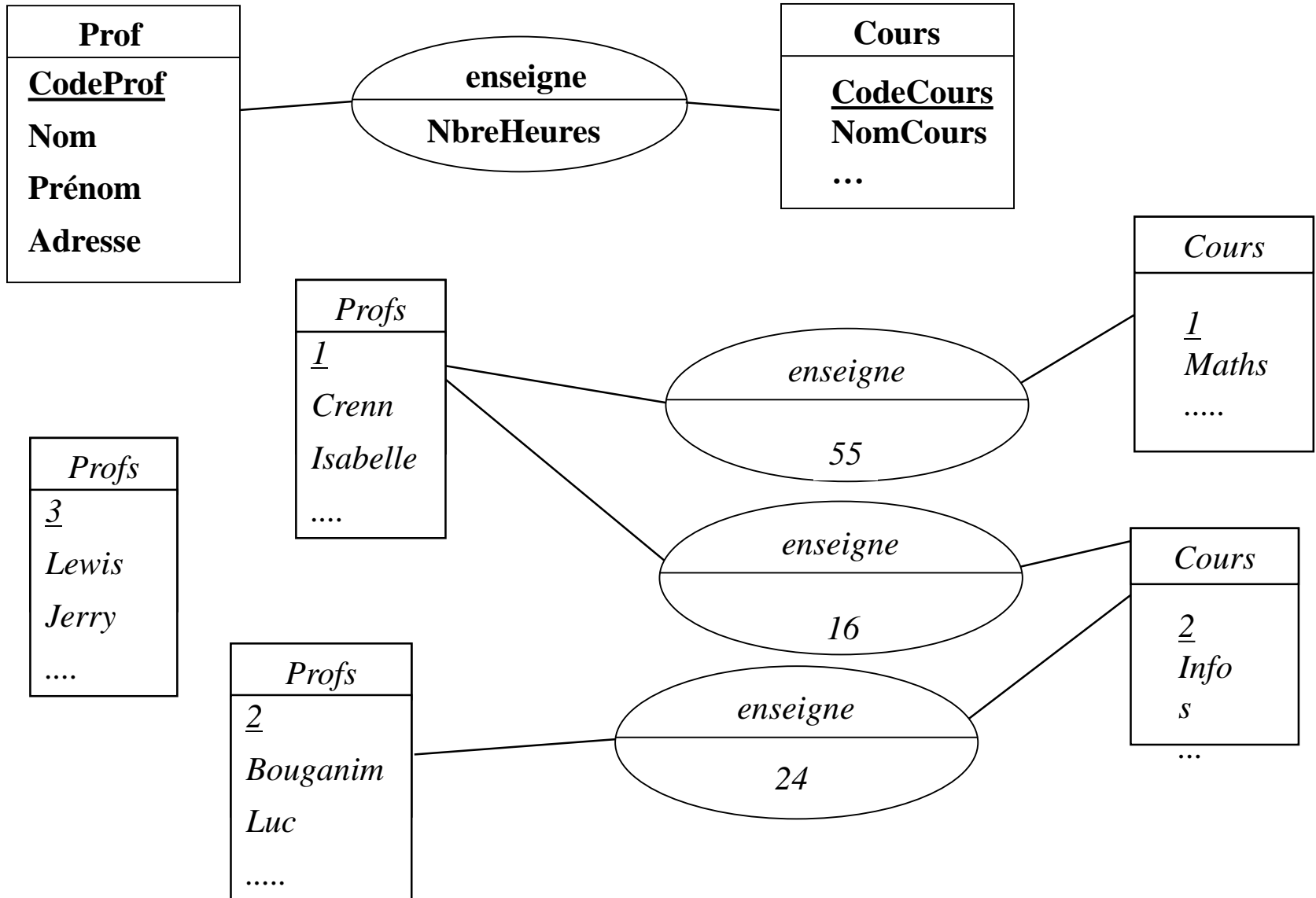
Déf° (2) : association / type d'association

- **Association** : représentation d'un lien non orienté entre plusieurs entités (qui jouent un rôle déterminé).
 - Ex. Un prof enseigne un cours
 - lien non orienté : un prof enseigne un cours → un cours est enseigné par un prof.
- **Type d'association** : représentation d'un ensemble d'associations ayant la même sémantique et les mêmes caractéristiques
 - Ex. enseigner
- **Question** : quid de visite : entité ou *association* ???
 - Un docteur *visite* un patient → association
 - Un docteur *effectue* une visite *concernant* un patient
 - Différence ? Et clients - commandes - produits ?

Déf° (3) : Propriétés / Identifiants

- **Propriété** : donnée élémentaire permettant de décrire une entité ou une association
 - Le nom du patient, la date de la visite, l'adresse du patient
- **Identifiant d'entité** : Une entité est identifiée de manière unique par au moins une propriété (généralement une)
 - Ex. n° de sécurité sociale du patient, référence d'un produit
- **Identifiant d'association** : il n'existe pas
 - On peut identifier une association par l'ensemble des identifiants des entités associées
 - Ex. pour *enseigne* : Code du prof, Code du cours

Exemple : Profs et cours...



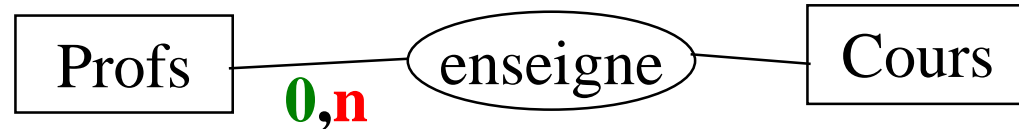
Déf°(4) : Cardinalités

- **Cardinalité** : Exprime le nombre minimum et maximum d'association(s) par entité. Il est indiqué sur chaque arc, entre le type d'entité et le type d'association concernées

- Pour un prof donné, combien d'enseignements ?

- Au minimum : **Min=0**

- Au maximum : **Max=n**

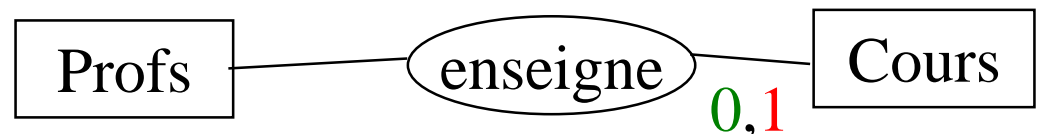


- Un prof enseigne de 0 à n cours

- Pour un cours donné, combien d'enseignements ?

- Au minimum : **0**

- Au maximum : **1**



- Un cours est enseigné par 0 à 1 prof

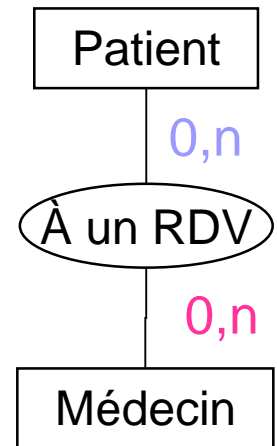
Comment produire le MCD ? (1)

- “Énoncer le réel” à modéliser avec des phrases

- Exemple pour la gestion de rendez vous

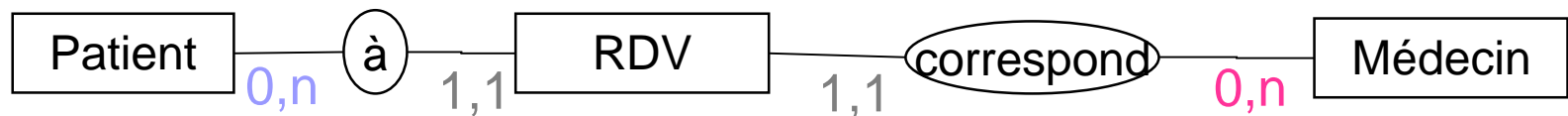
- Les patients ont des rendez vous avec des médecins

- Un patient peut avoir plusieurs RDV (voire aucun) → 0,n
- Un médecin reçoit plusieurs patients (voire aucun) → 0,n



- Un patient prendre plusieurs rendez-vous avec un même médecin

→ Ce MCD n'est pas bon !

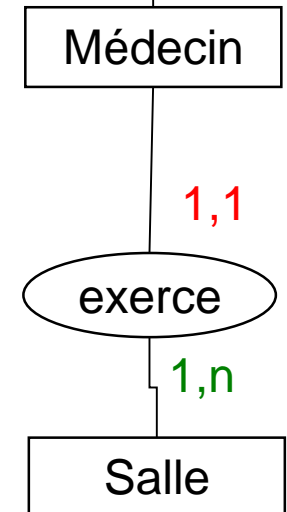


Comment produire le MCD ? (2)



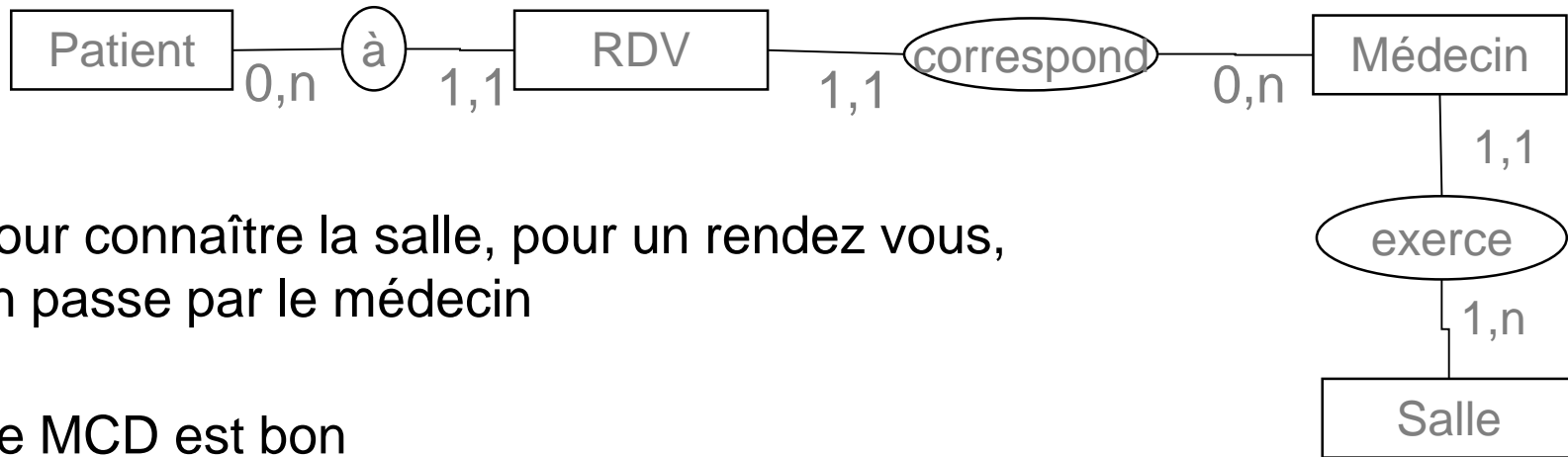
– Un médecin exerce dans une salle

- Un médecin n'exerce que dans une seule salle → 1,1
- Une salle peut être partagée par plusieurs médecins → 1,n



Comment produire le MCD ? (3)

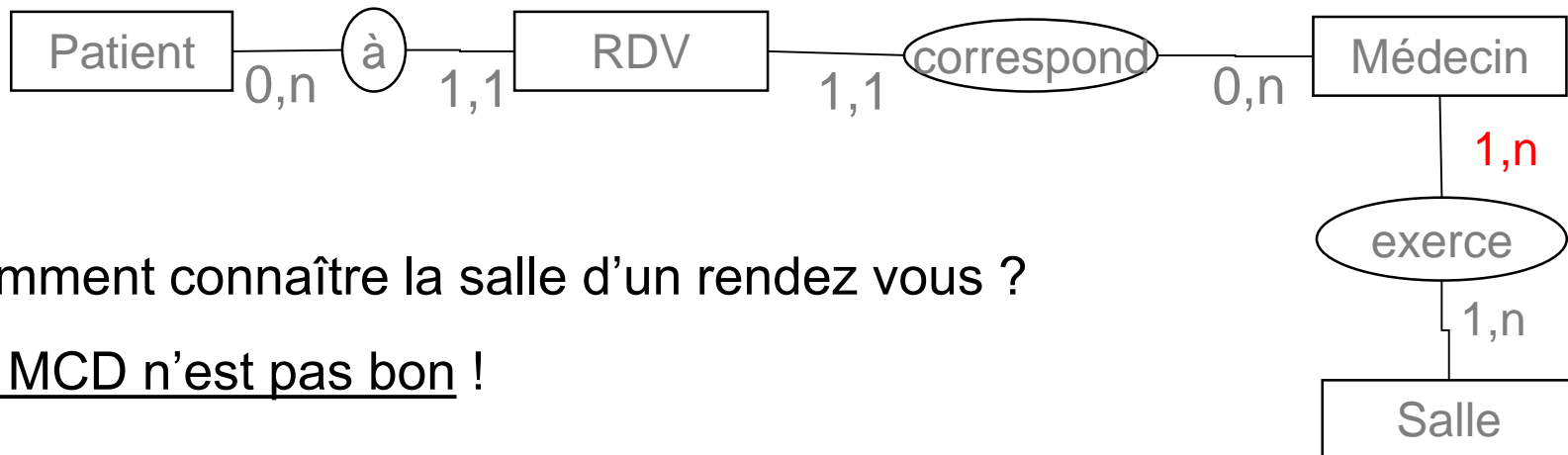
- On obtient :



- Pour connaître la salle, pour un rendez vous, on passe par le médecin

→ Ce MCD est bon

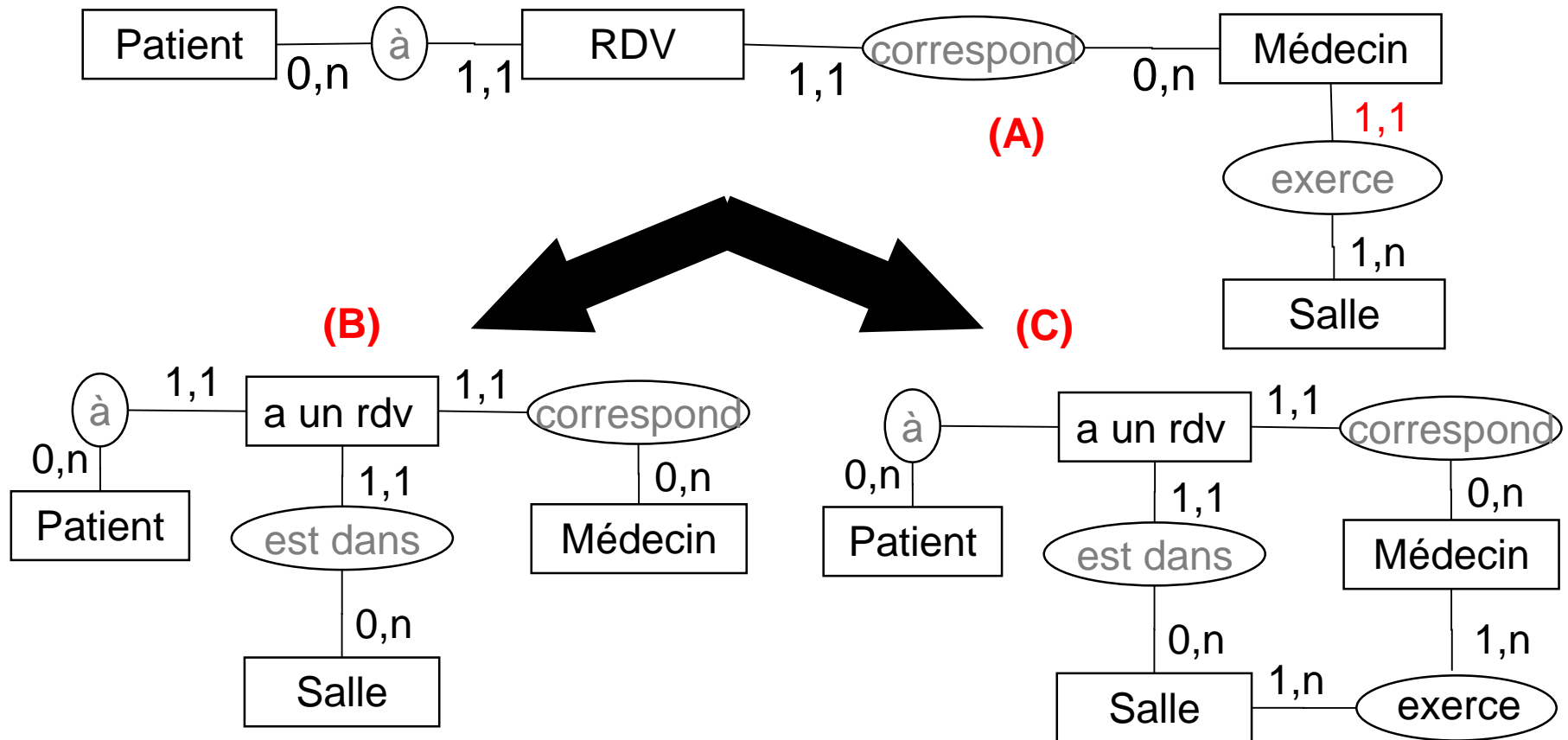
- Et si maintenant le médecin peut exercer dans **plusieurs** salles ?



- Comment connaître la salle d'un rendez vous ?

→ Ce MCD n'est pas bon !

Comment produire le MCD ? (4)



Différences fonctionnelles	(A)	(B)	(C)
Connaître la salle pour 1 RDV donné	Non	Oui	Oui
Pré-allouer des salles aux médecins	Non	Non	Oui

Règles (1) respect des règles de gestion

- Il faut vérifier que le MCD correspond bien au 'réel', c'est à dire aux règles fixées (celles que l'application doit respecter)
- par exemple:
 - un prof enseigne plusieurs cours
 - une matière est enseignée par plusieurs profs (info/anglais)
 - les notes peuvent être données par n'importe quel prof ou par plusieurs profs enseignant une matière... (info par exemple)
 - On peut redoubler une fois....
 - etc...

Règles (2) : propriétés élémentaires, calculées, constantes

- Toute propriété doit être élémentaire
 - sinon, complexité de traitement
 - Ex. de propriétés élémentaires : Age, Salaire, N° de rue
 - Ex. de propriétés non élémentaires : Adresse (complète), N°SS
 - la notion *d'élémentaire* dépend de l'application. L'adresse peut devenir élémentaire si elle est toujours manipulée comme telle (on ne cherchera jamais à faire, par exemple, un tri par ville)
 - « Il n'est pas gênant d'éclater des propriétés qui devrait être groupés, mais on ne peut pas grouper des propriétés qui devrait être éclatées »
- Une propriété calculée n'est pas à stocker !
 - Sinon, c'est redondant → source d'incohérence
- Une propriété constante n'est pas à stocker
 - Sinon, c'est redondant → source d'incohérence
 - On utilisera une table de constantes ...

Règles (3) : propriétés répétitives

- Pour une entité, il ne peut y avoir qu'une instance de chaque propriété de l'entité
 - **exemple**: Cours ne peut être une propriété de Prof, puisqu'un prof enseigne plusieurs cours...
 - **Remarque** : Si un prof ne peut enseigner qu'un seul cours, cours peut être une propriété de prof
- Attention aux propriétés n'ayant pas le même nom mais la même sémantique
 - Ex. : Enfant1, Enfant2, Enfant3 (cf. Slide 16)
 - Ex. : différents types de notes d'un étudiant
- Remarque : pour éviter d'éventuelles erreurs, on nommera différemment des propriétés de différentes entités :
 - Ex. NomPatient, NomDocteur, etc.
 - Du coup une propriété n'apparaît que dans une seule entité ou association.

Règles (4) : propriétés sans signification

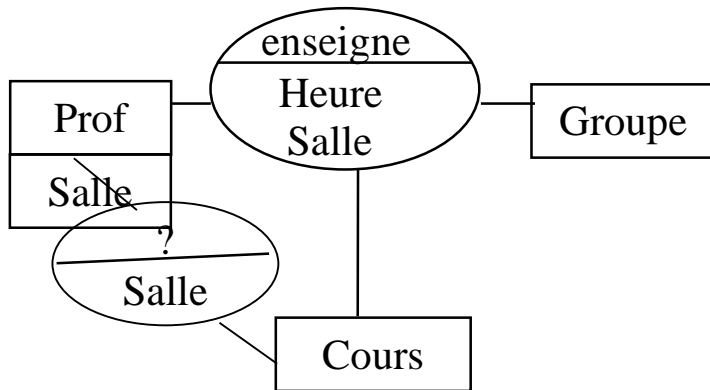
- Une propriété ne peut être sans signification pour une partie des entités ou associations
 - **exemple** : si un prof ne peut enseigner qu'un seul cours, mais qu'on a choisi de créer une entité 'personnel' et non 'prof', on ne stockera pas le cours dans l'entité 'personnel' car il serait sans signification pour une secrétaire...
 - **contre exemple** : Téléphone et Fax pour un étudiant...

Règles (5) : identifiants d'entités

- Toute entité doit être identifiée !
- Un identifiant doit être pérenne. Ex. nom et prénom peut être l'identifiant de l'étudiant, mais ça peut être insuffisant.
 - il ne faut pas concevoir le MCD en observant les données telles qu'elles sont (ex. l'école tel qu'elle est). Il faut le concevoir pour le cas à modéliser (ex. l'école telle qu'elle peut être.... et telle que l'on se propose de la gérer....)
- Plusieurs identifiants peuvent coexister
 - En choisir un...
- Si vous ne trouvez aucun identifiant, votre « entité » est sans doute une association ...

Règles (6) : dépendance pleine de l'identifiant

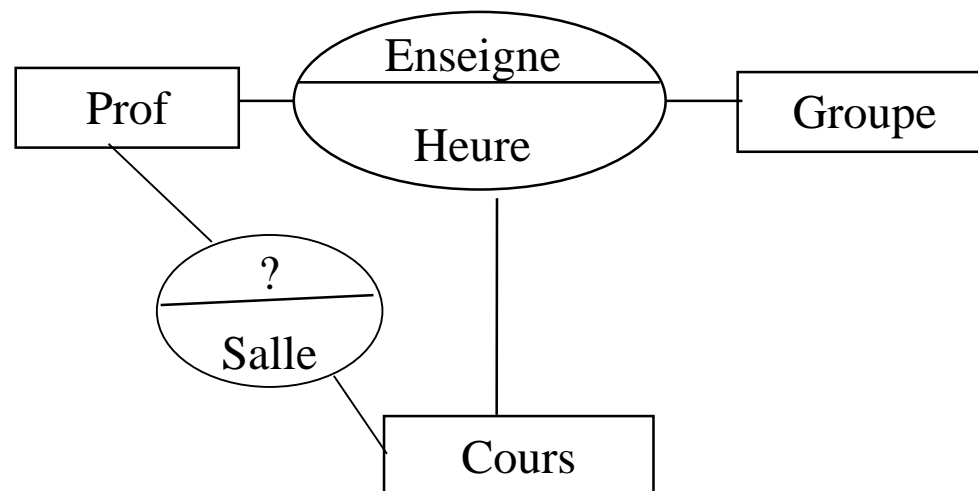
- Toute propriété doit dépendre pleinement de l'identifiant (et non d'une partie de celui-ci)
 - Sinon on introduit des redondances
- Les propriétés d'une **association** doivent dépendre de la totalité des entités associées
 - Sinon, les déplacer, voire créer une nouvelle association...
- Exemple :



- 1/ la salle dépend du prof, du cours et du groupe → OK
- 2/ la salle ne dépend que du prof → Not OK (dans prof)
- 3/ la salle ne dépend que du prof et du cours → not OK (nouvelle association entre prof et cours)

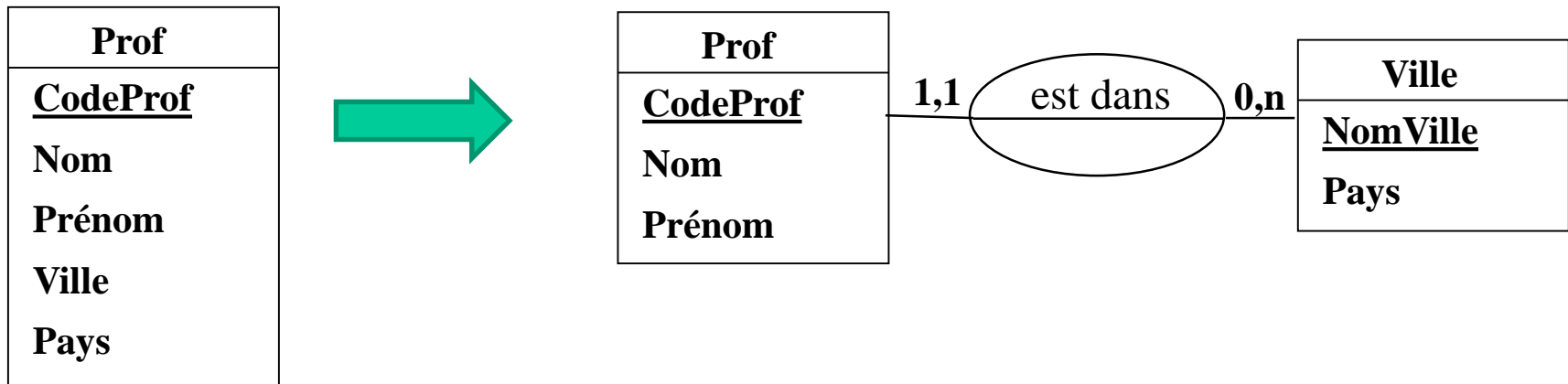
Règles (7) : entités et associations

- 2 entités ne peuvent être directement liées : il faut une association !
- Cependant:
 - Une association peut ne pas avoir de « nom »
 - Il est des fois difficile à trouver...
 - Une association peut ne pas avoir de propriété
 - C'est un cas très fréquent



Règles (8) : pas de dépendance transitive

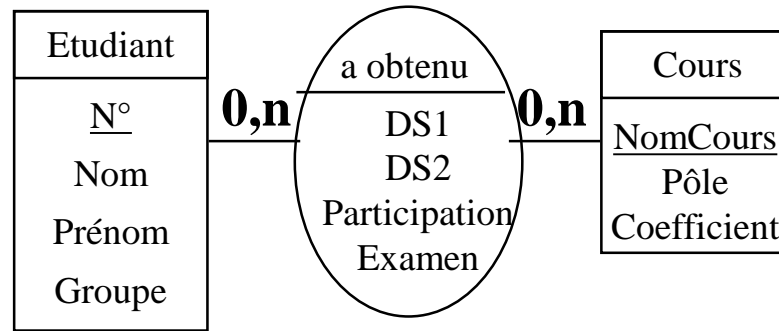
- Une propriété ne peut dépendre d'une autre propriété qui ne soit pas l'identifiant
 - Permet d'éliminer des sous-entités incluses dans une entité
- Exemple :
 - Etudiant(nom, adresse, ville, pays)
 - Le pays dépend de la ville or l'identifiant d'étudiant est le nom.



Première modélisation 'restreinte'

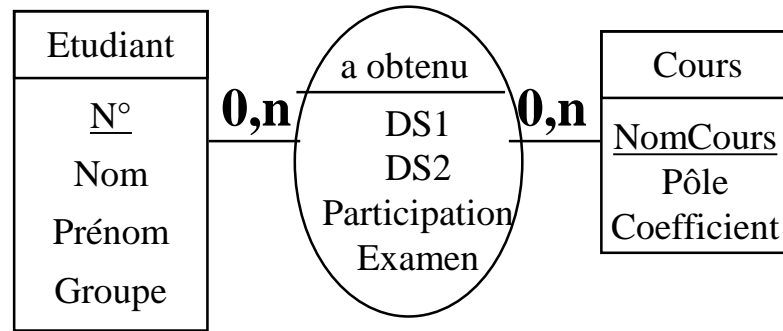
- Gérer les notes des étudiants
 - Hypothèses :
 - une base de données pour chaque promo et pour chaque semestre
 - Traitements :
 - Moyenne par cours pour chaque étudiant
 - Moyenne par pôle pour chaque étudiant
 - Moyenne générale pour chaque étudiant
 - Moyenne par groupe, par cours.
 - “Énoncer le réel” à modéliser avec des phrases
 - Un cours a un nom, un coefficient, et appartient à un pôle
 - Un étudiant a un nom et un prénom, et appartient à un groupe
 - Un étudiant obtient les notes DS1, DS2, participation, examen pour les cours qu'il suit

Modèle entité-association (1/2)



Commentaires?

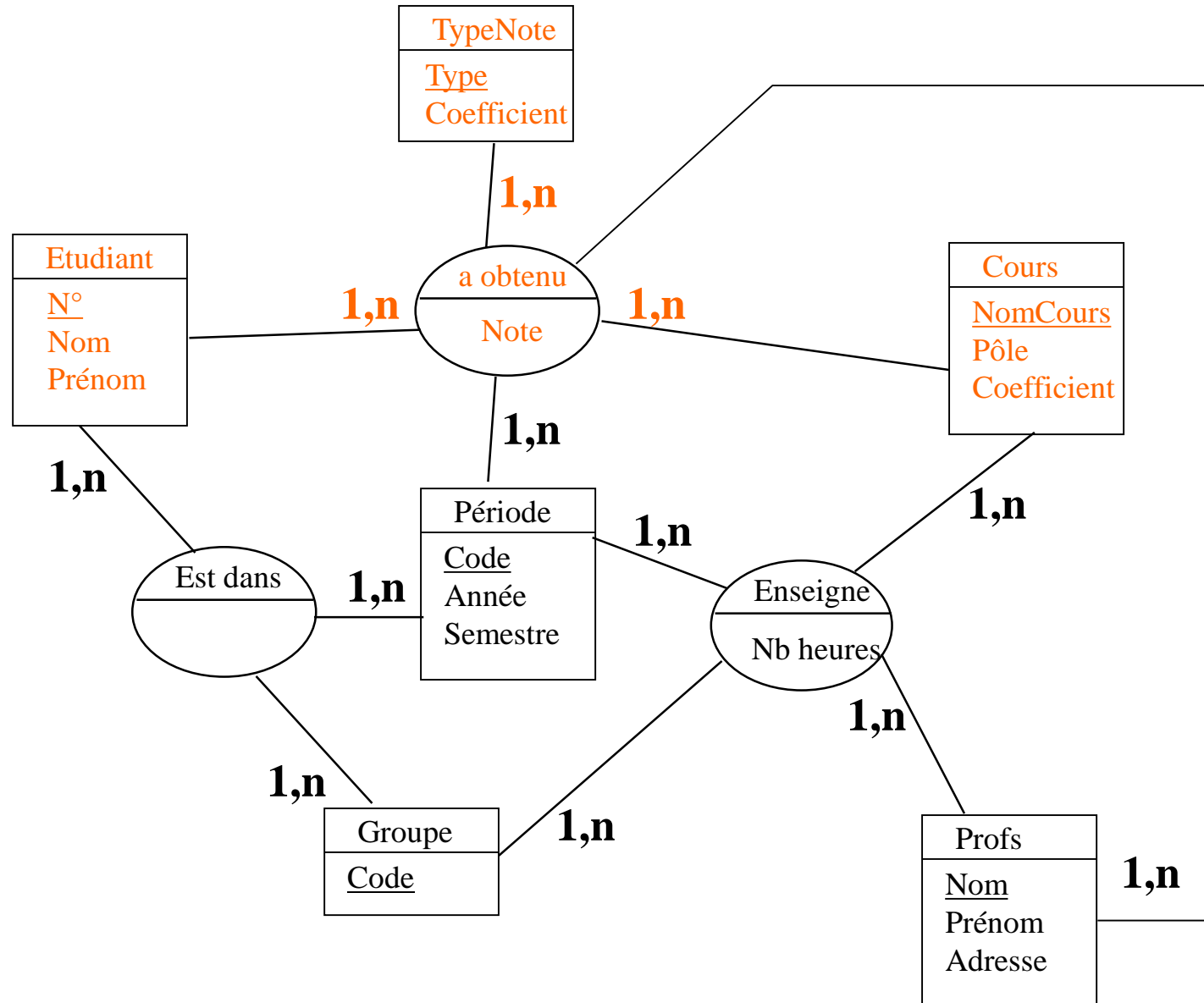
Modèle entité-association (2/2)



Modélisation 'complète'

- Gérer les notes des étudiants veut dire:
 - **Hypothèses :**
 - Une base de données pour l'Université (pour plusieurs années)
 - On veut gérer les profs pour faire des stats par profs, par promos, etc...
 - **Problèmes :**
 - Gestion des redoublement, de la situation (actuelle) d'un étudiant
 - Cohabitation de notes sur plusieurs années, des profs, des étudiants ??
 - Les matières sont enseignés par plusieurs profs, qui met les notes ??
 - Comment modéliser qu'un prof enseigne à un groupe de TP ?
 - **Données :**
 - Etudiants, Matières, Notes
 - TypeDeNotes, Perodes, Profs, Groupes, etc...

Modèle entité-association



TD Conception (MCD/Modèle EA)

Correction de l'exercice de conception

Quelques modèles Entité Associations possibles

Modèle relationnel

Le modèle relationnel

- En 1970, Edward Codd, Prix Turing 1981, chercheur chez IBM, propose le Modèle Relationnel, basé la Logique du premier ordre définie par les mathématiciens de la fin du 19e siècle pour formaliser le langage des mathématiques

*A Relational Model of Data for Large Shared Data Banks,
CACM 13, No. 6, June 1970*

Théorème de Codd: une question est exprimable en calcul relationnel **si et seulement si** elle peut être évaluée avec une expression de l'algèbre relationnelle – de plus, il est facile de transformer une requête du calcul relationnel en une expression algébrique qui évalue cette requête.

- Il définit le Calcul Relationnel et l'Algèbre Relationnelle, sur lesquels sont basés SQL (Structured Query Language), le langage standard de manipulation (LMD) et de description des données (LDD) de tous les SGBD Relationnels actuels

Domaine

- ENSEMBLE DE VALEURS
- Exemples:
 - ENTIER
 - REEL
 - CHAINES DE CARACTERES
 - EUROS
 - SALAIRE = {4 000..100 000}
 - COULEUR= {BLEU, BLANC, ROUGE}

Produit cartésien

- Le produit cartésien $D_1 \times D_2 \times \dots \times D_n$ est l'ensemble des tuples (n-uplets) $\langle V_1, V_2, \dots, V_n \rangle$ tels que $V_i \in D_i$
- Exemple:
 - $D_1 = \{\text{Bleu}, \text{Blanc}, \text{Rouge}\}$
 - $D_2 = \{\text{Vrai}, \text{Faux}\}$

$D_1 \times D_2 =$

Bleu	Vrai
Bleu	Faux
Blanc	Vrai
Blanc	Faux
Rouge	Vrai
Rouge	Faux

Relation, attribut

- Relation = sous-ensemble du produit cartésien d'une liste de domaines
 - Une relation est caractérisée par un nom
 - Exemple:
 - D1 = COULEUR
 - D2 = BOOLEEN
- Plus simplement ...
 - Une relation est une table à deux dimensions
 - Une ligne est un tuple
 - Un nom est associé à chaque colonne afin de la repérer indépendamment de son numéro d'ordre
- Attribut =
 - nom donné à une colonne d'une relation
 - prend ses valeurs dans un domaine

CoulVins	Coul	Choix
	Bleu	Faux
	Blanc	Vrai
	Rouge	Vrai

Exemple de relation

VINS	CRU	MILL	REGION	COULEUR
	CHENAS	1983	BEAUJOLAIS	ROUGE
	TOKAY	1980	ALSACE	BLANC
	TAVEL	1986	RHONE	ROSE
	CHABLIS	1986	BOURGOGNE	BLANC
	ST-EMILION	1987	BORDELAIS	ROUGE

Clé

- Définition = Groupe d'attributs minimum qui détermine un tuple unique dans une relation
- Exemples
 - {CRU,MILLESIME} DANS VINS
 - NSS DANS PERSONNE
- Contrainte d'entité
 - Toute relation doit posséder au moins une clé documentée

Clé Etrangère

- Définition = Groupe d'attributs devant apparaître comme clé dans une autre relation
- Les clés étrangères définissent les contraintes d'intégrité référentielles
 - Lors d'une insertion, la valeur des attributs doit exister dans la relation référencée
 - Lors d'une suppression dans la relation référencée les tuples référençant doivent disparaître
- Elles correspondent aux liens obligatoires entre les entités (modèle E/A)

Schéma

- Schéma de relation
 - Définition = Nom de la relation, liste des attributs avec domaines, et clé de la relation
 - Exemple
 - VINS(NV: Int, CRU:texte, MILL:entier, DEGRE: Réel, REGION:texte)
 - Par convention, la clé primaire est soulignée
- Intention et extension de relation
 - L'intention de la relation = le schéma de la relation
 - Une instance de table représente une extension de la relation
- Schéma d'une BD relationnelle = l'ensemble des schémas des relations composantes

Exemple de Schéma

- BUVEURS (NB, NOM, PRENOM, TYPE)
- VINS (NV, CRU, MILL, DEGRE)
- ABUS (NB, NV, DATE, QUANTITE)

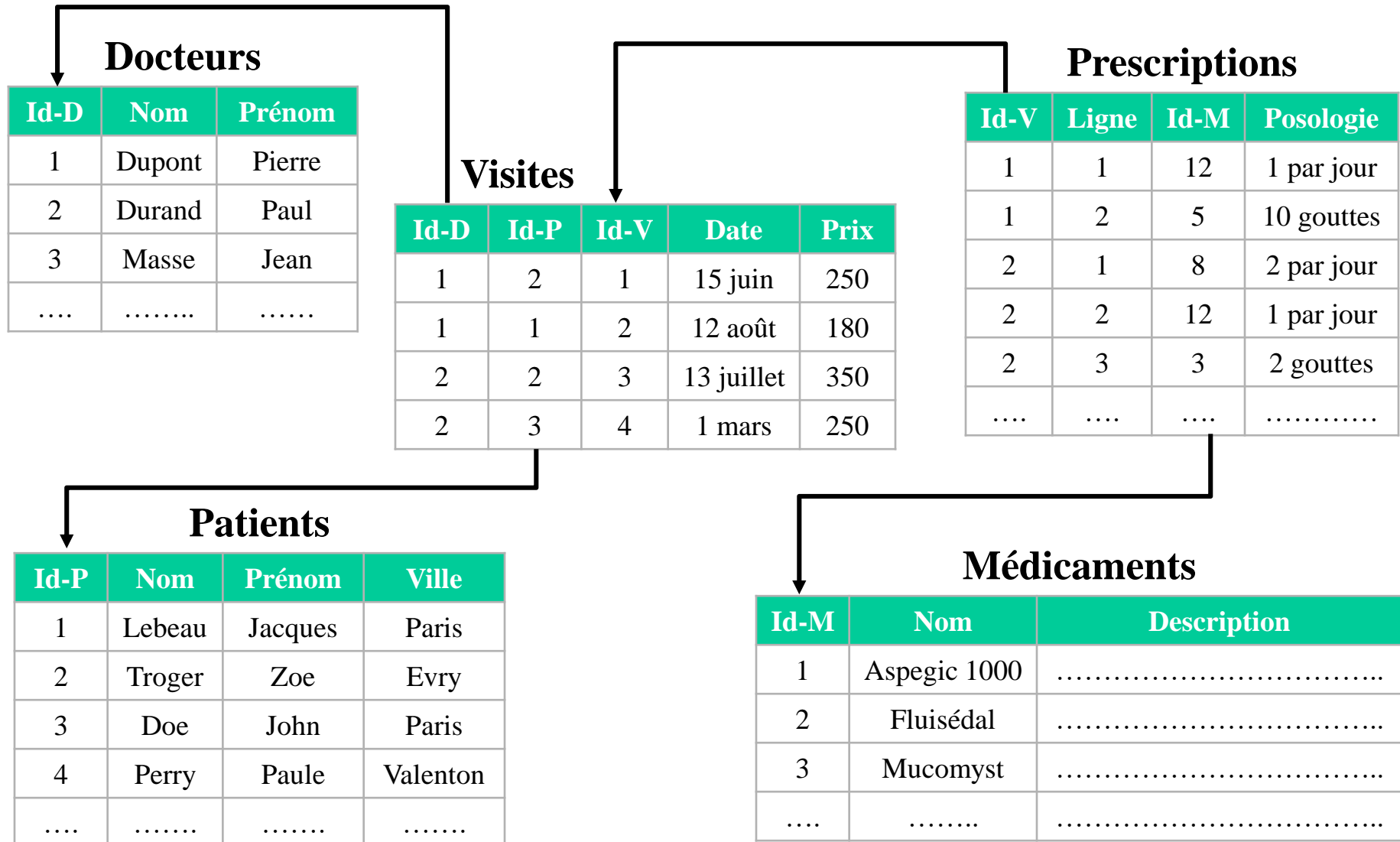
BUVEURS	<u>NB</u>	NOM	PRENOM	TYPE

VINS	<u>NV</u>	CRU	MILL.	DEGRE

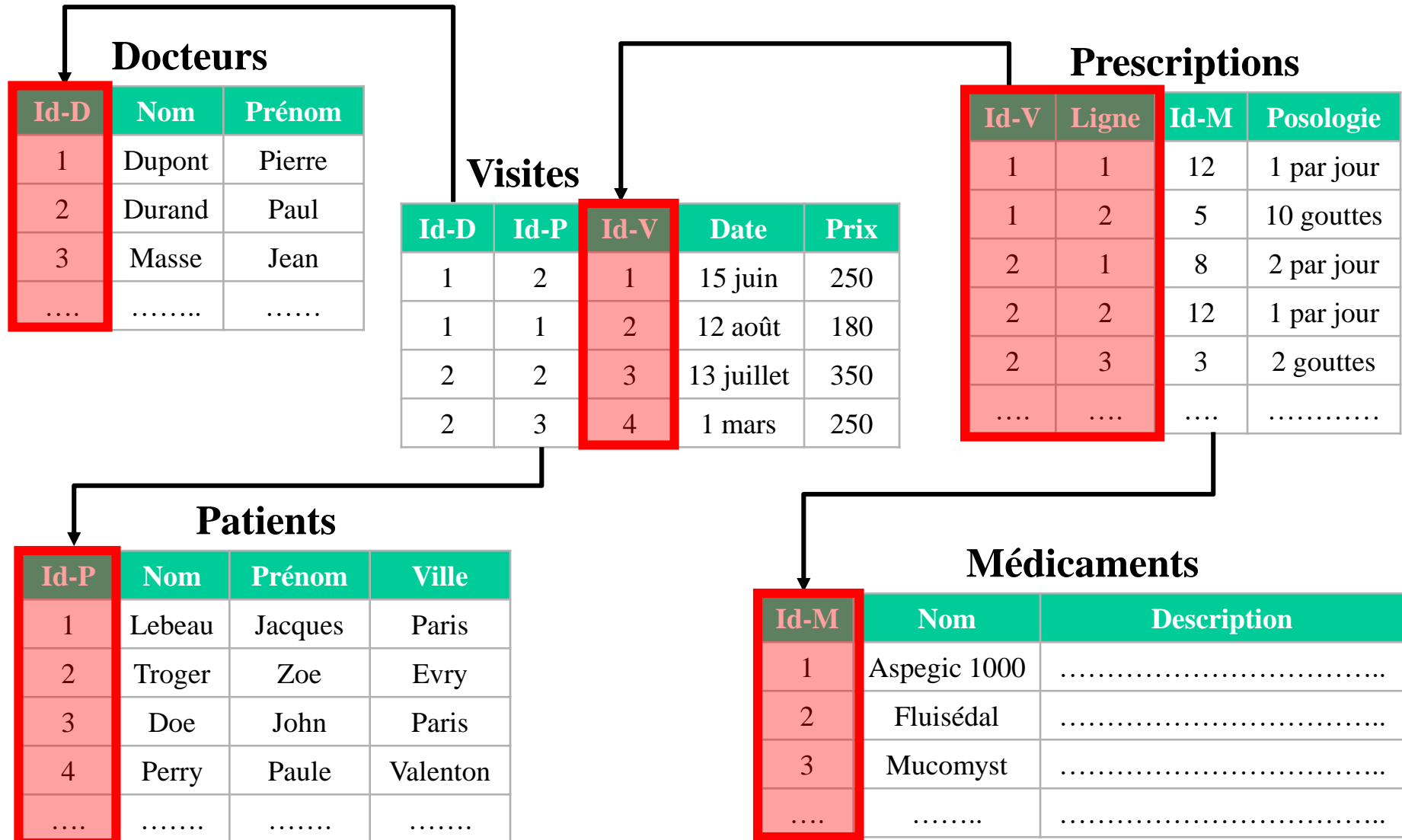
ABUS	<u>NB</u>	<u>NV</u>	<u>DATE</u>	QUANTITE

- CLES ETRANGERES (italique)
 - ABUS.NV référence VINS.NV
 - ABUS.NB référence BUVEURS.NB

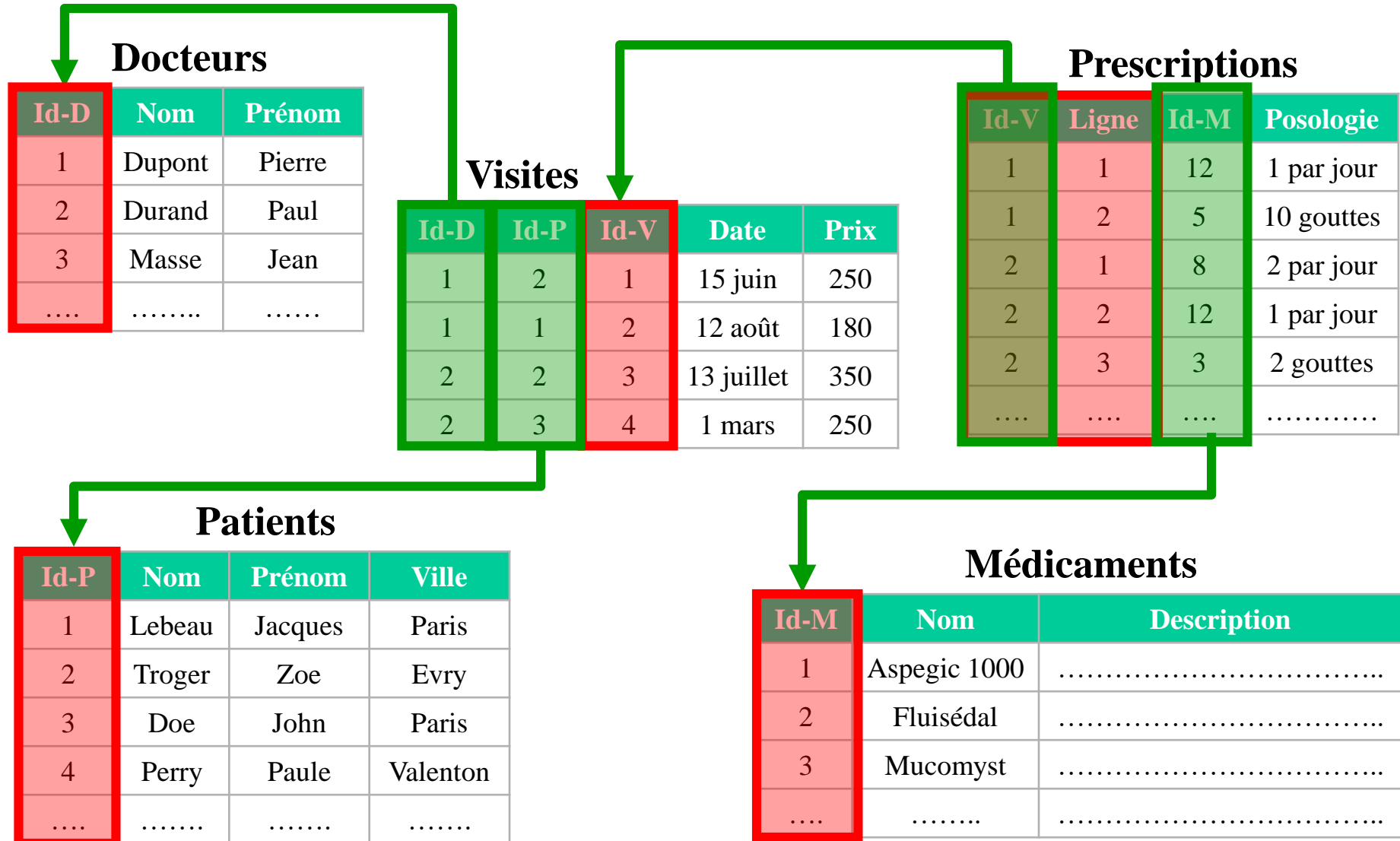
Exemple de modélisation relationnelle



Clés primaires



Clés étrangères



Métabase

DICTIONNAIRE DE DONNEES, ORGANISE SOUS FORME RELATIONNELLE, CONTENANT LA DESCRIPTION DES RELATIONS, ATTRIBUTS, DOMAINES, CLES, etc.

RELATIONS	NUM	BASE	NOM	NBATT
	12	PERSO	EMPLOYE	4
	14	PERSO	DEPARTEMENT	5
	1	SYS	RELATIONS	4

ATTRIBUTS	NUM	TYPE	NOM	NUMREL
	1	ENTIER	NUM	12
	2	TEXTE	NOM	12
	3	TEXTE	PNOM	12

Conception de bases de données (Passage au MLD)

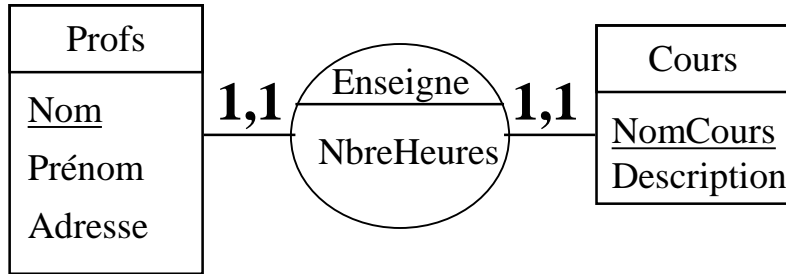
Passage au niveau logique

- Le modèle conceptuel est un **compromis** entre la flexibilité de la langue courante et la rigueur nécessaire d'un traitement informatisé.
- Le niveau logique est une étape de plus vers cette informatisation
 - Utilisation du **formalisme du modèle relationnel** :
 - tables (ou relations)
 - attributs
 - domaine
 - clefs
 - contrainte d'intégrité référentielles (relations entre tables)
 - **Simplification** du schéma de la base
 - Des règles trop strictes entraînent des schémas trop complexes
 - On "tolère" un peu de redondance ou quelques valeurs nulles....

Propriétés, Entités

- **Règle 1** : Chaque propriété devient un attribut.
- **Règle 2** : Chaque entité devient une table et son identifiant devient sa clef primaire
- **Règle 3** : Une association peut :
 - être “absorbée” par l’une ou l’autre des entités
 - devenir une table

Cas 1



- Un prof enseigne un et un seul cours
- Un cours est enseigné par un et un seul prof

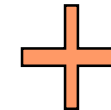
Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	44
Crenn	Isabelle	Paris	Math	Mathématiques	78
Rousseau	Martine	Versailles	Droit	Droit	26

Solution 2



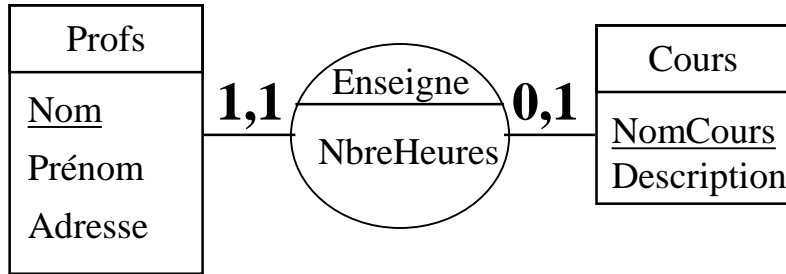
Nom	NomCours	Nbreheures
Bouganim	Info	44
Crenn	Math	78
Rousseau	Droit	26



Nom	Prénom	Adresse
Bouganim	Luc	Paris
Crenn	Isabelle	Paris
Rousseau	Martine	Versailles

NomCours	Description
Info	Informatique
Math	Mathématiques
Droit	Droit

Cas 2



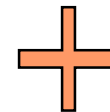
- Un prof enseigne un et un seul cours
- Un cours est enseigné par un prof ou n'est pas enseigné

Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	44
Crenn	Isabelle	Paris	Math	Mathématiques	78
			Droit	Droit	

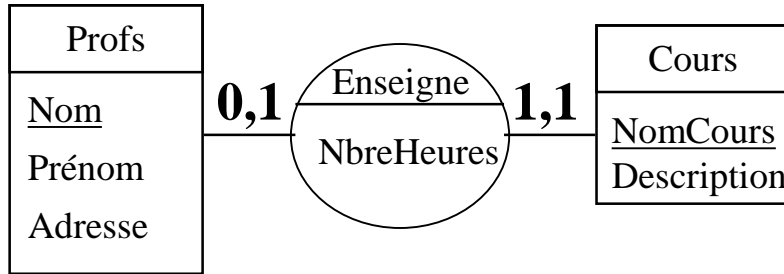
Solution 2

Nom	Prénom	Adresse	NomCours	NbreHeures
Bouganim	Luc	Paris	Info	44
Crenn	Isabelle	Paris	Math	78



NomCours	Description
Info	Informatique
Math	Mathématiques
Droit	Droit

Cas 3



- Un prof enseigne un cours ou aucun
- Un cours est enseigné par un et un seul prof

Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	44
Crenn	Isabelle	Paris	Math	Mathématiques	78
Rousseau	Martine	Versailles			

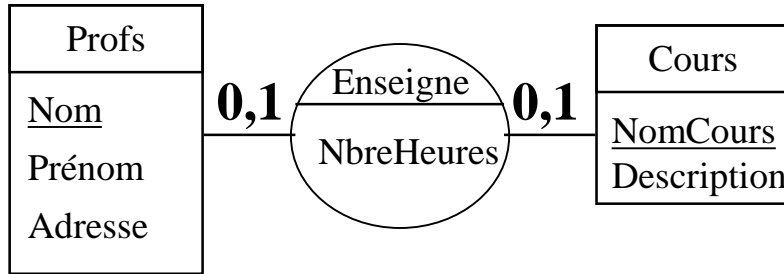
Solution 2

Nom	Prénom	Adresse
Bouganim	Luc	Paris
Crenn	Isabelle	Paris
Rousseau	Martine	Versailles



Nom	NomCours	Description	NbreHeures
Bouganim	Info	Informatique	44
Crenn	Math	Mathématiques	78

Cas 4

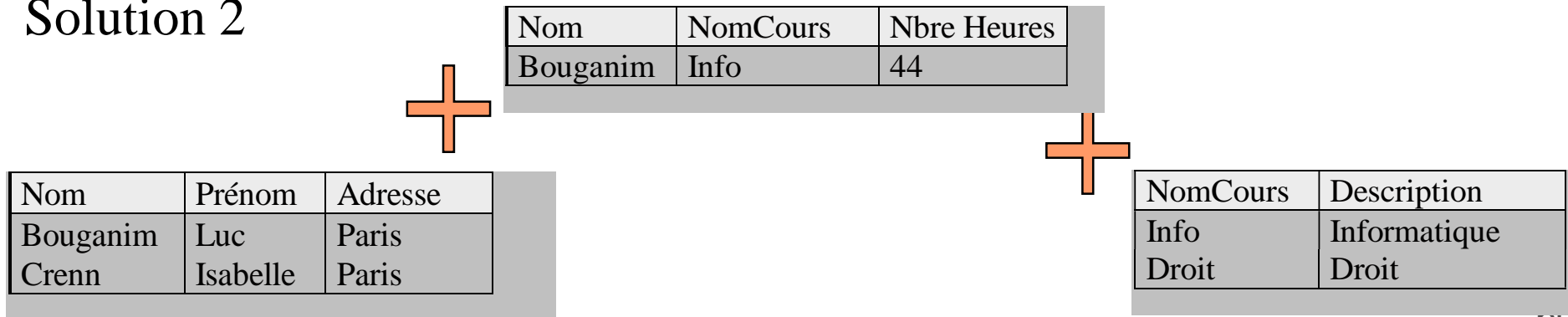


- Un prof enseigne un cours ou aucun
- Un cours est enseigné par un prof ou n'est pas enseigné

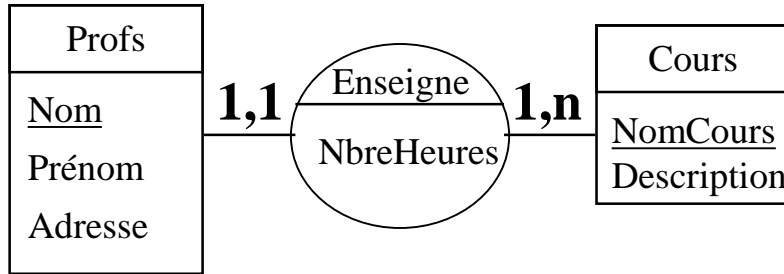
Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	44
Crenn	Isabelle	Paris	Droit	Droit	

Solution 2



Cas 5



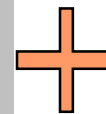
- Un prof enseigne un et un seul cours
- Un cours est enseigné par un ou plusieurs profs

Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	20
Crenn	Isabelle	Paris	Info	Informatique	24
Rousseau	Martine	Versailles	Droit	Droit	26

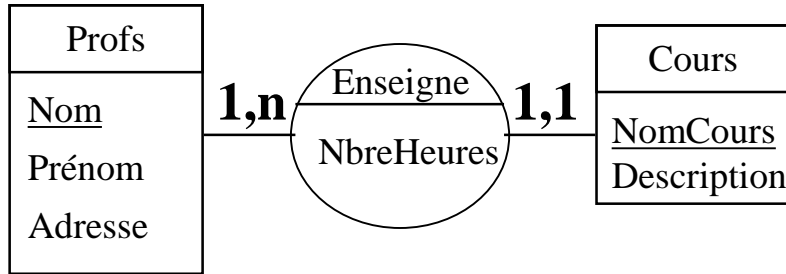
Solution 2

Nom	Prénom	Adresse	NomCours	NbreHeures
Bouganim	Luc	Paris	Info	20
Crenn	Isabelle	Paris	Info	24
Rousseau	Martine	Versailles	Droit	26



NomCours	Description
Info	Informatique
Droit	Droit

Cas 6



- Un prof enseigne un ou plusieurs cours
- Un cours est enseigné par un et un seul prof

Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	20
Crenn	Isabelle	Paris	Math	Mathématique	48
Crenn	Isabelle	Paris	Droit	Droit	26

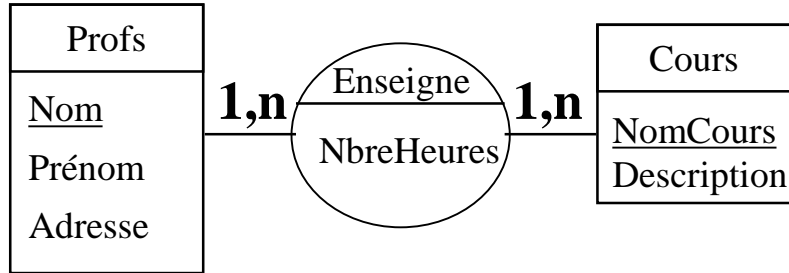
Solution 2

Nom	Prénom	Adresse
Bouganim	Luc	Paris
Crenn	Isabelle	Paris



Nom	NomCours	Description	NbreHeures
Bouganim	Info	Informatique	20
Crenn	Math	Mathématique	48
Crenn	Droit	Droit	26

Cas 7



- Un prof enseigne un ou plusieurs cours
- Un cours est enseigné par un ou plusieurs profs

Solution 1

Nom	Prénom	Adresse	NomCours	Description	NbreHeures
Bouganim	Luc	Paris	Info	Informatique	22
Crenn	Isabelle	Paris	Info	Informatique	26
Crenn	Isabelle	Paris	Droit	Droit	34

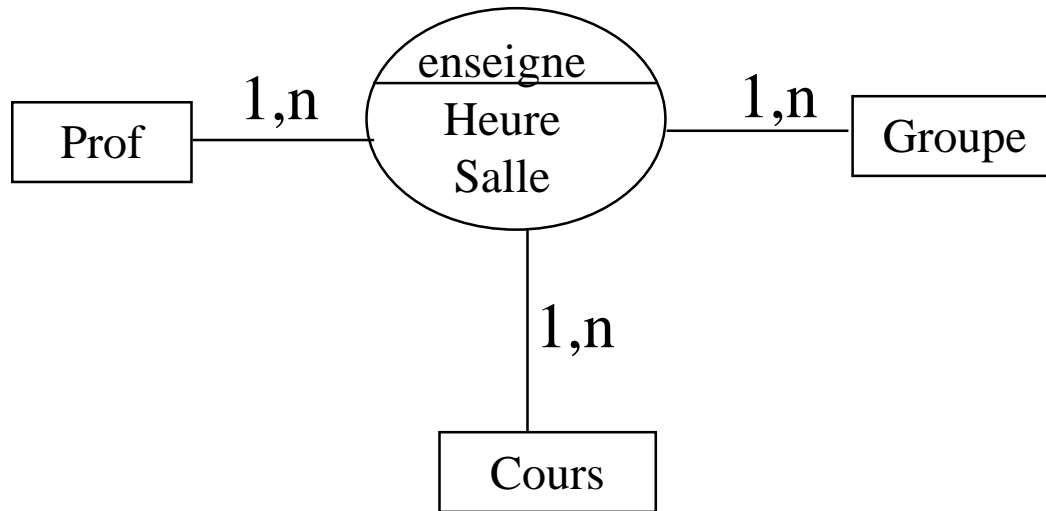
Solution 2

Nom	NomCours	Nbreheures
Bouganim	Info	22
Crenn	Info	26
Crenn	Droit	34

Nom	Prénom	Adresse
Bouganim	Luc	Paris
Crenn	Isabelle	Paris

NomCours	Description
Info	Informatique
Droit	Droit

Cas 8



Nom	NomCours	Groupe	heure	Salle
Bouganim	Info	2.1	10h	A1
Crenn	Math	2.1	12h	A3
Crenn	Info	2.2	17h	A1
Bouganim	Info	2.1	14h	A2

Nom	Prénom	Adresse
Bouganim	Luc	Paris
Crenn	Isabelle	Paris

NomCours	Description
Info	Informatique
Math	Mathématique

Groupe	Option	Responsable
2.1	Finance	Guter Paul
2.2	Comptabilité	Bourdin Jean

Passage au modèle relationnel - Conclusion

- **Objectifs**

- Ne pas créer de tables inutiles
- Ne pas dégrader le modèle conceptuel (pas de propriété répétitive ni sans signification)

- **Méthode**

- Si possible, passer les propriétés de l'association dans l'une **ou** l'autre des entités mais:
 - Si la cardinalité **minimum est 0**, on ne peut le faire car, pour certaines entités, il y aurait des valeurs nulles (ex. un prof ne donnant pas de cours)
 - Si la cardinalité **maximum est n**, on ne peut le faire car il y aurait des attributs répétitif (ex. un prof donnant plusieurs cours)
- Sinon, créer une table pour l'association contenant
 - les clefs des entités associées
 - les propriétés de l'association

Algèbre relationnelle

Algèbre relationnelle

OPERATIONS PERMETTANT D'EXPRIMER LES REQUETES SOUS FORME D'EXPRESSIONS ALGEBRIQUES

- Avantages
 - Concis
 - Sémantique simple
 - Représentation graphique
 - Utile pour raisonner (cf. TD) – peu d'erreur de syntaxe !
 - Utile pour l'optimisation de requêtes

Projection

- Elimination des attributs non désirés et suppression des tuples en double
- Relation \rightarrow Relation notée: $\pi_{a1,a2,\dots,Ap} (R)$

VINS	Cru	Mill	Région	Qualité
VOLNAY	1983	BOURGOGNE	A	
VOLNAY	1979	BOURGOGNE	B	
CHENAS	1983	BEAUJOLAIS	A	
JULIENAS	1986	BEAUJOLAIS	C	

$\pi_{\text{Cru,Région}}$



$\pi(\text{VINS})$	Cru	Région
	VOLNAY	BOURGOGNE
	CHENAS	BEAUJOLAIS
	JULIENAS	BEAUJOLAIS

$\pi_{\text{Cru,Région}} (\text{VINS}) =$

Restriction

- Obtention des tuples de R satisfaisant un critère Q
- Relation \rightarrow Relation, notée $\sigma_Q(R)$
- Q est le critère de qualification de la forme :
 - $A_i \theta \text{ Valeur}$, $\theta = \{ =, <, \geq, >, \leq, \neq \}$
 - Il est possible de réaliser des "ou" (conjonction) et des "et" (disjonction) de critères simples

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C



$\sigma_{\text{MILL} > 1983}$

$\sigma_{\text{MILL} > 1983}(\text{VINS}) =$

VINS	Cru	Mill	Région	Qualité
	JULIENAS	1986	BEAUJOLAIS	C

Jointure

- Composition des deux relations sur un domaine commun
- $\text{Relation} \times \text{Relation} \rightarrow \text{Relation}$
 - notée \bowtie
- Critère de jointure
 - Attributs de même nom égaux
 - $\text{Attribut} = \text{Attribut}$
 - Jointure naturelle
 - Comparaison d'attributs
 - $\text{Attribut1} \Theta \square \text{Attribut2}$
 - Théta-jointure
 - Cas particulier : équi-jointure

Exemple de Jointure


VINS	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A
	JULIENAS	1986	C



LIEU	Cru	Région	QualMoy
	VOLNAY	Bourgogne	A
	CHABLIS	Bourgogne	A
	CHABLIS	Californie	B



VINSREG	Cru	Mill	Qualité	Région	QualMoy
	VOLNAY	1983	A	Bourgogne	A
	VOLNAY	1979	B	Bourgogne	A
	CHABLIS	1983	A	Bourgogne	A
	CHABLIS	1983	A	Californie	B

VINS  **LIEU** =
Cru=Cru

Exemple de Θ -Jointure

EMPLOYES	NOM	DEPT	SALAIRE	RESPONSABLE	DEPT	SALAIRE
	MARTIN	1	130		1	230
	DURAND	1	235		2	250
	DUPONT	2	280		3	300
	DUPOND	3	150		4	270

EMPLOYES E \bowtie RESPONSABLE R =
 E.salaire > R.salaire et E.dept = R.dept

RESULTAT	NOM	DEPT	E.SALAIRE	R.SALAIRE
	DURAND	1	235	230
	DUPONT	2	280	250

NB : signification de la requête ?

Employés ayant un salaire supérieur à celui du responsable de leur département

Unions, intersections, différences

- Opérations binaires
 - Relation \times Relation \rightarrow Relation
- Opérations pour des relations de même schéma
 - UNION notée \cup
 - INTERSECTION notée \cap
 - DIFFERENCE notée $-$
- Extension pour des relations de schémas différents
 - Ramener au même schéma avec des valeurs nulles

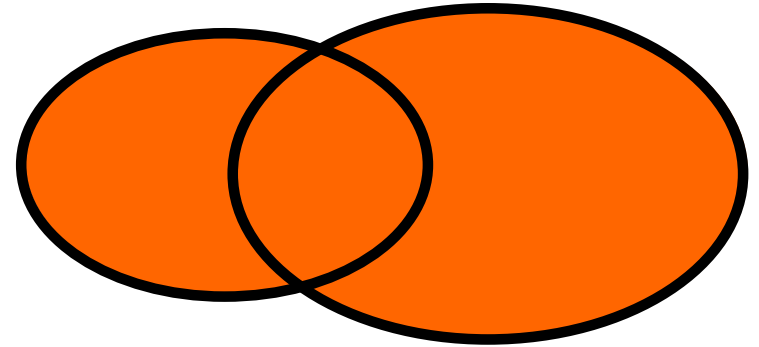
Union (\cup)

UNION ENSEMBLISTE POUR DES RELATIONS DE MEME SCHEMA

Exemple : VINS1 \cup VINS2

Vins1	Cru	Mill	Region	Couleur
	CHENAS	1983	BEAUJOLAIS	ROUGE
	TOKAY	1980	ALSACE	BLANC
	TAVEL	1986	RHONE	ROSE

Vins2	Cru	Mill	Region	Couleur
	TOKAY	1980	ALSACE	BLANC
	CHABLIS	1985	BOURGOGNE	ROUGE



VINS	CRU	MILL	REGION	COULEUR
	Chenas	1983	Beaujolais	Rouge
	Tokay	1980	Alsace	Blanc
	Tavel	1986	Rhône	Rosé
	Chablis	1985	Bourgogne	Rouge

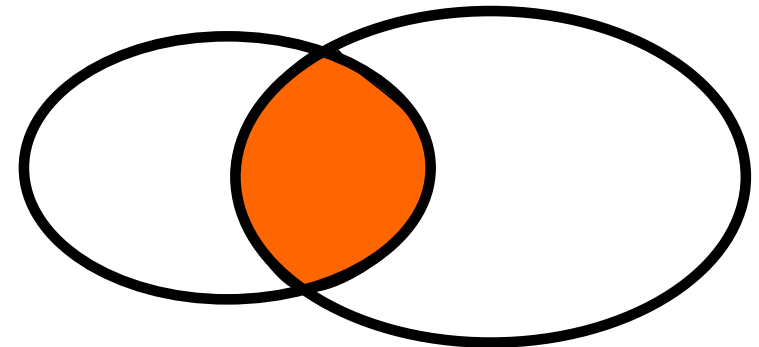
Intersection (\cap)

INTERSECTION ENSEMBLISTE POUR DES RELATIONS DE MEME SCHEMA

Exemple : VINS1 \cap VINS2

Vins1	Cru	Mill	Region	Couleur
	CHENAS	1983	BEAUJOLAIS	ROUGE
	TOKAY	1980	ALSACE	BLANC
	TAVEL	1986	RHONE	ROSE

Vins2	Cru	Mill	Region	Couleur
	TOKAY	1980	ALSACE	BLANC
	CHABLIS	1985	BOURGOGNE	ROUGE



VINS	CRU	MILL	REGION	COULEUR
	Tokay	1980	Alsace	Blanc

Différence (-)

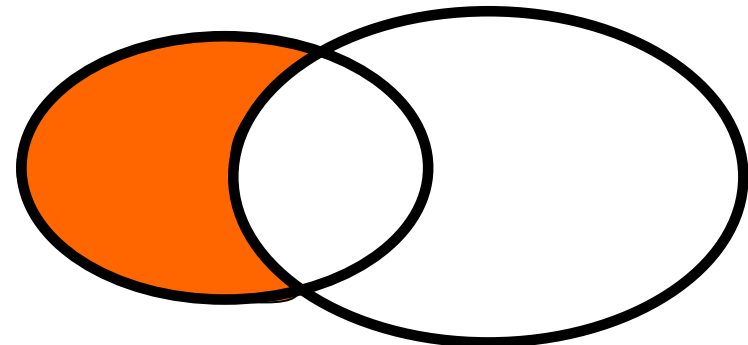
DIFFERENCE ENSEMBLISTE POUR DES RELATIONS DE MEME SCHEMA

Exemple : VINS1 — VINS2

Vins1	Cru	Mill	Region	Couleur
	CHENAS	1983	BEAUJOLAIS	ROUGE
	TOKAY	1980	ALSACE	BLANC
	TAVEL	1986	RHONE	ROSE

Vins2	Cru	Mill	Region	Couleur
	TOKAY	1980	ALSACE	BLANC
	CHABLIS	1985	BOURGOGNE	ROUGE

VINS	CRU	MILL	REGION	COULEUR
	Chenas	1983	Beaujolais	Rouge
	Tavel	1986	Rhône	Rose



Division (\div)

- PERMET DE RECHERCHER DANS UNE RELATION, L'ENSEMBLE DES 'SOUS TUPLES' QUI SATISFONT UNE 'SOUS-RELATION'
 - inverse du produit cartésien
- Le quotient de la relation $D(A_1, \dots A_p, A_{p+1} \dots A_n)$ par la relation $d(A_{p+1} \dots A_n)$ est la relation $Q(A_1, \dots A_p)$ dont les tuples sont ceux qui concaténés à tout tuple de d donnent un tuple de D .
 - Notations $\rightarrow D \div d = Q$

Division (\div) - Exemple

Compte	nomAgence	nomClient
	ParisAuteuil	Dupont
	NantesCentre	Queffelec
	Bellecourt	Girard
	Brotteaux	Letailleur
	LaDoua	Girard
	ParisDupleix	Dutour
	NantesCentre	Passard
	NantesCentre	Martin
	Bellecourt	Letailleur
	Brotteaux	Girard
	LaDoua	Letailleur

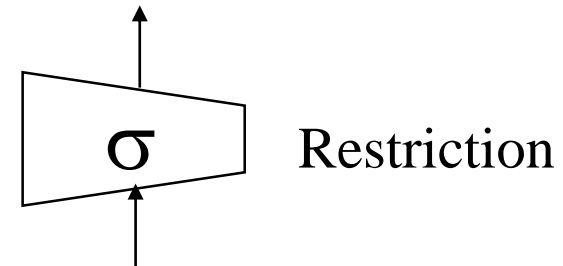
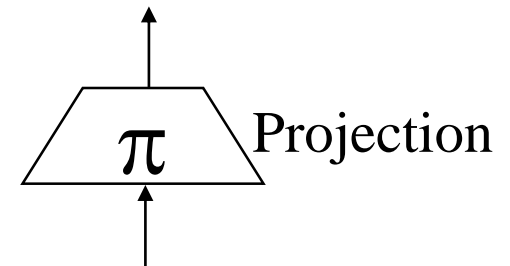
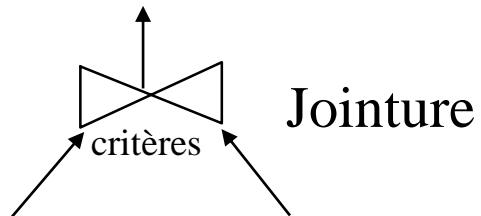
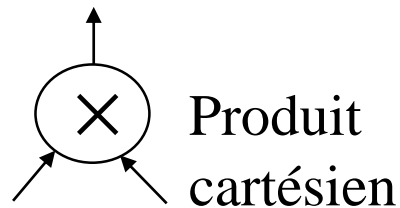
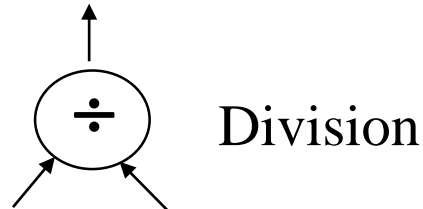
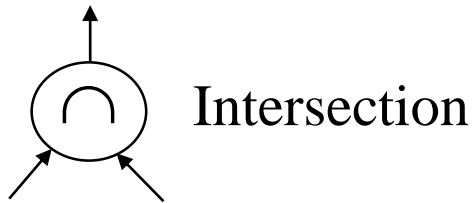
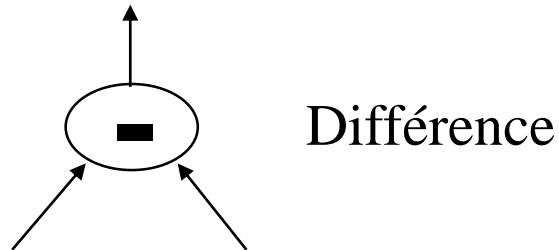
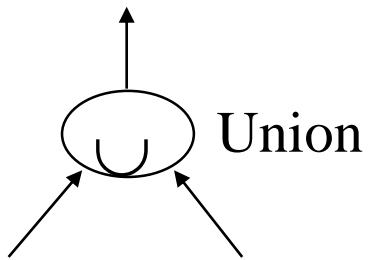
\div

Agence	NomAgence
	Bellecourt
	Brotteaux
	LaDoua

=

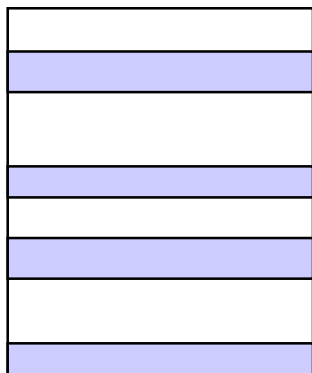
Resultat	nomClient
	Girard
	Letailleur

Représentation graphique

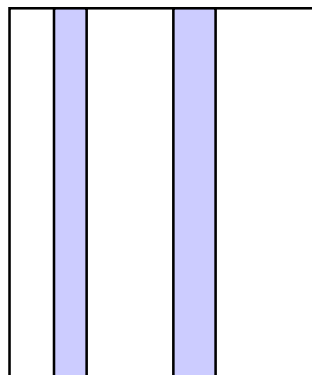


Récapitulatif

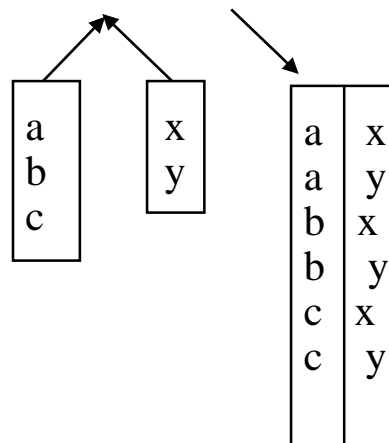
Sélection



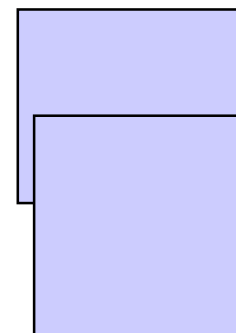
Projection



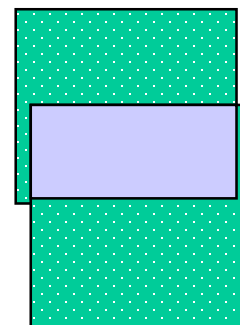
Produit



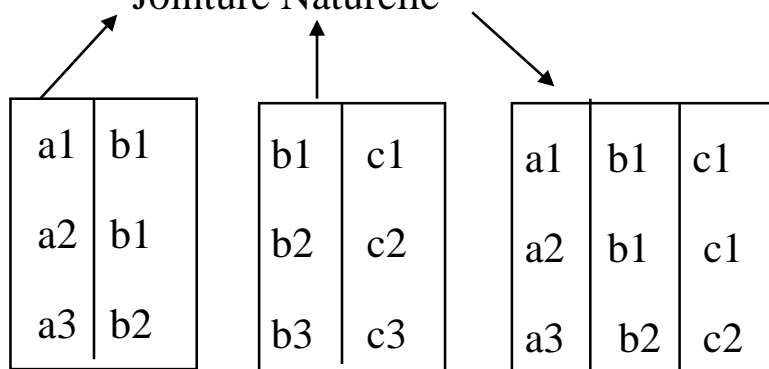
Union



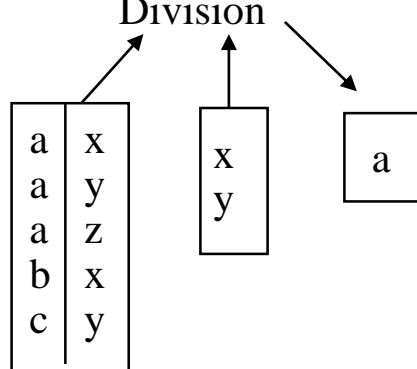
Intersection



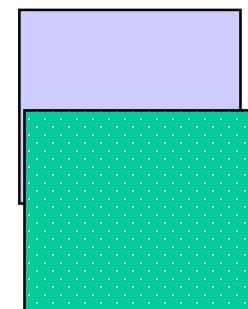
Jointure Naturelle



Division



Différence



Autres opérateurs

Renommage (ρ)

Permet de renommer des attributs

Notation : $\rho_{a1:b1, \dots, an:bn} (Rel)$

Affectation (\leftarrow)

Affecte une expression à une relation temporaire

Notation : $Temp \leftarrow Expression_relationnelle$

Exemple : $Temp \leftarrow r - S$

SemiJoin, AntiJoin, OuterJoin

- SemiJoin



Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee \bowtie Dept

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

- AntiJoin



Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee $\not\bowtie$ Dept

Name	EmpId	DeptName
Harry	3415	Finance
George	3401	Finance

- Left OuterJoin



Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee \bowtie X Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	ω
Sally	2241	Sales	Harriet
George	3401	Finance	ω
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω

- Right OuterJoin



Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee X \bowtie Dept

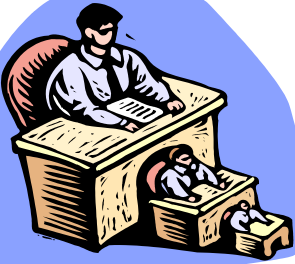
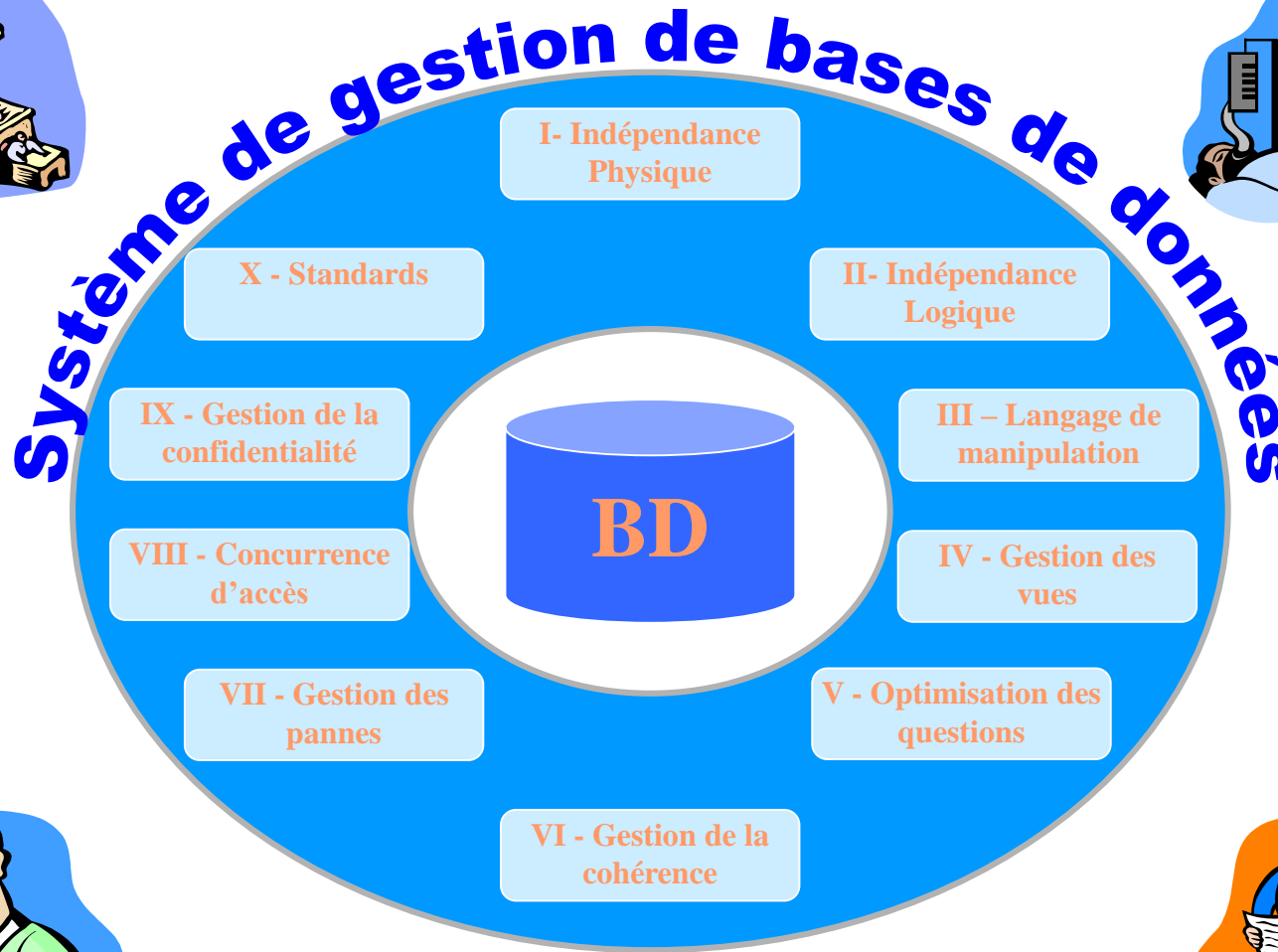
Name	EmpId	DeptName	Manager
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
ω	ω	Production	Charles

TD Conception (Passage au MLD)

TD algèbre relationnelle

Fonctionnalités des bases de données (en 10 grands principes)

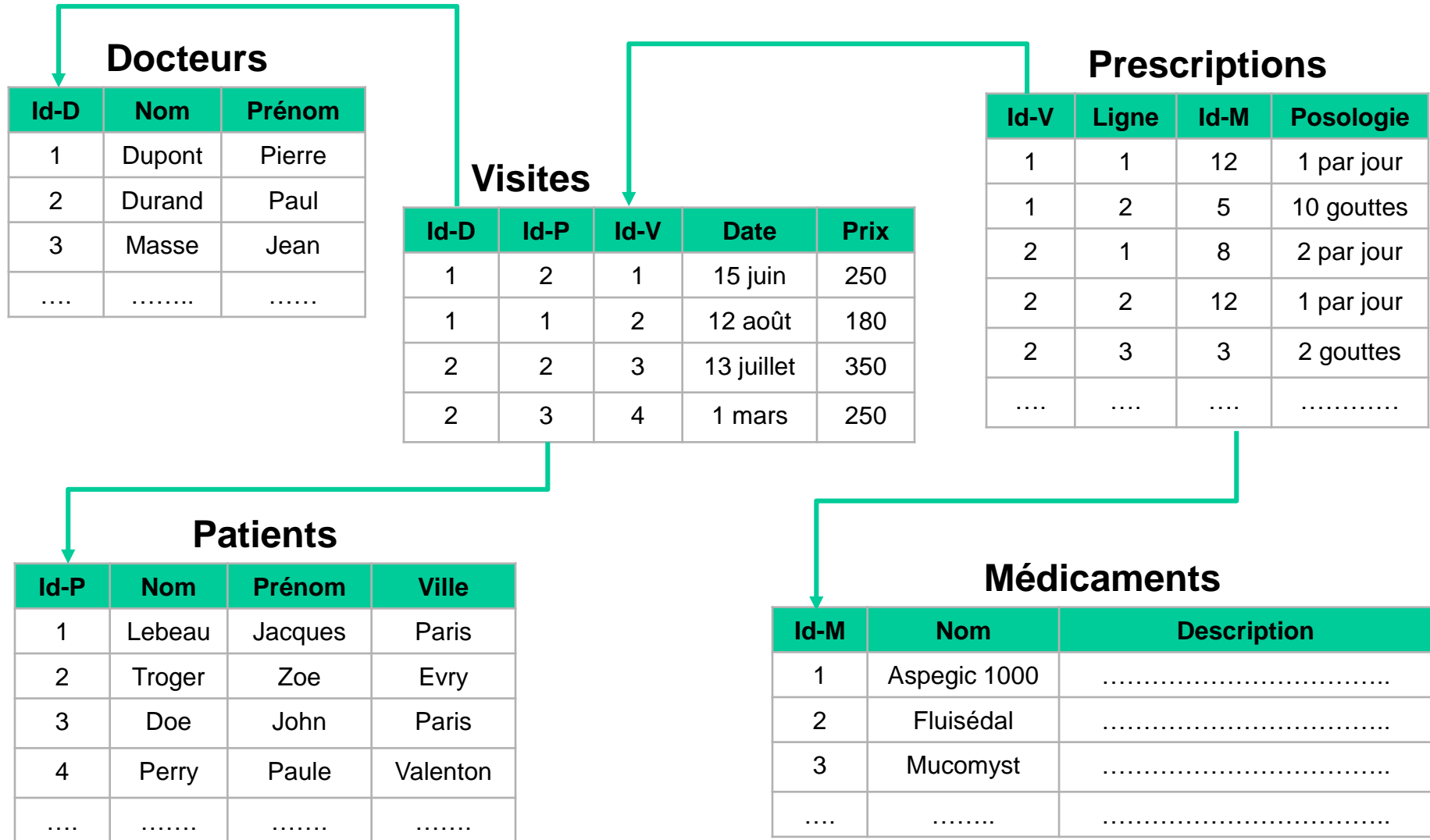
L'approche "Bases de données"



I - Indépendance Physique

- **Indépendance des programmes d'applications vis à vis du modèle physique :**
 - Possibilité de modifier les **structures de stockage** (fichiers, index, chemins d'accès, ...) sans modifier les programmes;
 - Ecriture des applications par des **non-spécialistes des fichiers** et des structures de stockage;
 - Meilleure **portabilité** des applications et **indépendance** vis à vis du matériel.

Modélisation Relationnelle



II - Indépendance Logique

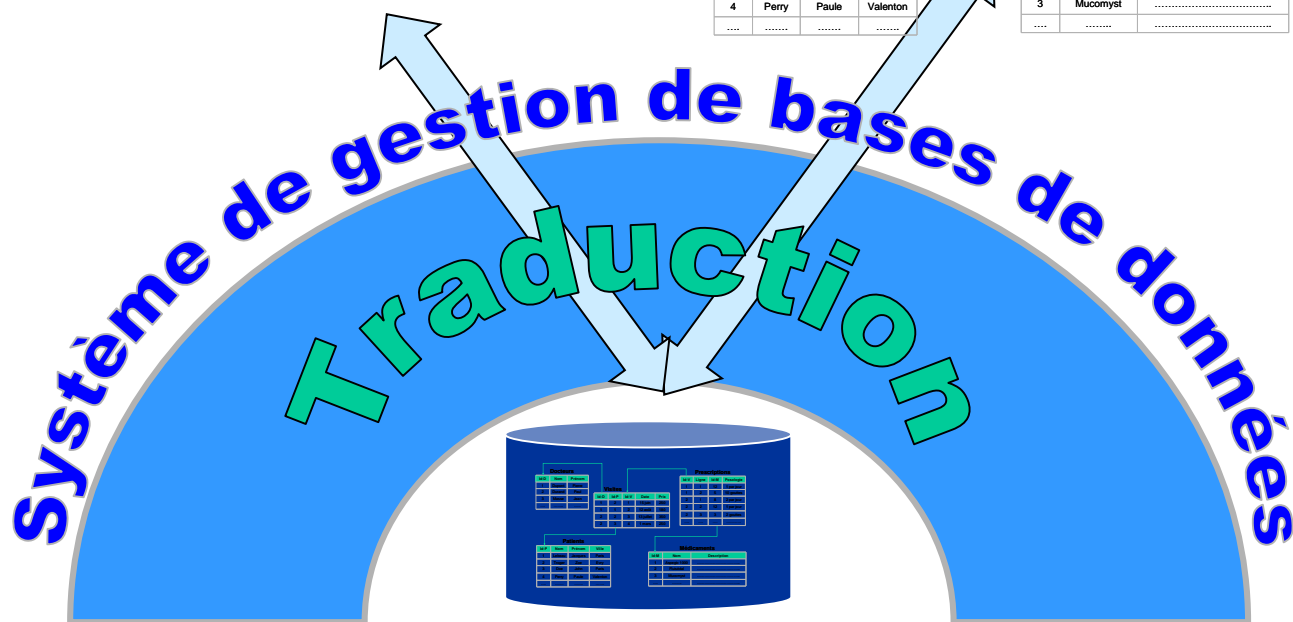
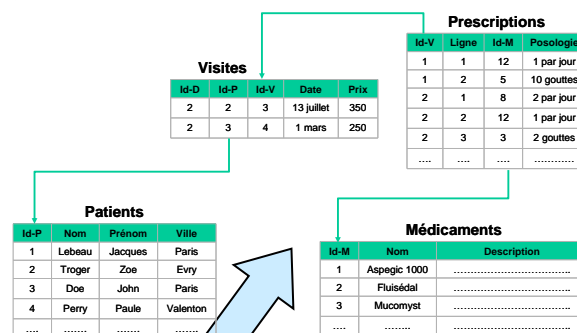
Les applications peuvent définir des **vues logiques** de la BD

Gestion des médicaments

Nombre_Médicaments

Id-M	Nom	Description	Nombre
1	Aspegic 1000	30
2	Fluisédal	20
3	Mucomyst	230
....

Cabinet du Dr. Masse



Avantages de l'indépendance logique

- Possibilité pour chaque application **d'ignorer** les besoins des autres (bien que partageant la même BD).
- Possibilité **d'évolution de la base de données** sans réécriture des applications :
 - ajout de champs, ajout de relation, renommage de champs.
- Possibilité **d'intégrer des applications existantes** sans modifier les autres.
- Possibilité de limiter les conséquences du partage : **Données confidentielles.**

III - Manipulation aisée

- La manipulation se fait via un langage **déclaratif**
 - La question déclare l'objectif sans décrire la méthode
 - Le langage suit une norme commune à tous les SGBD
 - **SQL : Structured Query Language**
- Syntaxe (aperçu !)

Select <Liste de champs ou de calculs à afficher>

From <Liste de relations mises en jeu>

Where <Liste de prédicats à satisfaire>

Group By <Groupement éventuel sur un ou plusieurs champs>

Order By <Tri éventuel sur un ou plusieurs champs>

Exemple de question SQL (1)

- Nom et description des médicaments de type aspirine

Select	Nom, Description
From	Médicaments
Where	Type = 'Aspirine'

Exemple de question SQL (2)

- Patients parisiens ayant effectués une visite le 15 juin

```
Select Patients.Nom, Patients.Prénom  
From Patients, Visites  
Where Patients.Id-P = Visites.Id-P  
and Patients.Ville = 'Paris'  
and Visites.Date = '15 juin'
```

Exemple de question SQL (3)

- Dépenses effectuées par patient trié par ordre décroissant

```
Select    Patients.Id-P, Patients.Nom, sum(Prix)
From      Patients, Visites
Where     Patients.Id-P = Visites.Id-P
Group By  Patients.Id-P, Patients.Nom
Order By  sum(Prix) desc
```

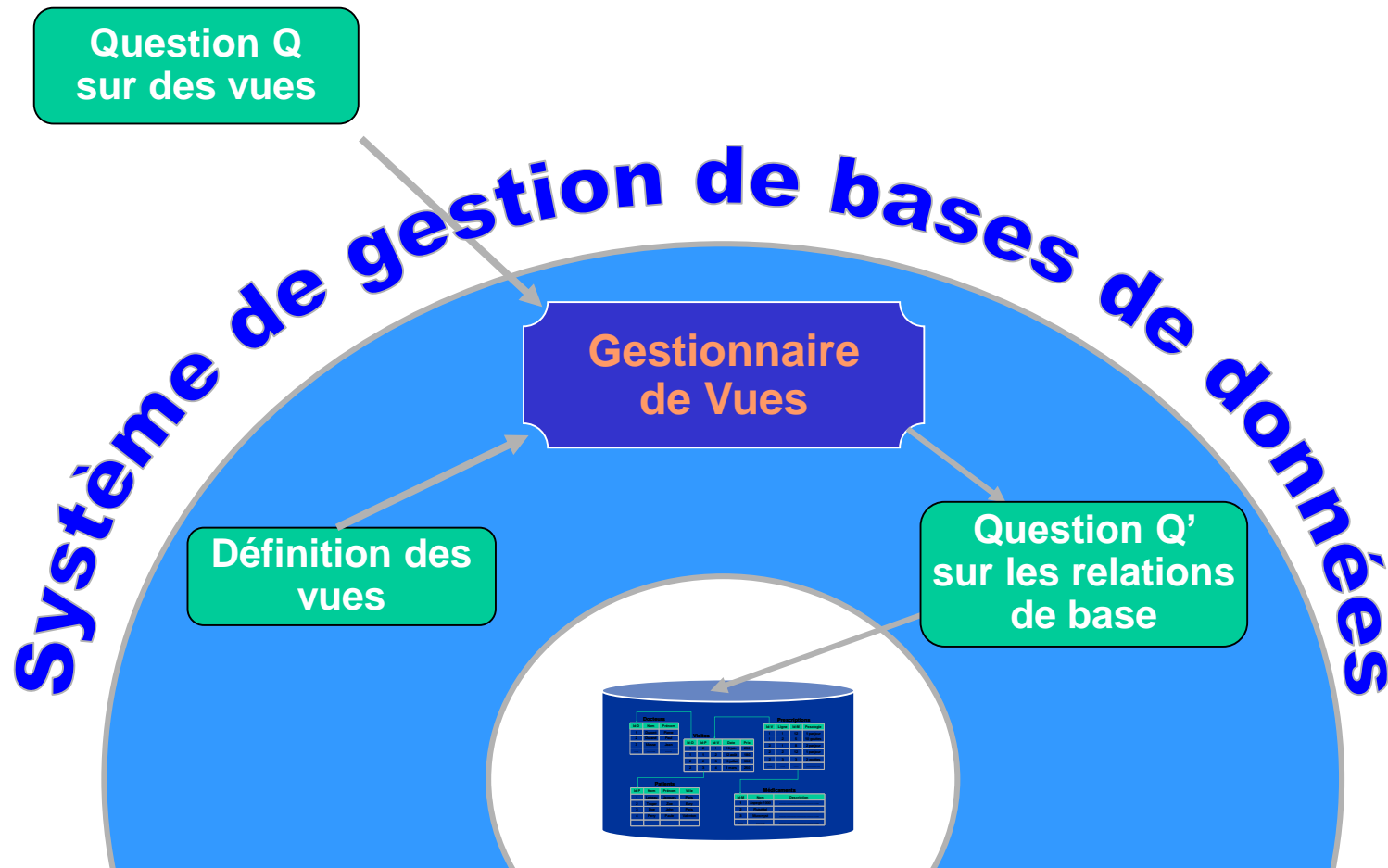
IV – Gestion des vues

- Les vues permettent d'implémenter l'indépendance logique en permettant de créer des **objets virtuels**
- Vue = Question SQL stockée
- Le SGBD stocke la **définition** et non le résultat
- Exemple : la vue des patients parisiens

```
Create View Parisiens as (  
    Select      Nom, Prénom  
    From        Patients  
    Where       Patients.Ville = 'Paris' )
```

Les vues : des relations virtuelles !

Le SGBD **transforme** la question sur les vues en question sur les relations de base



Les vues : Mise à jour

- Non définie si la répercussion de la mise à jour vers la base de données est ambiguë
 - ajouter un tuple à la vue calculant le nombre de médicaments ?
- Restrictions SQL (norme):
 - Pas de distinct, d'agrégats, ni d'expression
 - La vue contient les clés et les attributs « non nulls »
 - Il y a une seule table dans le from
 - Requêtes imbriquées possibles
 - Certains SGBDs supportent plus de mises à jour
- Clause « With check option »
 - Le SGBD vérifie que les tuples insérés ou mis à jour correspondent à la définition de la vue

Les vues : Les instantanés (snapshot)

- Instantané, Snapshot, vue concrète, vue matérialisée
 - matérialisée sur le disque
 - accessible seulement en lecture
 - peut être réactualisé
- Exemple
 - **create snapshot** Nombre_Médicaments **as**
Select Id-M, Nom, Description, count(*)
From Médicaments M, Prescriptions P
Where M.Id-M = P.Id-M
refresh every day
- Objectif principal : la performance

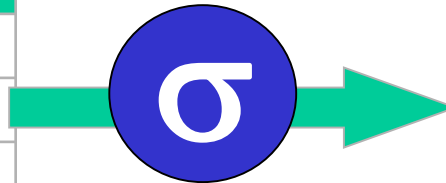
V – Exécution et Optimisation

- Traduction **automatique** des questions déclaratives en programmes procéduraux :
 - ➔ Utilisation de l'algèbre relationnelle
- Optimisation **automatique** des questions
 - ➔ Utilisation de l'aspect déclaratif de SQL
 - ➔ Gestion centralisée des chemins d'accès (index, hachages, ...)
 - ➔ Techniques d'optimisation poussées
- Economie de l'astuce des programmeurs
 - milliers d'heures d'écriture et de maintenance de logiciels.

Sélection

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

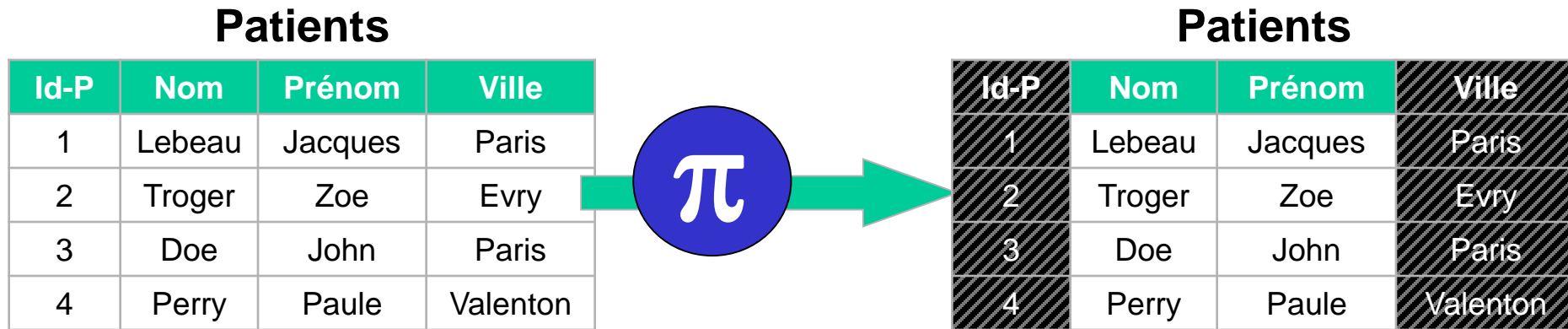


Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Patients de la ville de Paris

Projection



Nom et prénom des patients

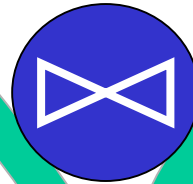
Jointure

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

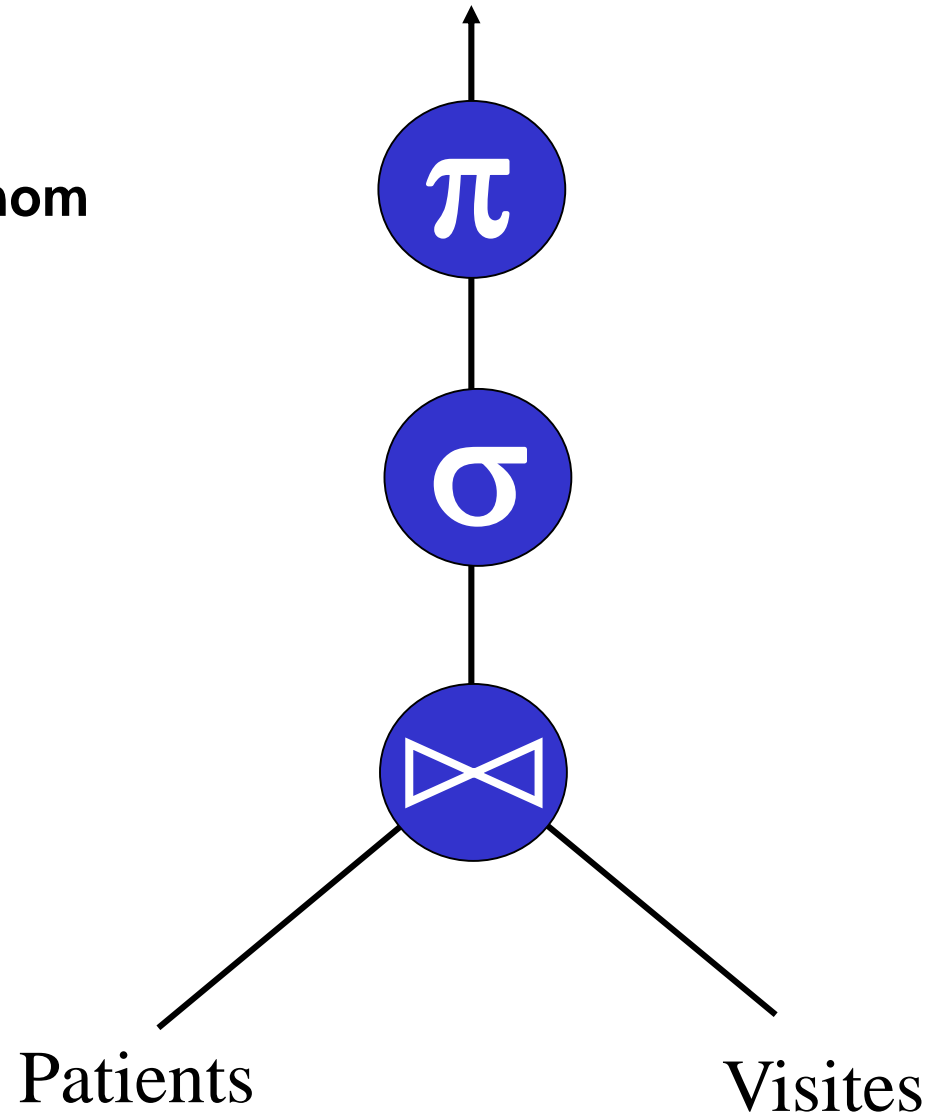


Id-P	Nom	Prénom	Ville	Id-D	Id-P	Id-V	Date	Prix
1	Lebeau	Jacques	Paris	1	1	2	12 août	180
2	Troger	Zoe	Evry	1	2	1	15 juin	250
2	Troger	Zoe	Evry	2	2	3	13 juillet	350
3	Doe	John	Paris	2	3	4	1 mars	250

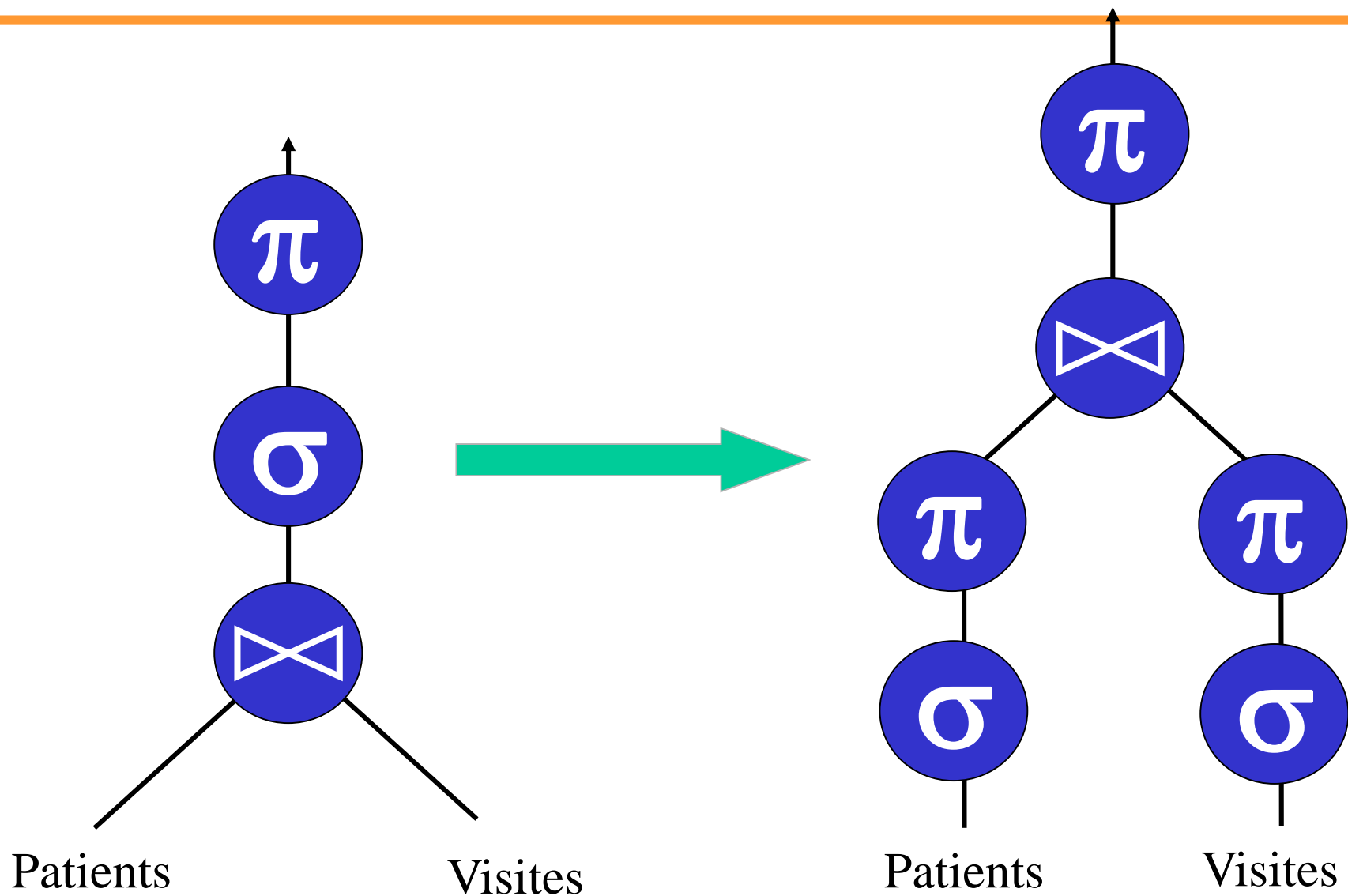
Patients et leurs visites

Exemple de plan d'exécution

Select Patients.Nom, Patients.Prénom
From Patients, Visites
Where Patients.Id-P = Visites.Id-P
and Patients.Ville = 'Paris'
and Visites.Date = '15 juin'



Plan d'exécution optimisé

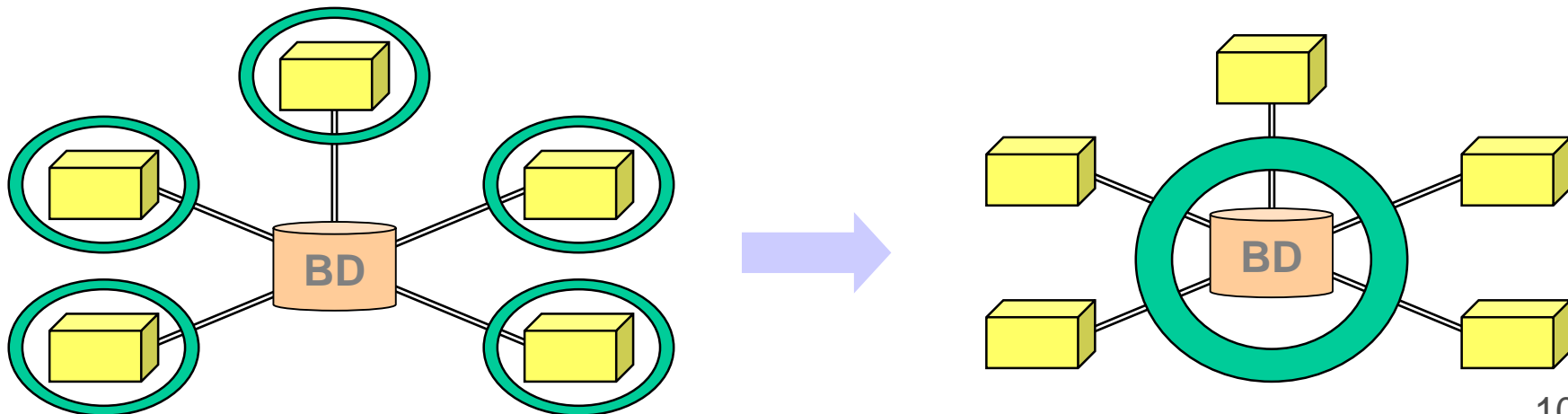


VI - Intégrité Logique

- **Objectif : Détecter les mises à jour erronées**
- Contrôle sur les données élémentaires
 - Contrôle de types: ex: Nom alphabétique
 - Contrôle de valeurs: ex: Salaire mensuel entre 5 et 50kf
- Contrôle sur les relations entre les données
 - Relations entre données élémentaires:
 - Prix de vente > Prix d'achat
 - Relations entre objets:
 - Un électeur doit être inscrit sur une seule liste électorale

Contraintes d'intégrité

- **Avantages :**
 - **simplification** du code des applications
 - **sécurité renforcée** par l'automatisation
 - **mise en commun** des contraintes, **cohérence**
- **Nécessite :**
 - un langage de définition de contraintes d'intégrité
 - la vérification **automatique** de ces contraintes

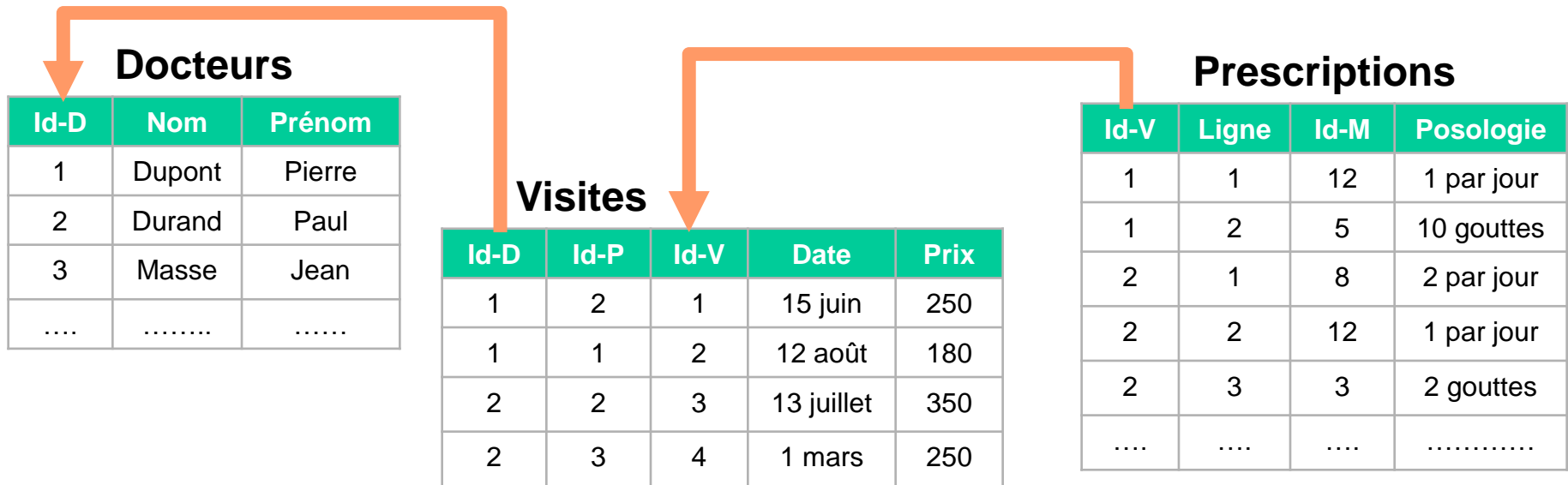


Types de contraintes

- Contrôle sur les données élémentaires
 - Contrôle de types : ex: Nom alphabétique
 - Contrôle de valeurs : ex: Salaire mensuel entre 1 et 10 K€
 - Non Nullité
- Contrôle sur les relations entre les données
 - Unicité
 - Relations entre données élémentaires : PrixVente>PrixAchat
 - Relations entre objets : Un électeur doit être inscrit sur une seule liste électorale
 - Contrainte d'intégrité référentielle : une visite doit être liée à un médecin et un patient !
- Les SGBD commerciaux supportent généralement peu de contraintes (par rapport à la norme SQL2)

Exemples de contrainte

- Contraintes d'intégrité référentielles



- Vérification lors de l'insertion, la suppression, la modification**
- Propagation des suppression, modification en cascade possible (on delete cascade)**

Contraintes d'intégrité : Syntaxe

create table <nom de table> (

<attribut> <domaine> [**<contrainte d'attribut>**], (mono-attribut)

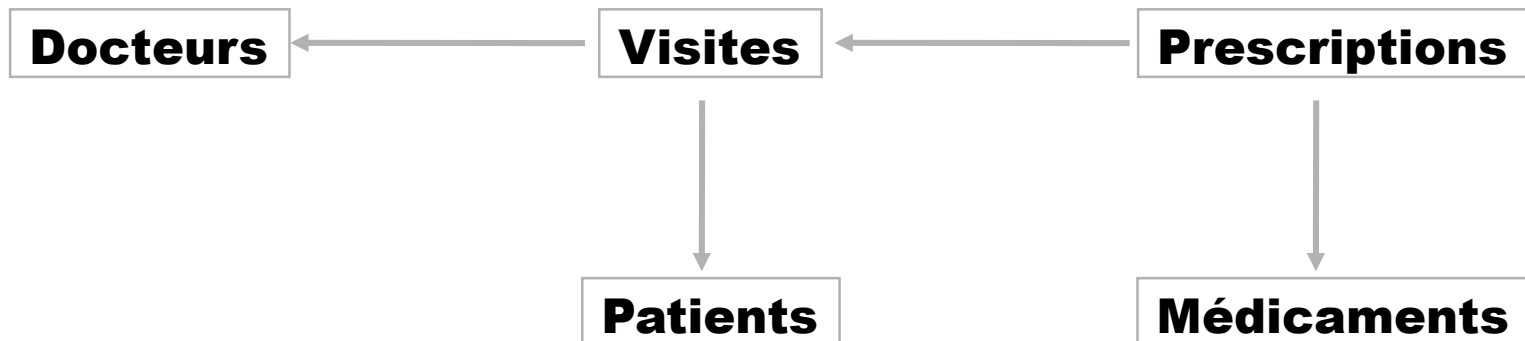
<attribut> <domaine> [**<contrainte d'attribut>**], ...

[**<contrainte de relation>**]) (mono ou multi-attributs)

- Différent types de contraintes :
 - Non nullité : **not null**
 - Unicité : **unique**
 - Vérification : **check <formule>**
 - Clé primaire : **primary key**
 - Contrainte d'intégrité référentielle : **references <relation> (<attribut>)**
 - **on delete / on update → cascade, set null, set default**
- On peut nommer les contraintes

Contraintes d'intégrité : Remarques finales

- Contraintes supportées (exemple d'Oracle)
 - Les contraintes faisant intervenir un ou plusieurs attributs **d'une seule table et d'un seul tuple**
 - Contraintes d'intégrité référentielles avec cascade
- Mécanisme puissant :
 - Exemple : supprimer les patients n'ayant pas consulté depuis 2 ans



Déclencheurs : Définition

- Définition : Déclencheurs ou Triggers
 - Règle E – C – A : Évènement – Condition – Action

Lorsque l'**évènement** se produit

- Insert / Update / Delete **pour une relation donnée**

si la **condition** est remplie

- Prédicat SQL optionnel

alors exécuter l'**action**

- Code à exécuter (ex. PLSQL sous Oracle)
- Pour chaque tuple concerné ou une fois pour l'évènement

Déclencheurs : Objectifs

- Objectif : une base de données 'active'
 - valider les données entrées
 - créer un audit de la base de données
 - dériver des données additionnelles
 - maintenir des règles d'intégrité complexes
 - implanter des règles métier
 - supporter des alertes (envoi de e-mails par exemple)
- Gains
 - simplification du code des applications
 - sécurité renforcée par l'automatisation
 - les déclencheurs sont stockées dans la base
 - Cohérence globale des déclencheurs

Déclencheurs : Syntaxe (dans Oracle)

Create trigger <nom de trigger>

before | after permet d'indiquer quand le trigger va être exécuté

insert | delete | update [of <attributs>] Quel est l'évènement déclencheur

on <relation> indique le nom de la table qui doit être surveillée

[referencing old as <var>, new as <var>] en SQL3, Oracle utilise :new et :old

for each row Précise si l'action est exécuté 1 fois par tuple concerné ou pour toute la table

[when <condition>] permet d'indiquer une condition pour l'exécution du trigger

DECLARE Déclaration de variables pour le bloc PL/SQL

BEGIN Bloc PL/SQL contenant le code de l'action à exécuter

 <PL/SQL bloc> Dans SQL3, on peut indiquer une suite de commande SQL

END

/!\ La syntaxe diffère légèrement suivant le SGBD

Déclencheurs : Exemples simples

Create trigger calcul_TTC **after insert on** Vente

Begin

 update vente set Prix_TTC = Prix_HT*1.206

End ;

Create trigger ModifCommande **after update on** Commande

For each row

Begin

 if :new.qte < :old.qte

 then raise_application_error(-9996, ' La quantité ne peut diminuer');

End ;

Create trigger ModifCommande **after update on** Commande

For each row

When (new.qte < old.qte)

Begin

 raise_application_error(-9996, , ' La quantité ne peut diminuer');

End ;

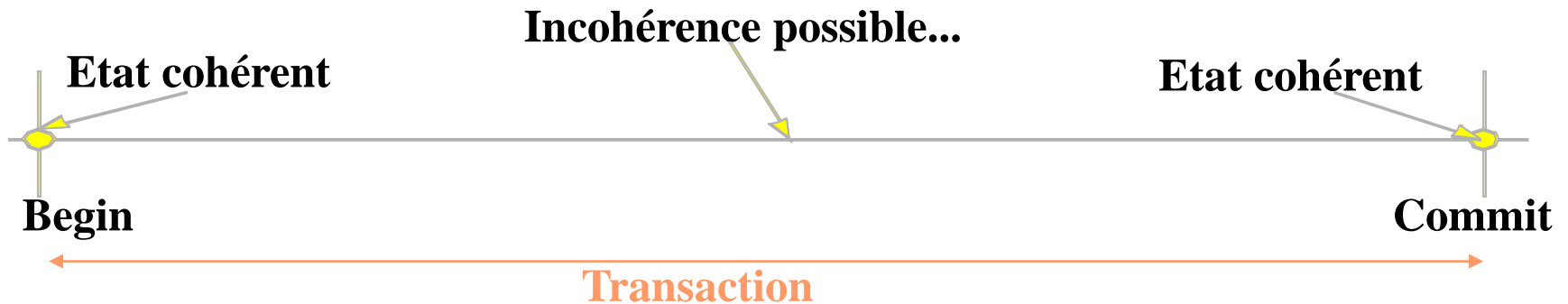
Déclencheurs : Remarques finales

- Cascade de triggers
 - l'action d'un trigger peut déclencher d'autres triggers
- Interactions avec les contraintes
 - l'action d'un trigger peut causer la vérification des contraintes
 - les actions des contraintes référentielles peuvent déclencher des triggers (delete cascade, update cascade)
- Mécanisme très (trop ?) puissant
 - Cascade 'infinie'
 - Tables en 'mutation'
 - ➔ Usage limité

VII - Intégrité Physique

- **Motivations : Tolérance aux fautes**
 - Transaction Failure : Contraintes d'intégrité, Annulation
 - System Failure : Panne de courant, Crash serveur ...
 - Media Failure : Perte du disque
 - Communication Failure : Défaillance du réseau
- **Objectifs :**
 - Assurer l'**atomicité** des transactions
 - Garantir la **durabilité** des effets des transactions commises
- **Moyens :**
 - Journalisation : Mémorisation des **états successifs** des données
 - Mécanismes de reprise

Transactions



Begin

$CE_{pargne} = CE_{pargne} - 3000$

$CC_{courant} = CC_{courant} + 3000$

Commit T1

Atomicité et Durabilité

ATOMICITE

Begin

$CEpargne = CEpargne - 3000$

$CCourant = CCourant + 3000$

Commit T1

Panne

→ Annuler le débit !!

DURABILITE

Begin

$CEpargne = CEpargne - 3000$

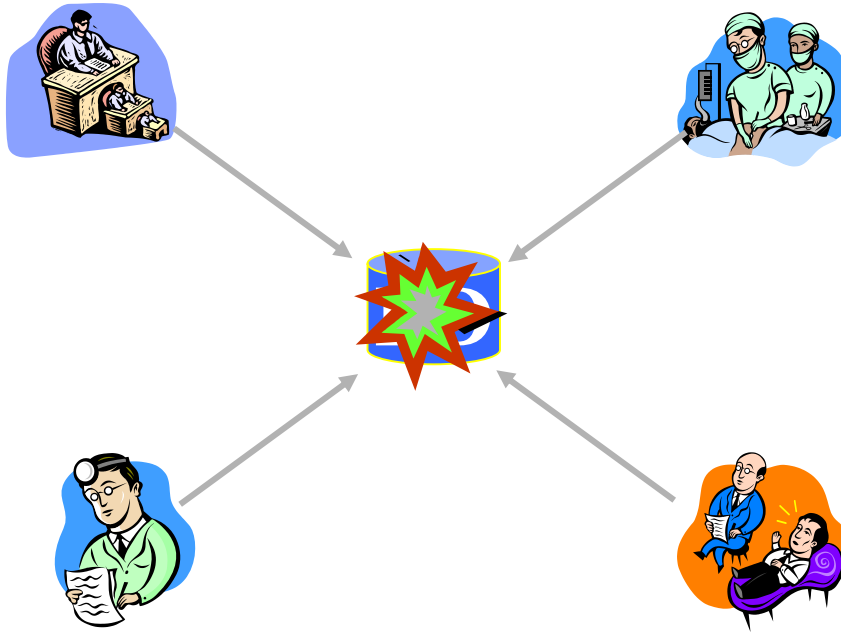
$CCourant = CCourant + 3000$

Commit T1

Crash disque

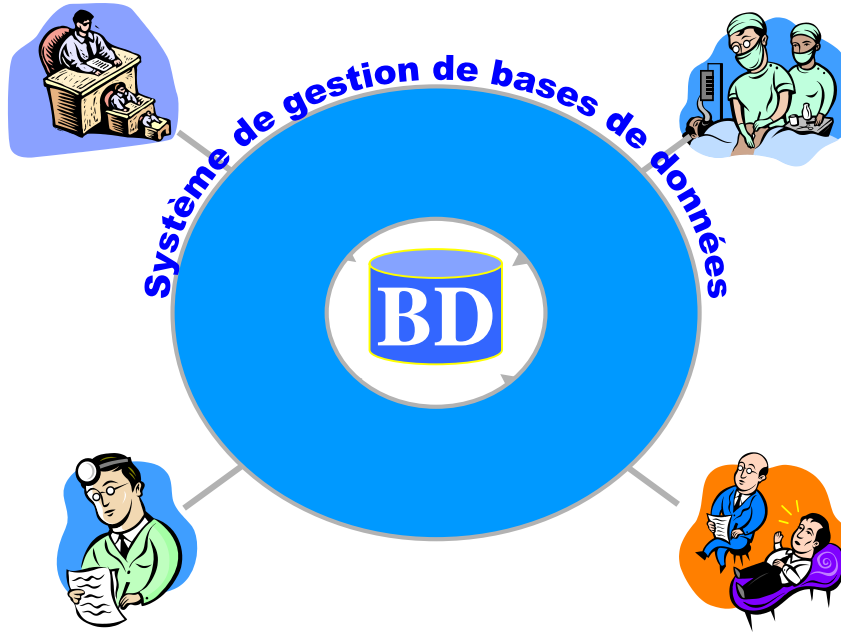
→ S'assurer que le virement a été fait !

VIII - Partage des données



- Accès concurrent aux mêmes données
- ➔ Conflits d'accès !!

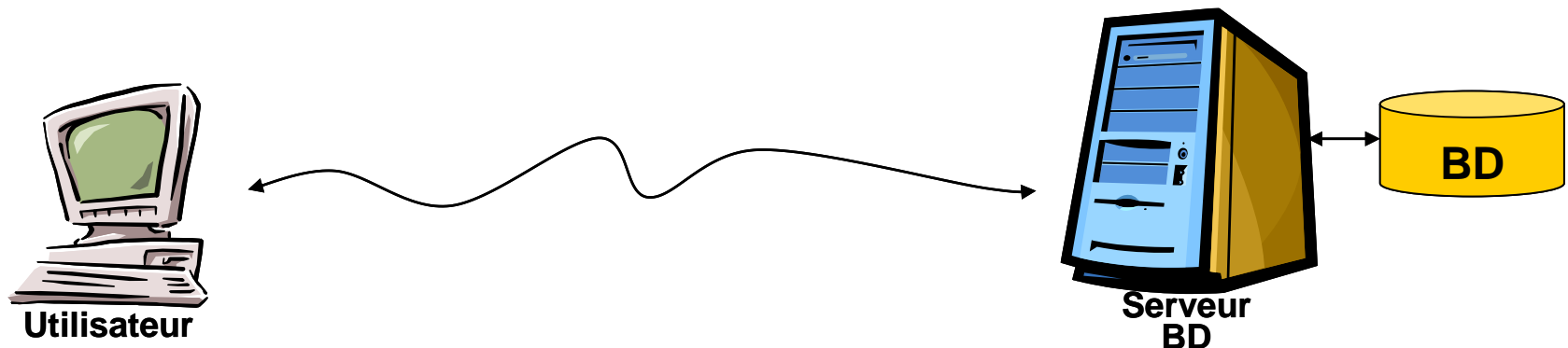
VIII - Partage des données



- Le SGBD gère les accès concurrents
 - ➔ Chacun à l'*impression* d'être seul (Isolation)
 - ➔ Cohérence conservée (Verrouillage)

IX – Confidentialité

- Objectif :
 - protéger les données de la base contre des accès non autorisés
- Mécanismes simples et puissants
 - Connexion restreinte aux usagers répertoriés (mot de passe)
 - Privilèges d'accès aux objets de la base (relation, vue, etc.)
- Repose sur la sécurité de l'architecture (de bout en bout)



**Identification
Authentification**

Sécurisation des communications
(confidentialité, intégrité, non répudiation)

**Protection du
serveur**

**Protection
des fichiers**

Droits d'accès : Syntaxe de base et rôles

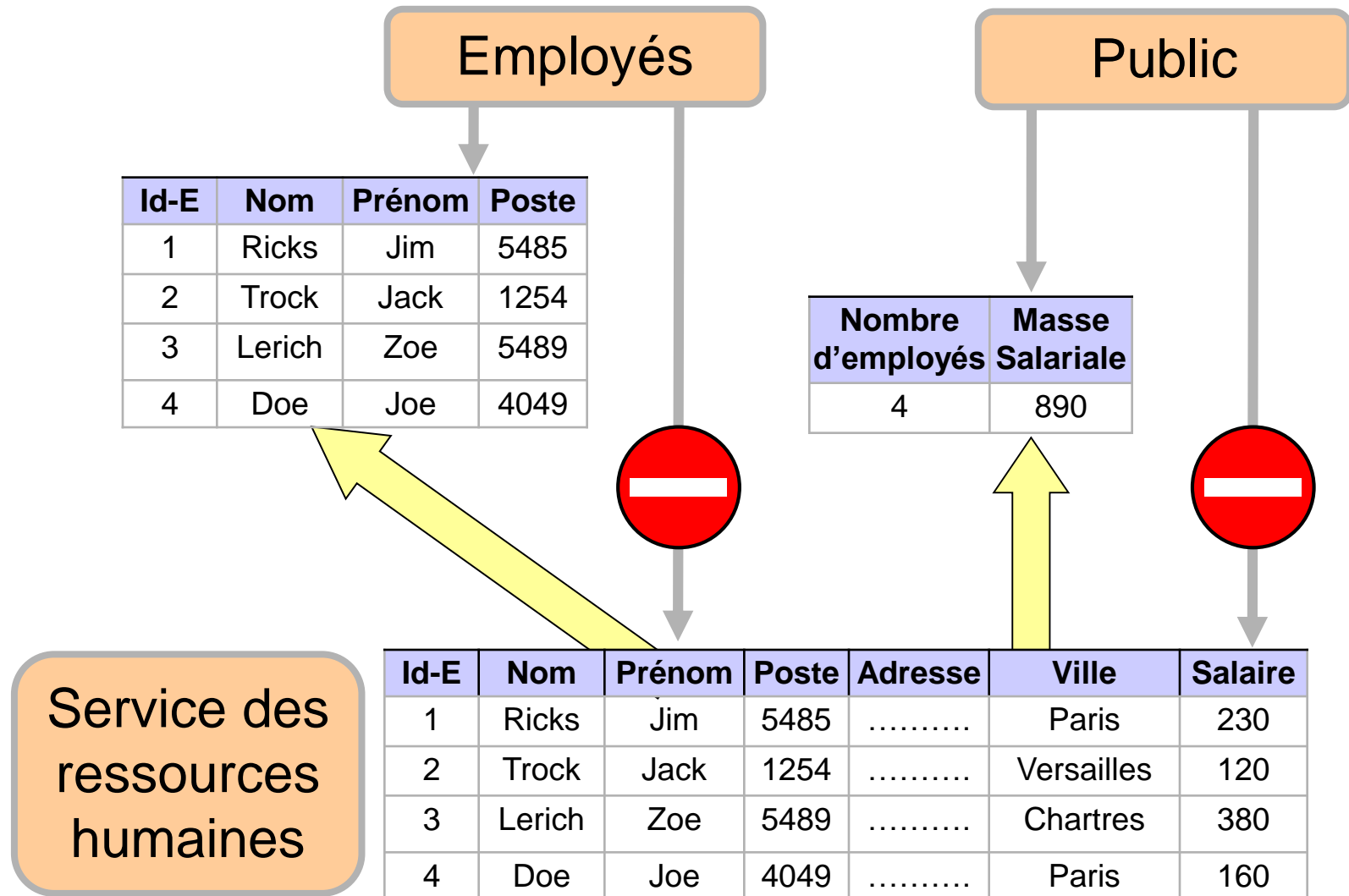
- Syntaxe de base

- **Grant** <droits> **on** <objet> **to** (<usager>*) [**with grant option**]
- **Revoke** <droits> **on** <objet> **from** (<usager>*)
- <droits> ::= all | select | insert | delete | update | alter
- <objet> ::= relation | vue | procedure
- <usager> ::= public | nom d'usager

- Rôles (SQL3):

- **Create role** <nom_role> : Création d'un nouveau rôle
- **Drop role** <nom_role> : Suppression d'un rôle
- Les instructions **Grant** et **Revoke** s'appliquent sur des rôles
- **Grant** <liste roles> **to** (<usager>*) : Affectation de rôles aux usagers
- **Set role** <liste_roles> : Activation de rôle(s) pendant une session

Droits d'accès : droits sur les vues



Droits d'accès : droits évolués

- Règles indépendantes du contenu :
 - Ex. la secrétaire a accès aux infos administratives
 - grant ou revoke sur relation ou vue
- Règles dépendantes du contenu :
 - Ex. Le médecin a accès aux dossier de ses patients
 - Utilisation de vues paramétrées (e.g. avec Current_User)
- Règles dépendantes du contexte :
 - Ex. en l'absence du médecin traitant, tout médecin a accès au dossier du patient
 - Utilisation de vues paramétrées avec tables contextuelles
 - Ex. Les salaires ne peuvent être modifié que le 1^{er} du mois
 - Utilisation de procédures stockées

X - Standardisation

- L'approche bases de données est basée sur plusieurs standards
 - Langage SQL (SQL1, SQL2, SQL3)
 - Communication SQL CLI (ODBC / JDBC)
 - Transactions (X/Open DTP, OSI-TP)
- Force des standards
 - Portabilité
 - Interopérabilité
 - Applications multi sources...

Applications traditionnelles des SGBD

- **OLTP** (On Line Transaction Processing)
 - Cible des SGBD depuis leur existence
 - Banques, réservation en ligne ...
 - **Très grand nombre de transactions** en parallèle
 - Transactions **simples**
- **OLAP** (On Line Analytical Processing)
 - Entrepôts de données, DataCube, Data Mining ...
 - **Faible nombre de transactions**
 - Transactions très **complexes**

Problématiques Web et Big Data

- Traitements Big Data
 - Découvertes de règles associatives, dépendances fonctionnelles, time series, learning, IA ...
- Gestion de données peu structurées
 - Semi-structurées : documents XML/Json, KVS, NoSQL...
 - Non-structurées : index inversé, full text search...
- Intégration
 - De données : intégration de schémas, RDF, requêtes continues...
 - De programmes : workflows, optimisation des flux...
- Mobilité
 - BD de localisation, BD embarquées, réseaux de capteurs ...
- Sécurité des bases de données
 - Contrôle d'accès, contrôle d'usage, anonymisation, chiffrement, hardware sécurisé (Intel SGX, ARM Trustzone), Private Information Retrieval...

Le langage SQL

Structured Query Langage

SQL : pour quoi faire (1) ?

- Avant SQL, rechercher des données satisfaisant une qualification ressemblait à :

Quelles sont les plages où se sont baignés d'excellents nageurs pendant le mois d'août 89 ?

```
READY F_baignade, F_nageur  
FIND FIRST nageur WITHIN F_nageur  
PERFORM UNTIL fin-de-fichier  
  GET nageur  
  IF qualité IN nageur = 'excellent'  
    FIND FIRST baignade WITHIN bain  
    PERFORM UNTIL fin-de l'ensemble  
      GET baignade  
      IF date BETWEEN 01/01/89 AND 31/08/89  
        FIND OWNER WITHIN lieu  
        GET plage  
        PRINT NOMP  
      END_IF  
    FIND NEXT baignade  
  END_PERFORM  
END_IF  
FIND NEXT nageur WITHIN F_nageur  
END_PERFORM  
FINISH F_plage, F_baignade, F_nageur.
```

```
Select B.plage  
From Nageurs N, Baignade B  
Where N.qualité = 'excellent'  
and B.date between '01-08-89'  
and '31-08-89' and N.idn = B.idn
```


SQL : pour quoi faire (2) ?

- Une interface plus simple pour les utilisateurs ?
 - Manipulations via des formulaires !
 - Mais rend possible les requêtes libres (ex: analyse)
- Simplifie le développement des applications
 - Code plus compact et facile à valider/maintenir
 - Indépendance physique et logique des programmes par rapport aux données
 - Optimisation automatique et dynamique des requêtes par le SGBD
- Une puissance d'expression exploitable de nb façons
 - Sélection, mises à jour, intégrité, droits d'accès, audit ...

Le standard SQL (1)

- Trois versions normalisées de SQL :
 - SQL1 (1989) : addendum (intégrité)
 - SQL2 (1992) : version étendue à trois niveaux de conformité (entry, intermediate, full)
 - SQL3 (2000) : version étendue à la gestion d'objets complexes (XML)
 - SQL3 (2008) : introduction de fonctions de manipulation XML/Json...

Le standard SQL (2)

LANGAGE DE DEFINITION DE DONNEES

CREATE TABLE

CREATE VIEW

ALTER

...

LANGAGE DE CONTROLE

GRANT

REVOKE

COMMIT WORK

ROLLBACK WORK

LANGAGE DE MANIPULATION DE DONNEES

SELECT

INSERT

UPDATE

DELETE

INTEGRATION AUX LANGAGES DE PROGRAMMATION

EXEC SQL

MODULE

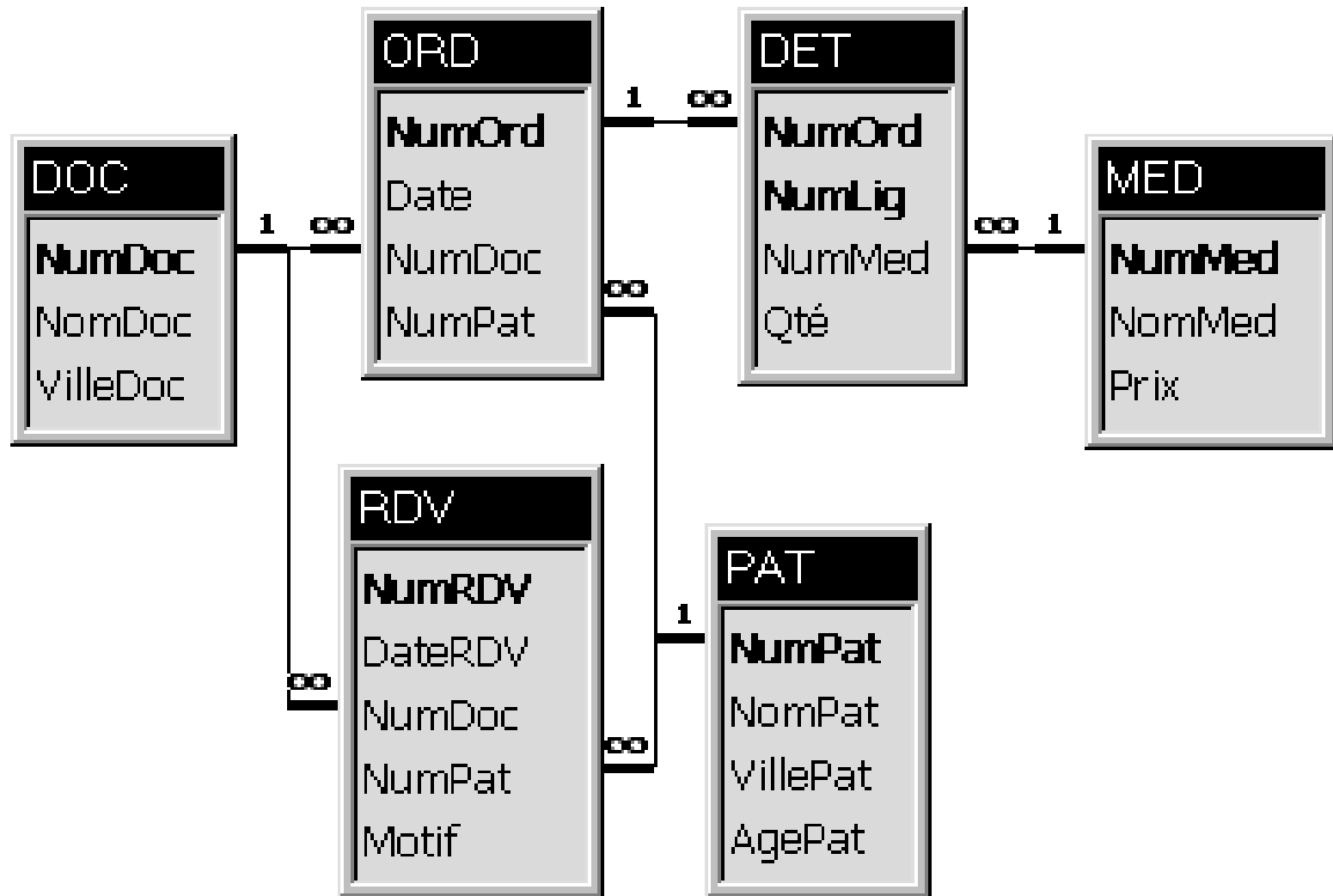
PROCEDURE ...

Le langage SQL (1/3)

LDD : Langage de Définition de Données

Création, modification du schéma

EXEMPLE DE BASE DE DONNEES



Création de table

```
CREATE TABLE <nom_table>  
    (<def_colonne> * [<def_contrainte_table>*]) ;
```

```
< def_colonne > ::= <nom_colonne> < type > [CONSTRAINT nom_contrainte <  
    NOT NULL |  
    UNIQUE |  
    PRIMARY KEY |  
    CHECK (condition) |  
    REFERENCES nom_table (colonne) > ]
```

```
< def_contrainte_table > ::= CONSTRAINT nom_contrainte <  
    UNIQUE (liste_colonnes) |  
    PRIMARY KEY (liste_colonnes) |  
    CHECK (condition) |  
    FOREIGN KEY (liste_colonnes) REFERENCES nom_table (liste_colonnes)>
```

Exemple de création de table

```
CREATE TABLE RDV(  
    NumRdv Integer,  
    DateRDV Date,  
    NumDoc Integer,  
    NumPat Integer,  
    Motif Varchar(200),  
    CONSTRAINT Clé_Primaire_RDV PRIMARY KEY (NumRdv),  
    CONSTRAINT Référence_DOC FOREIGN KEY (NumDoc) REFERENCES  
    DOC,  
    CONSTRAINT Référence_PAT FOREIGN KEY (NumPat) REFERENCES PAT);
```

L'association d'un nom à une contrainte est optionnelle. Ce nom peut être utilisé pour référencer la contrainte (ex: messages d'erreurs).

- Exercice 1
 - Donnez l'expression SQL de la création des tables DOC et DET

Index, modification du schéma

- Création d'index
 - `CREATE [UNIQUE] INDEX nom_index ON nom_table (<nom_colonne> *);`
- Suppression
 - `DROP TABLE <nom_table>`
 - `DROP INDEX <nom_index>`
- Modification
 - `ALTER TABLE <nom_table> ADD COLUMN <def_colonne>`
 - `ALTER TABLE <nom_table> ADD CONSTRAINT <def_contrainte_table >`
 - `ALTER TABLE <nom_table> MODIFY <def_colonne>`
 - `ALTER TABLE <nom_table> DROP COLUMN <nom_colonne>`
 - `ALTER TABLE <nom_table> DROP CONSTRAINT <nom_contrainte >`
- Exemples
 - `CREATE INDEX Index_date_RDV ON RDV (DateRDV) ;`
 - `ALTER TABLE RDV ADD COLUMN Commentaires varchar(300);`
 - `ALTER TABLE RDV ADD CONSTRAINT MotifNN NOTNULL(Motif);`
- Exercice 2
 - Créez un index sur le nom du docteur
 - Supprimez l'attribut Motif de la table RDV
 - Ajoutez une contrainte de clé primaire à la table MED (sur NumMed)

Le Langage SQL (2/3)

LMD : Langage de Manipulation des Données

(INSERT, DELETE , UPDATE)

Insertion de données : INSERT

```
INSERT INTO < nom_table >  
    [( attribute [,attribute] ... )]  
    {VALUES (<value spec.> [, <value spec.>] ... ) |  
     <query specification>} ;
```

- Exemples :
 - INSERT INTO DOC VALUES (1, 'Dupont', 'Paris');
 - INSERT INTO DOC (NumDoc, NomDoc) VALUES (2, 'Toto');
 - INSERT INTO PAT (NumPat, NomPat, VillePat)
 SELECT NumDoc, NomDoc, VilleDoc FROM DOC;
- Exercice 3
 - Insérez un ensemble cohérent de tuples dans chaque table

Mise à jour : UPDATE

SYNTAXE :

```
UPDATE      <relation_name>
SET         <attribute> = value_expression [, <attribute> =
value_expression ] ...
[WHERE      <search condition> ];
```

EXEMPLES :

Mettre "Inconnue" quand VilleDoc n'est pas renseignée

```
UPDATE DOC SET VilleDoc = "Inconnue" WHERE VilleDoc IS NULL
```

Mettre en majuscule le nom des docteurs qui n'ont jamais prescrit

```
UPDATE DOC SET NomDoc = UPPER(NomDoc) WHERE NumDoc NOT IN
(SELECT NumDoc FROM ORD)
```

ATTENTION AUX CONTRAINTES D'INTEGRITE REFERENTIELLES !!!

EXERCICE 4

- Augmenter le prix des aspirines de 10%
- Fixer le prix des médicaments non prescrits à 100 euros

Suppression : DELETE

SYNTAXE :

DELETE FROM <relation_name>
[WHERE <search_condition>];

EXEMPLES :

Supprimer les docteurs quand VilleDoc n'est pas renseignée

DELETE FROM **DOC** WHERE **VilleDoc** IS NULL

Supprimer les docteurs qui n'ont jamais prescrit

DELETE FROM **DOC** WHERE **NumDoc** NOT IN (SELECT **NumDoc** FROM **ORD**)

ATTENTION AUX CONTRAINTES D'INTEGRITE REFERENTIELLES !!!

EXERCICE 5

- Supprimer les patients de moins de 10 ans
- Supprimer les médicaments non prescrits

Correction exercice 1

Donnez l'expression SQL de la création des tables DOC et DET

```
CREATE TABLE DOC(  
    NumDoc integer,  
    NomDoc char(30),  
    VilleDoc varchar(50),  
    CONSTRAINT Clé_Primaire_Doc PRIMARY KEY (NumDoc)) ;
```

```
CREATE TABLE DET(  
    NumOrd integer,  
    NumLig integer,  
    NumMed integer,  
    Qté integer,  
    CONSTRAINT Clé_Primaire_DET PRIMARY KEY (NumOrd, NumLig),  
    CONSTRAINT Référence_ORD FOREIGN KEY (NumOrd) REFERENCES ORD,  
    CONSTRAINT Référence_MED FOREIGN KEY (NumMed) REFERENCES MED);
```

Correction exercice 2

Créez un index sur le nom du docteur

```
CREATE INDEX Index_nom_docteur ON DOC (NomDoc) ;
```

Supprimez l'attribut Motif de la table RDV

```
ALTER TABLE RDV  
    DROP COLUMN Motif;
```

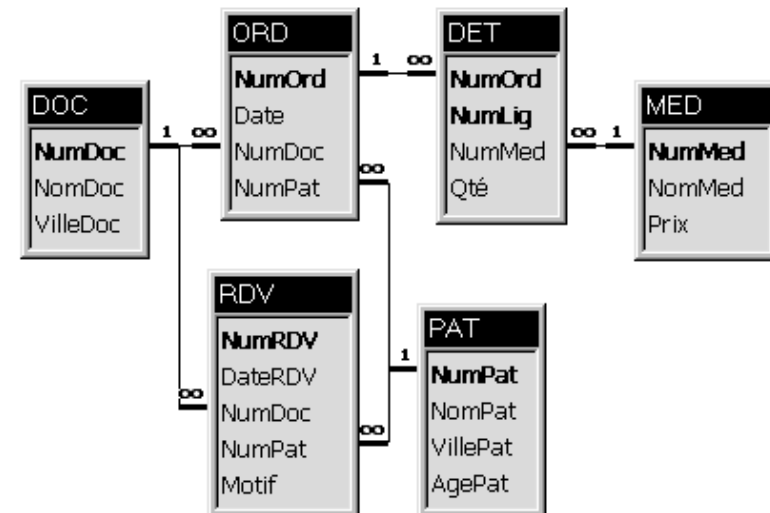
Ajoutez une contrainte de clé primaire à la table MED (sur NumMed)

```
ALTER TABLE MED  
    ADD CONSTRAINT cle_prim_MED PRIMARY KEY (NumMed) ;
```

Correction exercice 3

Insérez un ensemble cohérent de tuples dans chaque table (1 par table)

- INSERT INTO DOC VALUES (11, 'Dupont', 'Paris');
- INSERT INTO PAT VALUES (23,'Toto', 'Paris', 25)
- INSERT INTO MED VALUES (17, 'Aspirine', 4)
- INSERT INTO MED VALUES (18, 'Feroxite', 19)
- INSERT INTO RDV VALUES (45, #20/11/2006#, 11, 23, 'mal de tête')
- INSERT INTO ORD VALUES (38, #20/11/2006#, 11, 23)
- INSERT INTO DET VALUES (38, 1, 17, 3)
- INSERT INTO DET VALUES (38, 2, 18, 1)



Exercices 4 et 5 : correction

- **Augmenter le prix des aspirines de 10%**

UPDATE MED

SET Prix = Prix * 1.1

WHERE NomMed LIKE "aspirine%"

- **Fixer le prix des médicaments non prescrits à 100 euros**

UPDATE MED

SET Prix = 100

WHERE NumMed NOT IN (SELECT NumMed FROM DET);

- **Supprimer les patients de moins de 10 ans**

DELETE FROM PAT

WHERE AgePat < 10;

- **Supprimer les médicaments non prescrits**

DELETE FROM MED

WHERE NumMed NOT IN (SELECT NumMed FROM DET);

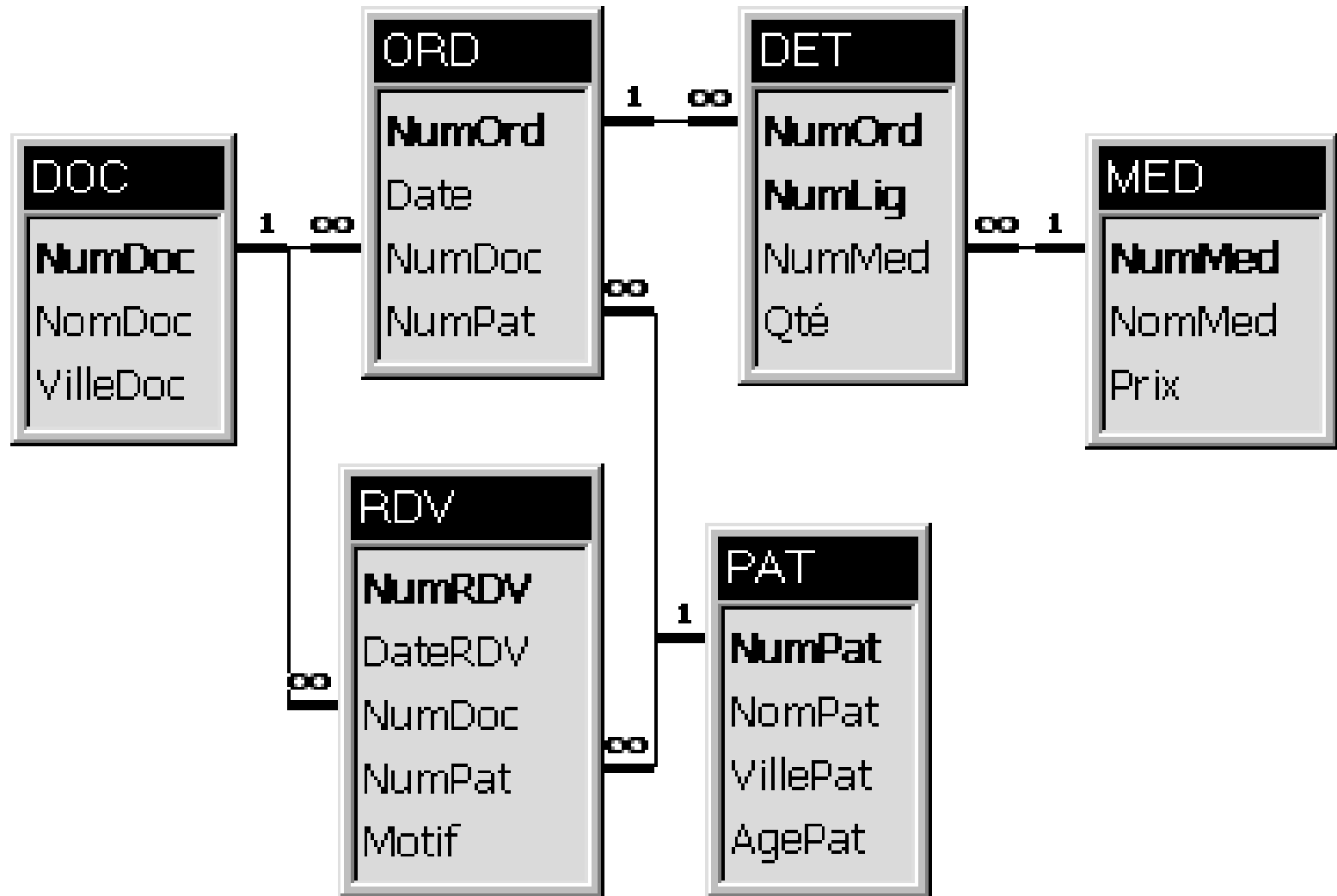
TD – Création d'une base, modifications, et chargement massif

Le Langage SQL (2/3)

LMD : Langage de Manipulation des Données

(SELECT)

EXEMPLE DE BASE DE DONNEES



SELECT : forme générale

SELECT [DISTINCT| ALL] { * | <value exp.> [, <value exp.>]...}

FROM relation [alias], relation [alias]...

[WHERE <search condition>

[GROUP BY <attribute> [,<attribute>]...]

[HAVING <search condition>

[ORDER BY <attribute> [{ASC | DESC}] [,<attribute>[{ASC | DESC}]]...]

* EXPRESSION DE VALEURS

- Calculs arithmétiques
- Fonctions agrégats

* CONDITION DE RECHERCHE

- Sélection, projection, jointure
- Recherche textuelle
- Recherche par intervalle
- Recherche sur valeur nulle

Forme générale de la condition de recherche

```
<search condition> ::= [NOT]
  <nom_colonne> IS [NOT] NULL
  <nom_colonne>  $\theta$  constante | <nom_colonne>
  <nom_colonne> LIKE <modèle_de_chîne>
  <nom_colonne> IN <liste_de_valeurs>
  <nom_colonne>  $\theta$  (ALL | ANY/SOME) <liste_de_valeurs>
  EXISTS <liste_de_valeurs>
  UNIQUE <liste_de_valeurs>
  <tuple> MATCH [UNIQUE] <liste_de_tuples>
  <nom_colonne> BETWEEN constante AND constante
  AND | OR <search condition>
```

avec

$$\theta ::= < \mid = \mid > \mid \geq \mid \leq \mid \diamond$$

Remarques:

<liste_de_valeurs> et <liste_de_tuples> peuvent être déterminés par une requête

<tuple> ::= (<nom_colonne> [, <nom_colonne>]...)

Projections et restrictions simples

Liste des médicaments de plus de 500 FF → NomMed

SELECT NomMed FROM MED WHERE Prix > 500 ;

Liste des médicaments de plus de 100 € → NomMed (prix stocké en FF)

SELECT NomMed FROM MED WHERE Prix/6,55957 > 100 ;

Nom des docteurs de LAON → NomDoc

SELECT NomDoc FROM DOC WHERE VilleDoc = "Laon"

Quelles ont été les motifs de consultations de 25/12/2006 → Motif

SELECT DISTINCT Motif FROM RDV WHERE DateRDV = #25/12/2006#

EXERCICE 6

- Liste des patients de plus de 40 ans → NumPat, NomPat
- Age des patients en mois (il est stocké en année) → NumPat, AgeMois
- Numéro des docteurs ayant consulté entre le 1/9 et le 31/12/06 → NumDoc
- Nom des patients niçois de plus de 36 ans → NumPat, NomPat

Restrictions complexes et jointures

Liste des patients ayant un RDV avec le docteur "Dupont" → NomPat

```
SELECT DISTINCT PAT.NomPat FROM PAT, RDV, DOC  
WHERE PAT.NumPat = RDV.NumPat and RDV.NumDoc = DOC.NumDoc  
and DOC.NomDoc like "dupont";
```

Médicaments commençant par « ASPI » prescrits le 25/12/2006 → NomMed

```
SELECT DISTINCT M.NomMed FROM MED M, DET D, ORD O  
WHERE M.NumMed = D.NumMed and D.NumOrd = O.NumOrd  
and O.Date = #25/12/2006# and NomMed like "ASPI%";
```

Docteurs ayant le même nom qu'un patient → NomDoc

```
SELECT D.NomDoc FROM PAT P, DOC D WHERE P.NomPat = D.NomDoc;
```

EXERCICE 7

- Docteurs ayant le même nom que leur patient → NomDoc
- Nom des médicaments prescrits le 25/12/2006 → NomMed
- Nom des docteurs ayant un RDV pour un motif commençant par 'grippe' → NomDoc
- Liste des quantité distinctes prescrites du médicament Bubol → Qté

Requêtes imbriquées : IN et EXISTS

Liste des patients ayant un RDV avec le docteur "Dupont" → NomPat

```
SELECT DISTINCT P.NomPat FROM PAT P, RDV R, DOC D  
WHERE P.NumPat = R.NumPat and R.NumDoc = D.NumDoc and D.NomDoc = "Dupont";
```

```
SELECT P.NomPat FROM PAT P WHERE P.NumPat in  
  (SELECT R.NumPat FROM RDV R WHERE R.NumDoc in  
    (SELECT D.NumDoc FROM DOC WHERE D.NomDoc = "Dupont"));
```

```
SELECT P.NomPat FROM PAT P WHERE EXISTS  
  (SELECT * FROM RDV R WHERE P.NumPat = R.NumPat and EXISTS  
    (SELECT * FROM DOC D WHERE R.NumDoc=D.NumDoc and D.NomDoc = "Dupont"));
```

```
SELECT P.NomPat FROM PAT P WHERE EXISTS  
  (SELECT * FROM RDV R WHERE EXISTS  
    (SELECT * FROM DOC D WHERE P.NumPat = R.NumPat and R.NumDoc = D.NumDoc  
      and D.NomDoc = "Dupont"));
```

EXERCICE 8 (avec IN ou EXISTS)

- Nom des docteurs ayant au moins un RDV pour une grippe → NomDoc
- Nom des patients ayant eu des RDV avec "Dupont" et "Durand" → NomPAT
- Nom des patients qui n'ont jamais eu de rendez-vous → NomPAT

Calculs d'agrégats

Les fonctions d'agrégation (Count, Sum, Avg, Min, Max) permettent de réaliser des calculs sur des ensembles de données

- **Calcul de statistiques globales**

- Nombre de patients : **SELECT count(*) FROM PAT**
- Prix moyen des médicaments : **SELECT avg(Prix) FROM MED**

- **Calcul de statistiques par groupe**

- Nombre de patients par ville :

SELECT VillePat, count(*) NbPatient FROM PAT GROUP BY VillePat

- Nombre de patients par ville ayant consulté pour un mal de tête

SELECT VillePat, count(DISTINCT(NumPat)) NbPatient FROM PAT, RDV

WHERE PAT.NumPat = RDV.NumPat and Motif = 'mal de tête' GROUP BY VillePat

- Ville où plus de 10 patients ont consulté pour un mal de tête

SELECT VillePat FROM PAT, RDV

WHERE PAT.NumPat = RDV.NumPat and Motif = 'mal de tête'

GROUP BY VillePat

HAVING count(DISTINCT(NumPat)) > 10

Agrégats : autres exemples et exercices

- **Nom du(des) médicament(s) le(s) moins cher(s)**
SELECT NomMed **FROM** MED
WHERE Prix = **SELECT** MIN(Prix) **FROM** MED
- **Total des prix des médicaments prescrits par patient**
SELECT P.NomPat, sum(M.prix * D.Qté) PrixTotal
FROM PAT P, ORD O, DET D, MED M
WHERE M.NumMed = D.NumMed **AND** D.NumOrd = O.NumOrd **AND** O.NumPat = P.NumPat
GROUP BY P.NomPat
- **Moyenne du : Total des prix des médicaments prescrits par patient ...**
SELECTAVG(TMP.PrixTotal)
FROM (
 SELECT P.NomPat, sum(M.prix * O.Qté) PrixTotal
 FROM PAT P, ORD O, DET D, MED M
 WHERE M.NumMed = D.NumMed **AND** D.NumOrd = O.NumOrd **AND** O.NumPat = P.NumPat
 GROUP BY P.NomPat
) TMP

EXERCICE 9

- Nombre total d'ordonnances
- Nombre d'ordonnances par docteur
- Age du plus jeune patient
- Nom du plus jeune patient

Union/Intersection/Différence

<requêteSQL_A>

UNION | INTERSECT | EXCEPT

[ALL]

[CORRESPONDING [BY <liste_de_colonnes>]]

<requêteSQL_B>

CORRESPONDING précise que l'intersection se fait sur le nom des colonnes et non sur leur position...

BY liste de colonnes précisée réduit « l'assiette » de l'opération au sous-ensemble spécifié de colonnes...

[ALL] permet de conserver les doublons dans le résultat

[CORRESPONDING BY] n'est en général pas implanté (mais peut être remplacé par une jointure)

Union/Inter°/Diff. : exemples et exercices

- **Ensemble des personnes de la base médicale**

```
SELECT NomDoc NomPers FROM DOC  
UNION  
SELECT NomPat NomPers FROM PAT
```

- **Patients qui sont aussi médecins**

```
SELECT NomPat PatDoc FROM PAT  
INTERSECT  
SELECT NomDoc PatDoc FROM DOC
```

- **Patients qui ne sont pas médecins**

```
SELECT NomPat Patient FROM PAT  
EXCEPT  
SELECT NomDoc Patient FROM DOC
```

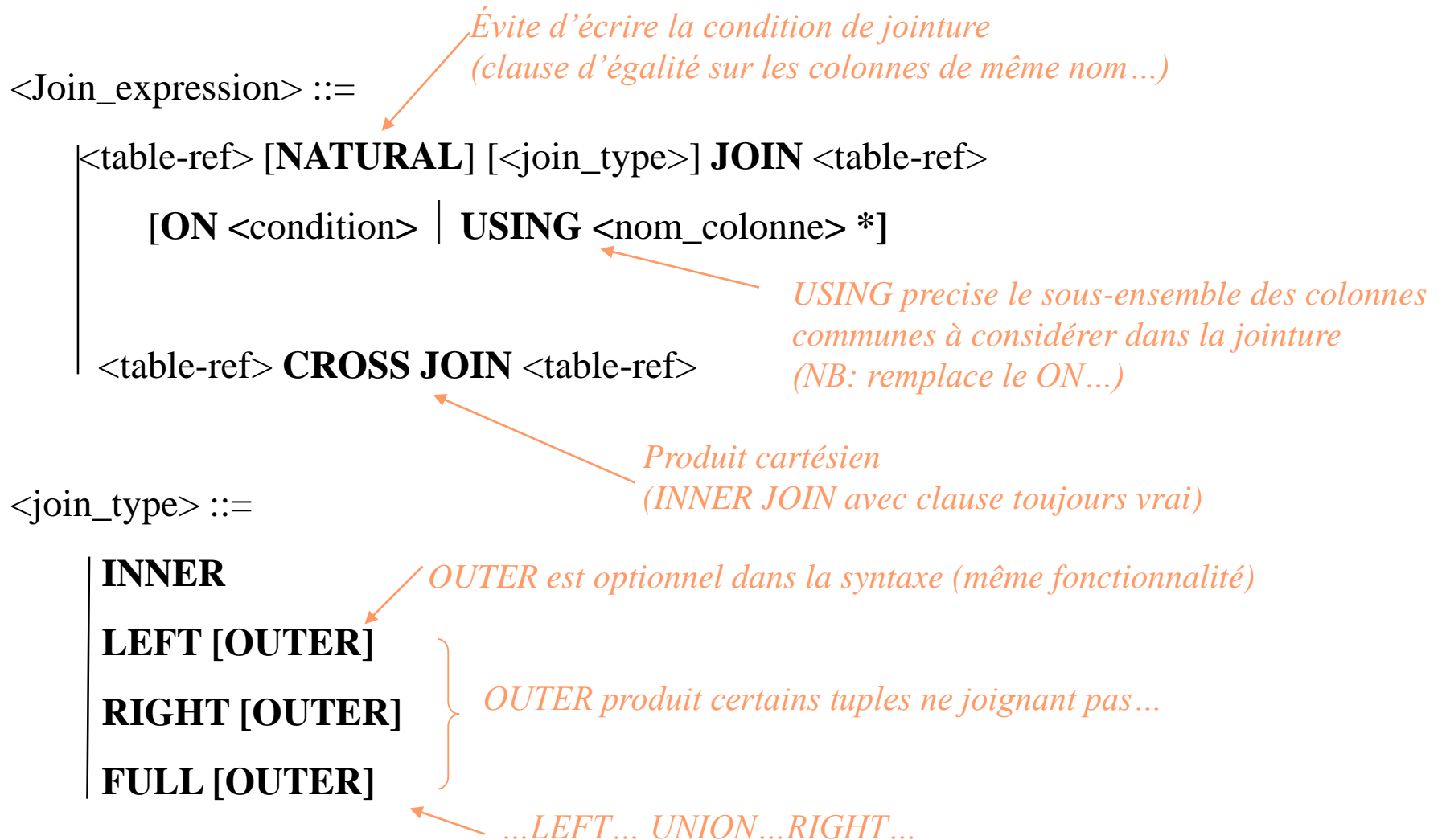
EXERCICES 10

- Nom des médicaments jamais prescrits
- Villes mentionnées dans la base
- Villes où il y a au moins un docteur mais aucun patient
- Villes où il y a au moins un docteur et au moins un patient

Des différences selon le SGBD...

	Access	SQL Server	Oracle
GROUP BY	Y	Y	Y
HAVING	Y	Y	Y
UNION	Y	Y	Y
INTERSECT	N	N	Y
EXCEPT	N	N	Y (MINUS)
CORRESPONDING BY	N	N	N

Jointure Interne / Externe

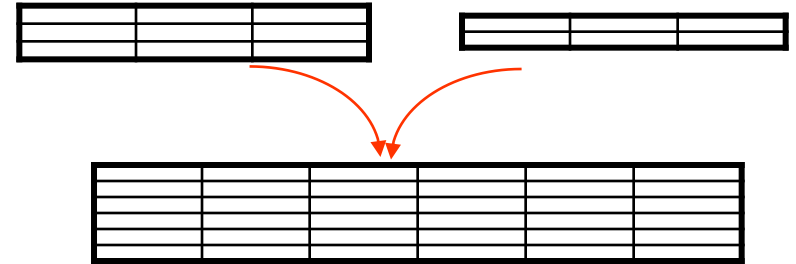


Le Langage SQL (3/3)

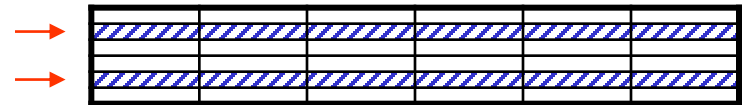
Méthodologie

Évaluation « sémantique » d'une requête SQL

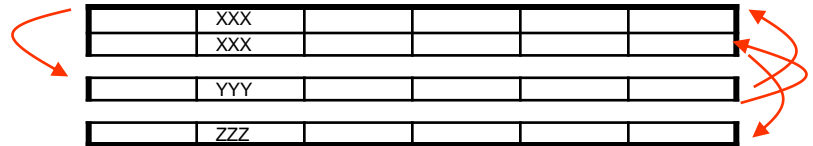
1. FROM
Réalise le produit cartésien des relations



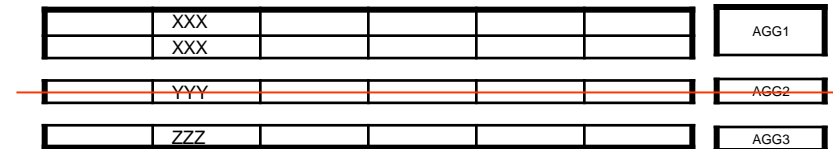
2. WHERE
Réalise restriction et jointures



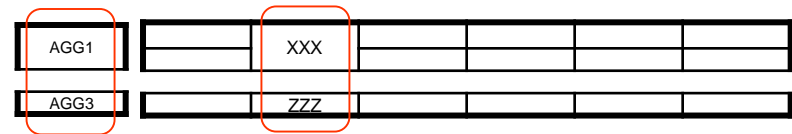
3. GROUP BY
Constitue les partitions
(e.g., tri sur l'intitulé du groupe)



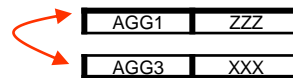
4. HAVING
Restreint aux partitions désirées



5. SELECT
Réaliser les projections/calculs finaux



6. ORDER BY
Trier les tuples résultat



Example (1)

- SELECT *
FROM PAT, RDV, DOC

DOC: NumDoc NomDoc VilleDoc

1	Jules	
2	Dupont	

RDV: NumRDV DateRDV NumDoc NumPat Motif

1		1	1	
2		2	2	
3		1	3	
4		2	1	
5		1	2	

PAT: NumPat NomPat VillePat AgePat

1			
2			
3			

DOC x RDV x PAT: NumDoc NomDoc VilleDoc NumRDV DateRDV NumDoc NumPat Motif NumPat NomPat VillePat AgePat

1	Jules		1					1			
2	Dupont		1					1			
1	Jules		2					1			
2	Dupont		2					1			
1	Jules		3					1			
1	Dupont		3					1			
2	Jules		4					1			
1	Dupont		4					1			
2	Jules		5					1			
1	Dupont		5					1			
1	Jules		1					1			
2	Dupont		1					1			
1	Jules		2					1			
2	Dupont		2					1			
1	Jules		3					1			
1	Dupont		3					2			
2	Jules		4					2			
1	Dupont		4					2			
2	Jules		5					2			
1	Dupont		5					2			
1	Jules		1					2			
2	Dupont		1					2			
1	Jules		2					2			
2	Dupont		2					2			
1	Jules		3					2			
1	Dupont		3					2			
2	Jules		4					2			
1	Dupont		4					2			
2	Jules		5					2			
1	Dupont		5					2			

Example (2)

- SELECT *
FROM PAT, RDV, DOC
WHERE DOC.NomDoc like "dupont";

$\sigma_{\text{DOC.NomDoc like 'Dupont'}}$ (**DOCxRDVxPAT**): NumDoc NomDoc VilleDoc NumRDV DateRDV NumDoc NumPat Motif NumPat NomPat VillePat AgePat

[illegible]

σ_{DOC.NomDoc like 'Dupont' (DOCxRDVxPAT)}: NumDoc NomDoc VilleDoc NumRDV DateRDV NumDoc NumPat Motif NumPat NomPat VillePat AgePat

[illegible]

Examples (3)

- SELECT *
FROM PAT, RDV, DOC
WHERE DOC.NomDoc like "dupont" AND
PAT.NumPat = RDV.NumPat AND
RDV.NumDoc = DOC.NumDoc;

$\sigma_{\text{DOC.NomDoc like 'Dupont'}}$ (DOCxRDVxPAT): NumDoc NomDoc VilleDoc NumRDV DateRDV NumDoc NumPat Motif NumPat NomPat VillePat AgePat

[illegible]

σ_{DOC.NomDoc} like 'Dupont' (DOCxRDVxPAT): NumDoc NomDoc VilleDoc NumRDV DateRDV NumDoc NumPat Motif NumPat NomPat VillePat AgePat

2	Dupont		4		2	1		1		
2	Dupont		2		2	2		2		

Utiliser l'algèbre relationnelle

- En s'entraînant un peu, il semble plus facile d'écrire une requête en algèbre puis de la traduire en SQL
 - Cette traduction s'effectue de manière systématique et peut générer des expressions un peu 'lourde'. Toutefois, elle permet de bien comprendre la requête
 - Certaines requêtes ne peuvent s'exprimer en algèbre (Ex: contenant des calculs d'agrégats, groupements)
Toutefois, on peut souvent en exprimer une partie en algèbre...
 - La traduction des opérateurs d'algèbre est assez simple (excepté pour la division)
- **Sélection** = $\sigma_{\langle \text{critère} \rangle}(R) \rightarrow \text{select } * \text{ from } R \text{ where } \langle \text{critère} \rangle;$
- **Projection** = $\pi_{\langle \text{liste} \rangle}(R) \rightarrow \text{select } \langle \text{liste} \rangle \text{ from } R;$
- **Jointures (naturelle/théta)** = $R \bowtie_{\langle \text{critère} \rangle} S = \text{select } * \text{ from } R, S \text{ where } \langle \text{critère} \rangle;$
- **Union/Intersection/Différence** \rightarrow opérateur UNION/INTERSECT/EXCEPT...
- **Division** \rightarrow double négation, expression relationnelle de la division...

Traduction par double négation

- On veut diviser $R(A_1, \dots, A_p, \dots, A_n)$ par $Q(A_1, \dots, A_p)$
- Et obtenir $S(A_{p+1}, \dots, A_n)$

```

Select Ap+1, ..., An from R R1
where not exists (
  select * from Q
  where not exists (
    select * from R R2
    where R2.A1 = Q.A1
    and R2.A2 = Q.A2
    and ....
    and R2.Ap = Q.Ap
    and R2.Ap+1 = R1.Ap+1
    and ....
    and R2.An = R1.An ))

```

- La logique sur un exemple...
- Nom des docteurs qui ont prescrit tous les médicaments
- $\pi_{\text{NomDoc}, \text{NumMed}}(\text{DOC} \bowtie \text{VIS} \bowtie \text{ORD} \bowtie \text{MED}) \div \pi_{\text{NumMed}} \text{MED}$
- Equivalent à: Nom des docteurs tels qu'il n'existe pas de médicaments tel qu'il n'existe pas de prescription de ces médicaments faites par ces docteurs

```

SELECT NomDoc,
FROM DOC WHERE NOT EXIST(
  SELECT * FROM MED WHERE NOT EXIST(
    SELECT * FROM VIS, ORD
    WHERE DOC.NumDoc=VIS.NumDoc AND
    VIS.NumVis=ORD.NumVis AND
    ORD.NumMed=MED.NumMed
  )
);

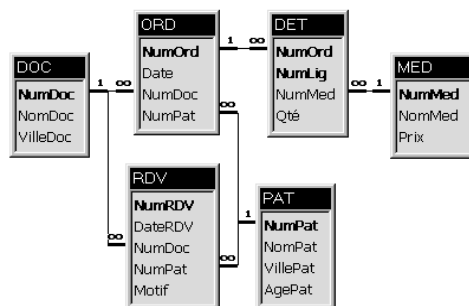
```

Erreurs dans l'écriture de requêtes

- Mauvaise compréhension du schéma de la base
- Mauvaise compréhension de la question
- Oubli de clauses dans les prédicats
- Formulation de requêtes correspondant à des négations
- Instances de la même table
- Doublons
- Base de tests

Schéma et question

- Avant de se lancer dans l'écriture d'une requête, il faut bien comprendre le schéma des tables sur lequel on va s'appuyer
- Si le schéma est complexe, il faut le dessiner, c.a.d. dessiner les tables et les relations entre ces tables
- En lisant la question, on repère sur le schéma dans quelle(s) relation(s) se trouve chaque donnée. On a alors une idée des tables mises en jeu
- Si la question est complexe, il faut la reformuler et/ou la décomposer.
- Une fois ce travail effectué, on peut commencer à écrire du SQL.

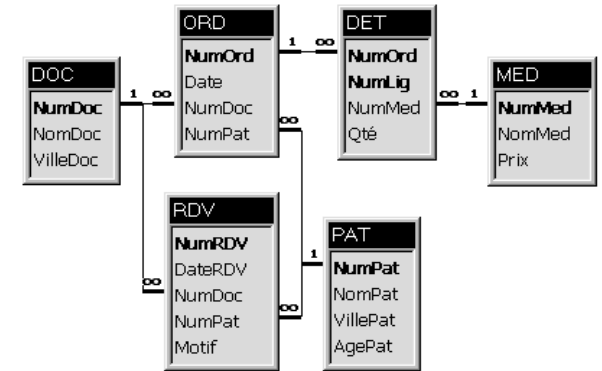


Oubli de clauses

- Il est assez facile d'oublier une clause dans les prédicats de sélection, surtout lorsque l'on joint plusieurs tables.
- Pour le vérifier, une règle simple pour les requêtes plates (nécessaire mais non suffisante) :
 - Une requête impliquant n tables doit posséder au moins $(n-1)$ prédicats de jointure (outre les prédicats de sélection).

Reformulation d'une question (1)

- Paraphrasage :
 - En exprimant de manière plus détaillée une requête, elle devient plus claire... (ou plus facilement exprimable)



- Exemple : Qui est dans la base ?
 - ➔ Ensemble des personnes de la base médicale
 - ➔ Nom des médecins et nom des patients
 - ➔ Union des noms des médecins et des noms des patients

```
SELECT NomDoc NomPers FROM DOC
UNION
SELECT NomPat NomPers FROM PAT
```

Reformulation (2) : Négation

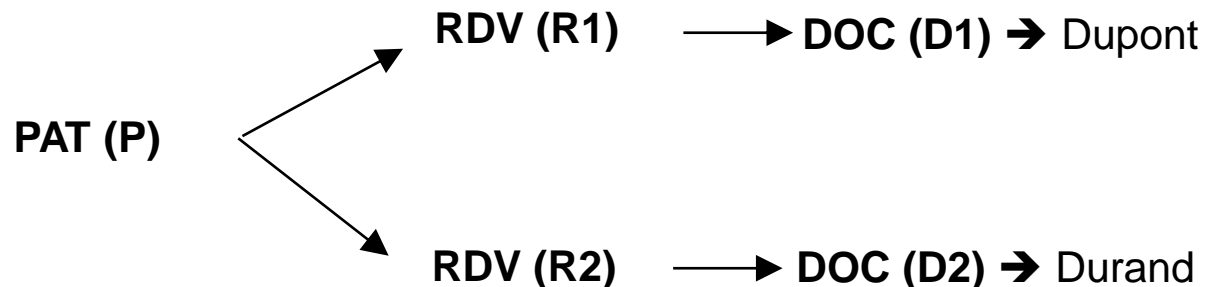
- Souvent l'inverse de la requête est plus facile à exprimer. Cela est particulièrement vrai lorsque la requête contient :
- Que
 - Dans quelle ville n'y a-t-il **QUE** des patients de plus de 40 ans?
 - Toutes les villes (où il y a au moins un patient de plus de 40 ans), moins les villes où il y a un patient de 40 ans ou moins
- Aucun
 - Dans quelle ville n'y a-t-il **AUCUN** patient de plus de 40 ans?
 - L'ensemble des villes, moins celles où il y a au moins un patient de plus de 40 ans
- Tous (conduisant à une division relationnelle...)
 - Quels sont les patients qui ont RDV avec **TOUS** les médecins
 - Les patients pour lesquels il n'existe pas de docteur avec qui ils n'ont pas de RDV
- Tous
 - Quels sont les patients dont **TOUS** les motifs de rendez vous sont « mal de tête »
 - L'ensemble des patients qui ont un RDV pour un « mal de tête », moins les patients qui ont un RDV pour un motif différent de « mal de tête »

Reformulation (3) : Décomposition

- En décomposant, on simplifie la compréhension et on diminue les risques d'erreurs :
 - Quels sont les patients dont tous les motifs de rendez vous sont « mal de tête »
- Paraphrase :
 - L'ensemble des patients qui ont un RDV pour un mal de tête moins ceux qui ont un RDV pour un motif différent
- Décomposition :
 - Create view V1 AS (patients qui ont un RDV pour un mal de tête)
 - Create view V2 AS (patients qui ont un RDV pour un motif \neq mal de tête)
 - SELECT NomPat FROM V1 EXCEPT SELECT NomPat FROM V2

Instance de tables

- Il faut parfois utiliser plusieurs instances de la même table
Comment savoir?
- Lorsqu'une même table est utilisée pour obtenir deux informations de '**sémantique différente**' (2 instances différentes d'entité)
➔ il faut prendre plusieurs instances de cette table
- Exemple :
Nom des patients ayant eu des RDV avec les docteurs "Dupont" et "Durand"



```
SELECT DISTINCT P.NomPat
FROM PAT P, RDV R1, DOC D1, RDV R2, DOC D2
WHERE P.NumPat = R1.NumPat AND R1.NumDoc = D1.NumDoc AND D1.NomDoc =
"Dupont"
      AND P.NumPat = R2.NumPat AND R2.NumDoc = D2.NumDoc AND D2.NomDoc =
"Durand";
```

Doublons

- Deux types de doublons peuvent apparaître
 - plusieurs réponses sont identiques
 - plusieurs réponses sont 'sémantiquement équivalentes'
- Les premiers s'éliminent en utilisant la clause distinct. Attention toutefois à son utilisation...
- Le deuxième cas se produit lorsque l'on demande des combinaisons (ex: couples de personnes, ensemble de 3 pièces etc...). Le résultat (A,B) est 'sémantiquement équivalent' à (B,A).
 - Dans ce cas, il suffit d'utiliser un prédicat > entre chaque composantes afin de n'obtenir qu'une seule des combinaisons équivalentes :
SELECT P1.Nom, P2.Nom
FROM Personne P1, Personne P2
WHERE P1.Nom > P2.Nom

Base de tests (1)

- Quand une requête est vraiment complexe, on peut, malgré tout, avoir un doute sur la solution trouvée. Dans ce cas, il faut créer l'extension d'une base de test afin de vérifier la validité de la requête.
- Preuve par l'exemple :
 - Un résultat positif ne prouve rien (c'est probablement juste...)
 - Un résultat négatif prouve que la requête est fausse
- Comment trouver l'extension la plus 'sensible' possible ?
 - C'est à dire, l'extension de taille minimum comportant tous les cas et garantissant la validité de la requête
 - Le problème est presque aussi complexe que la résolution de la requête elle-même
(mais souvent le fait d'aborder la requête de cette manière permet de mieux la comprendre...)

Base de tests (2)

- Exemple 1 : Les couples de parents ayant au moins deux enfants?
 - Dans la base de test il faut avoir :
 - des couples de parents qui ont 1, 2 et 3 enfants
 - et des parents seuls (ayant 2 et 3 enfants)
- Exemple 2 : Producteurs qui ne voient que les films qu'ils produisent
 - Les producteurs qui voient au moins 1 film qu'ils produisent, moins ceux qui voient au moins 1 film qu'ils ne produisent pas

Réponse :

```
SELECT DISTINCT P.Nom FROM Producteur P, Spectateur S WHERE P.Titre = S.Titre AND P.Nom = S.Nom  
MINUS
```

```
SELECT DISTINCT P.Nom FROM Producteur P, Spectateur S WHERE P.Titre = S.Titre AND P.Nom != S.Nom
```

Est-ce correct ?

- Dans la base de test il faut avoir :
 - un producteur qui ne voit aucun film
 - un producteur qui ne voit que des films qu'il produit
 - un producteur qui voit des films qu'il produit et d'autres
 - Attention, cette base n'est pas forcément suffisante....

Base de tests (3)

SELECT DISTINCT P.Nom FROM Producteur P, Spectateur S WHERE P.Titre = S.Titre AND P.Nom = S.Nom
MINUS

SELECT DISTINCT P.Nom FROM Producteur P, Spectateur S WHERE P.Titre = S.Titre AND P.Nom != S.Nom

Producteur.Nom	Producteur.Titre
Nom1	Titre1
Nom2	Titre2
...	...

Spectateur.Nom	Spectateur.Titre
Nom1	Titre1
Nom2	Titre1
...	...

Producteur Nom1 qualifié, ne voit que le film qu'il produit.
Producteur Nom2 non qualifié...

SELECT DISTINCT P.Nom FROM Producteur P, Spectateur S WHERE P.Titre = S.Titre AND P.Nom = S.Nom

Producteur.Nom	Producteur.Titre	Spectateur.Nom	Spectateur.Titre
Nom1	Titre1	Nom1	Titre1
Nom1	Titre1	Nom2	Titre1
Nom2	Titre2	Nom1	Titre1
Nom2	Titre2	Nom2	Titre1

SELECT DISTINCT P.Nom FROM Producteur P, Spectateur S WHERE P.Titre = S.Titre AND P.Nom != S.Nom

Producteur.Nom	Producteur.Titre	Spectateur.Nom	Spectateur.Titre
Nom1	Titre1	Nom1	Titre1
Nom1	Titre1	Nom2	Titre1
Nom2	Titre2	Nom1	Titre1
Nom2	Titre2	Nom2	Titre1

Un spectateur voit le film du producteur...

→ Resultat = ∅

Le 2ème membre devrait être: SELECT Nom,Titre FROM Spectateur MINUS SELECT Nom,Titre FROM Producteur

Le dictionnaire de données Oracle



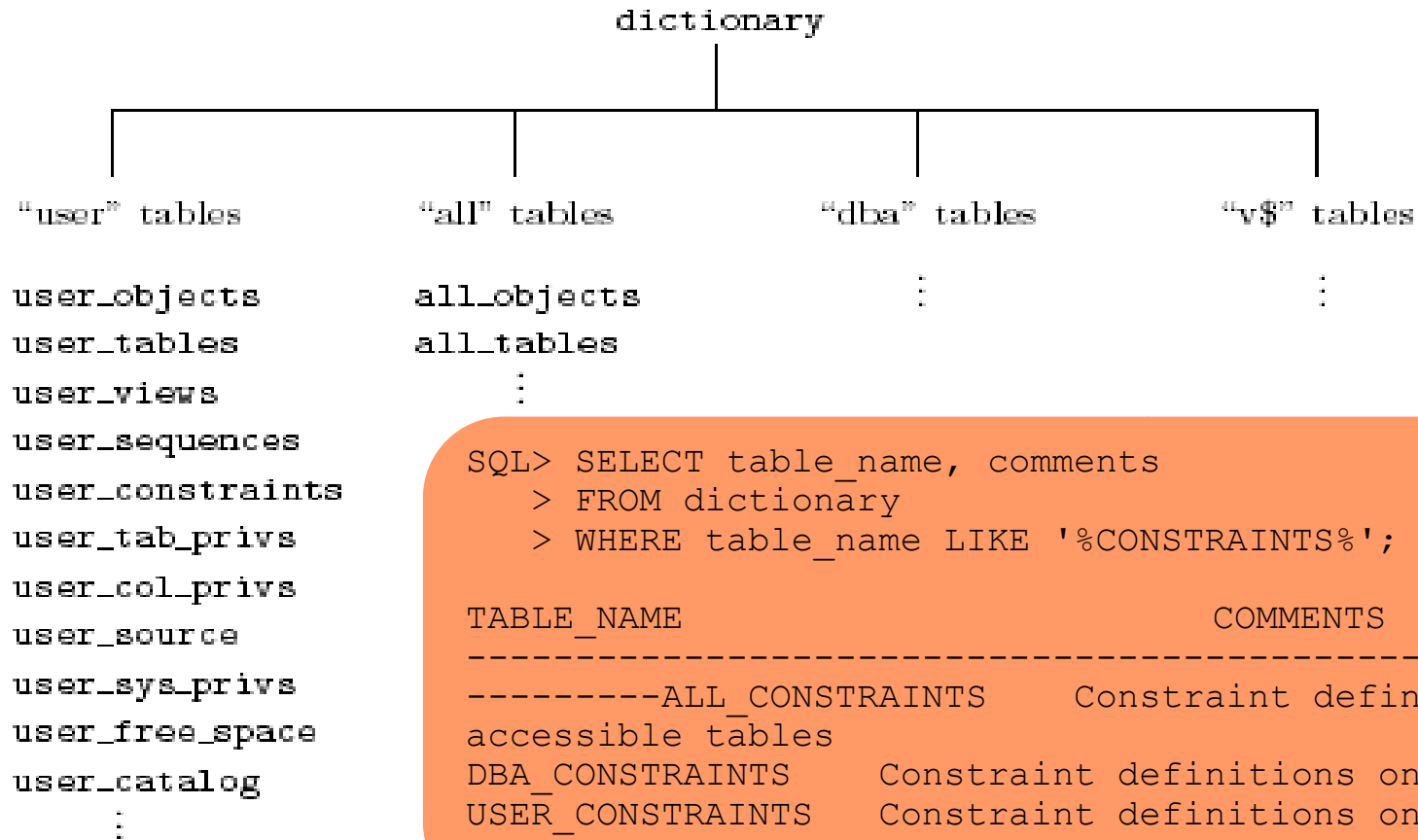
Contenu du Dictionnaire Oracle

- Dictionnaire Oracle = **tables systèmes** en lecture seule
- Evolution du dictionnaire
 - Généré au moment de la création de la DB
 - **Mis à jour automatiquement**
- Contient des **informations sur la structure** de la BD
 - Utilisateurs (privilèges, rôles)
 - Noms et caractéristiques des objets (tables, vues, index, clusters, triggers, packages, ...)
 - Contraintes d'intégrité
 - Ressources physiques allouées à la base
 - Valeurs par défaut pour des colonnes
 - Les autres informations générales sur la base des données
- Structure
 - **Tables** de base = tables fondamentales contenant les informations sur la BD
 - Conservées dans le tablespace SYSTEM
 - **Vues** de ces tables = présentation dépendant du rôle de celui qui consulte...

Utilisation du dictionnaire

- Accès avec **SQL**
- Utilisé par le **DBA** et les **utilisateurs**
 - Consultation d'informations sur la structure de la BD
 - Seulement des **droits en lecture** sur des **vues** du dictionnaire
 - NB: seulement par des requêtes SELECT (lecture seule)
- Utilisé par **Oracle**
 - A la connexion et pendant l'exécution des requêtes
 - Consultation d'informations sur les utilisateurs et leurs privilèges
 - Consultation d'informations sur les objets de la BD
 - Lors des requêtes SQL DDL (Data Definition Language)
 - **Modification** du dictionnaire

Différentes vues



```

SQL> SELECT table_name, comments
      > FROM dictionary
      > WHERE table_name LIKE '%CONSTRAINTS%';
  
```

TABLE_NAME	COMMENTS
-----ALL_CONSTRAINTS	Constraint definitions on accessible tables
DBA_CONSTRAINTS	Constraint definitions on all tables
USER_CONSTRAINTS	Constraint definitions on user's own tables

3 rows selected.

- La vue 'DICTIONARY' permet d'accéder aux noms/desc. des vues DBA, ALL, USER, V\$...

Les vues USER

- Description des objets logiques créés par l'utilisateur connecté
 - Objets logiques = tables, index, vues, triggers, procédures...
- Exemples de vues USER_
 - USER_TABLES
 - Tables créées par l'utilisateur
 - USER_INDEXES
 - Index créés par l'utilisateur
 - USER_CONSTRAINTS
 - Contraintes créées par l'utilisateur
 - USER_VIEWS
 - Informations sur les vues créées par l'utilisateur
 - USER_USERS
 - Information sur l'utilisateur

```
SQL> SELECT table_name
> FROM dictionary
> WHERE table_name LIKE
'%USER_%';
```

```
TABLE_NAME
```

```
-----
```

```
...
```

```
USER_VIEWS
```

```
USER_HISTOGRAMS
```

```
115 row(s) selected.
```

Les vues ALL

- Ces vues décrivent
 - les objets créés par l'utilisateur connecté (comme dans user_tables)
 - Et aussi tous les objets accessibles à cet utilisateur

Dans user_tables

```
SQL> desc user_tables;
```

Nom

TABLE_NAME ...

...

Dans all_tables

```
SQL> desc all_tables;
```

Nom

OWNER ...

TABLE_NAME ...

...

- Exemples de vues ALL_

```
SQL> SELECT table_name FROM all_tables;
```

...

```
SQL> SELECT owner FROM all_users;
```

...

```
SQL> SELECT index_name FROM all_indexes;
```

...

```
SQL> SELECT table_name, constraint_name FROM all_constraints;
```

...

Les vues ALL : exemple

- La vue ALL_CONSTRAINTS
 - Contraintes des tables accessibles à l'utilisateur

Column	Datatype	NULL	Description
OWNER	Varchar2(30)	Not Null	Le propriétaire de la définition de la contrainte
CONSTRAINT_NAME	VARCHAR2(30)	Not Null	Nom de la définition de la contrainte
CONSTRAINT_TYPE	VARCHAR2(1)		Type de la définition de la contrainte <ul style="list-style-type: none"> C (vérifier la contrainte sur une table) P (clé primaire) U (clé unique) R (intégrité référentielle) V (avec l'option de vérification sur la vue) O (avec lecture seulement sur la vue)
TABLE_NAME	VARCHAR2(30)	Not Null	Nom associé avec la table (ou la vue) avec définition de contrainte
SEARCH_CONDITION	LONG		Text de recherche de la condition pour vérifier la contrainte
R_OWNER	VARCHAR2(30)		Le propriétaire de la table référée par une contrainte référentielle
R_CONSTRAINT_NAME	VARCHAR2(30)		Nom de l'unique définition de la contrainte pour une table référencée
DELETE_RULE	VARCHAR2(9)		Règle de suppression pour la contrainte référentielle (CASCADE ou NO ACTION)
....			

Les vues DBA (1)

- Décrivent **tous** les objets de la base
 - Sur certains objets, la description est plus complète
- Accessibles qu'aux utilisateurs
 - Ayant le rôle SELECT_CATALOG_ROLE
 - Ou ayant le privilège système SELECT ANY DICTIONARY
- Ex. La vue DBA_TABLES

```
SQL> SELECT table_name, num_rows, blocks,  
      <      empty_blocks, avg_space  
      > FROM dba_tables  
      > WHERE table_name='Livre';
```

```
TABLE_NAME  NUM_ROWS  BLOCKS  EMPTY_BLOCKS  
AVG_SPACE
```

```
-----  
-----  
LIVRE       6764      180      19  
861
```

```
1 row(s) selected.
```


Les vues dynamiques V\$

- Vues dont les informations sont «dynamiques»
 - Evoluent du démarrage de l'instance jusqu'à son arrêt
 - Décritent l'activité de la DB et de l'instance
 - Sont appelées «dynamiques» mais
 - en fait, elle externalise l'état de variables internes à Oracle
- Usage de ces données
 - Principalement pour l'amélioration des performances de la BD
- Ex. V\$Session

```
SQL> SELECT sid, serial#, username, type, status  
> FROM v$session;
```

SID	SERIAL#	USERNAME	TYPE	STATUS
1	1		BACKGROUND	ACTIVE
...				
8	6	HEURTEL	USER	INACTIVE

- Le DBA peut déconnecter les utilisateurs avec la commande suivante

```
SQL> ALTER SYSTEM KILL SESSION '8,6' ;
```

Table DUAL

- Elle permet de
 - récupérer la date système (SYSDATE)
 - tester le formatage de données de type DATE
 - tester le bon «parenthésage» d'expressions
 - etc.
- Possède une seule colonne (DUMMY) et un seul tuple

```
SQL> SELECT TO_CHAR(SYSDATE, 'DD-MM-YYYY HH24:MI')  
        > FROM DUAL;
```

```
TO_CHAR(SYSDATE, 'DD-MM-YYYY HH24:MI')  
-----  
23-02-2004 22:05
```

```
SQL> SELECT 89456/562 FROM dual;
```

```
89456/562  
-----  
159.174377
```

TD – Manipulation des données

Programmer avec un SGBD

Langages procéduraux dédiés
et API de communication
standards
(ODBC-JDBC)

Programmation SGBD (1)

- Comment passer une commande dans la base telle que :

PROCEDURE COMM

Si le client n'est pas encore dans la base
→ Insérer les informations du client dans la table client
Si l'article est disponible dans la quantité commandée
ET le livreur disponible à la date de livraison désirée
→ Insérer sa commande dans la base
Sinon abandonner la commande

- Impossible en SQL pur (SQL n'est pas un langage complet)
 - Pas de structure de contrôle : itérations, tests ...
 - Besoin d'un langage complet pour programmer des actions sur les BD
- Avec un langage traditionnel (C, C++, Java, Cobol, ...)
 - Qui puisse parler avec la BD (avec SQL grâce à SQL CLI, JDBC, ODBC, ...)
- Avec un langage dédié au SGBD (PL/SQL pour Oracle, TSQL pour SQLServer, ...)
 - Types identiques aux types SQL
 - Facilités de récupération/lecture des résultats de requêtes
 - Adapté aux types de données BD

Programmation SGBD (2)

- **Langage procédural propriétaires** fournies par un éditeur de SGBD
 - Ex: PL/SQL (Oracle), PL/pgSQL (PostgreSQL), T-SQL (SQLServer, Sybase), ...
 - Extension de SQL à des éléments de programmation procédurale
 - instructions conditionnelles, itérations, variables, etc.
- **API Propriétaires** fournies par un éditeur de SGBD
 - Ex: OCI (Oracle), DB-Lib (Sybase), ...
 - Programme écrit dans un langage classique (C, C++, Cobol, etc.)
... avec des appels à ces API non standards fournies par l'éditeur du SGBD
- **Embedded SQL** (générateur de code - API propriétaires)
 - Ex: Pro*C (Oracle), ECPG (Postgres), Pro*Cobol (Oracle), etc...
 - Programme écrit dans un langage classique (C, C++, Cobol, etc.)
... avec des instructions SQL directement dans le programme
 - Précompilation : le source (ex: C+SQL) est transformé en code compilable (ex: C pur)
 - Suivi d'une compilation classique (ex: C pur)
- **API indépendantes** du SGBD
 - **SQL-CLI** (Call Level Interface) popularisé par les médiateurs **ODBC**, **JDBC**

Programmation SGBD (3)

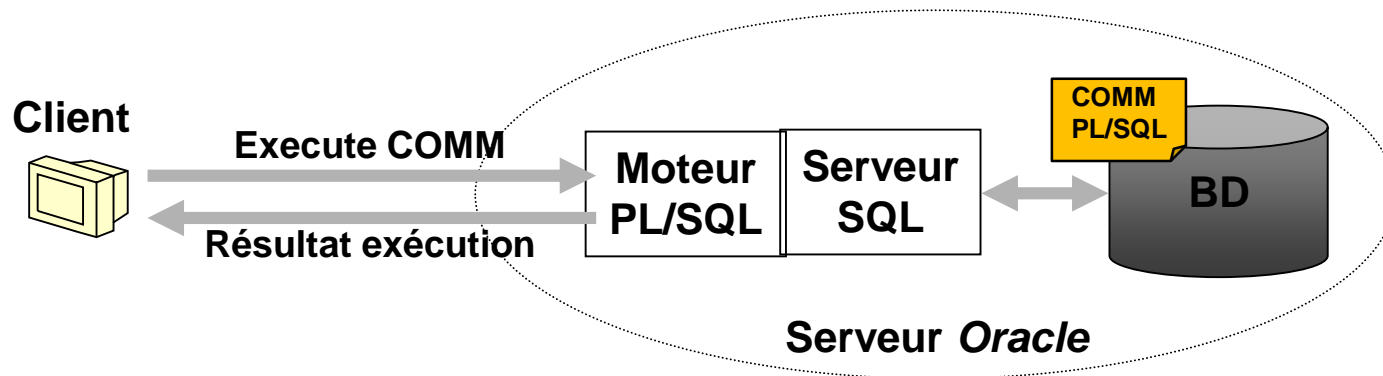
- API indépendantes du SGBD (standardisées)
 - **SQL-CLI** (Call Level Interface)
 - Permet de se connecter et d'envoyer des requêtes SQL à tout serveur SQL, quel que soit son type, sa localisation, le mode de connexion ...
 - API standardisée par X/Open depuis 1994
 - Intégrée dans le standard SQL (SQL 1992, SQL 2003)
 - **ODBC** (Open DataBase Connectivity)
 - Créée par Microsoft
 - Interface C de connection à un SGBD, administration via une interface Microsoft
 - Pour chaque SGBD, un pilote ODBC est nécessaire (Microsoft → Access, SQLServer)
 - Basée sur le standard SQL-CLI
 - Disponible aussi sur plateformes autres que Windows : <http://www.unixodbc.org/>
 - **JDBC** (Java DataBase Connectivity)
 - Créée par SUN (racheté par Oracle)
 - Interface JAVA de connection à un SGBD
 - Pour chaque SGBD, un pilote JDBC est nécessaire
 - Basée sur le standard SQL-CLI

Oracle PL/SQL

Procedural Language / Structured Query
Language
(Langage propriétaire Oracle)

Langages procéduraux dédiés

- Stockage de procédures sur le serveur
 - Chargement et compilation de la procédure sur le serveur
 - Lancement à distance par appel de procédure (*EXECUTE*)



- Stocké comme un objet base de données
 - Le créateur a les droits d'exécuter, modifier, re-compiler la procédure
 - Partage les droits avec d'autres (*GRANT/REVOKE EXECUTE*)

Bloc PL/SQL

- Bloc de déclaration de variables:

```
DECLARE
```

```
    <liste des variables utiles au programme>
```

- Suivi d'un bloc d'instructions et exceptions:

```
BEGIN
```

```
    <liste d'instructions du programme PL/SQL>
```

```
    [ EXCEPTION
```

```
        <instructions exceptionnelles> ]
```

```
END;
```

```
/
```

Types des variables PL/SQL (1)

- Types de base
 - Types Oracle (CHAR, NUMBER, DATE...)
 - Type Booléen : boolean
 - Types référençant le dictionnaire de données : table.col%TYPE
- Types complexes
 - Record
 - TYPE monType IS RECORD (champ1 NUMBER, champ2 VARCHAR2);
 - Table
 - TYPE maListe IS TABLE OF NUMBER ;
 - Curseurs (cf. plus loin)

Types et variables PL/SQL (2)

- Déclaration des variables dans le bloc *declare*

DECLARE

```
maVar  VARCHAR2 DEFAULT 'ROUGE';  
maVar  Personne.nom%TYPE;    -- type de l'attribut concerné  
maVar  Personne%ROWTYPE;     -- type RECORD  
maVar  MonCurseur%ROWTYPE;  -- type RECORD
```

- Affectation de valeur

- Dans toutes les sections avec l'opérateur **:=**

```
maVar := 2;
```

- Dans la section *begin end* avec l'opérateur *into*

```
Select max(table.col) into maVar from table;
```

- Appel de variables extérieures dans le bloc *begin end*
 - Variables SQL*FORMS préfixées par :
 - Variables SQL*PLUS préfixées par &

Structures de contrôle PL/SQL

- Traitement conditionnel

```
IF condition THEN traitement1  
[ELSIF condition THEN traitement2]  
[ELSE traitement3]  
END IF;
```

- Itérations

```
WHILE condition LOOP  
    traitement  
END LOOP;  
  
FOR i IN 1..n LOOP  
    traitement  
END LOOP;  
  
LOOP  
    traitement  
EXIT WHEN condition END LOOP;
```

Curseurs PL/SQL : déclaration

- Résultat d'une requête SQL géré comme un fichier séquentiel
- Déclaration

```
DECLARE  
    CURSOR monCurseur IS requête_SQL;
```

- Curseurs avec paramètres
 - Paramètre fixés à l'ouverture du curseur

```
DECLARE  
    CURSOR monCurseur (nomRecherche IN VARCHAR2) IS  
        SELECT nom, adresse FROM Personne  
        WHERE nom = nomRecherche;
```

Curseurs PL/SQL : manipulation (1)

- Attributs de curseur
 - %NOTFOUND, %FOUND, %ISOPEN, %ROWCOUNT
 - S'évaluent à *true*, *false*, *null*, ou au numéro de tuple
- commandes classiques d'ouverture, lecture, fermeture
 - *OPEN*, *FETCH*, *CLOSE*

```
DECLARE
    CURSOR dept_10 IS SELECT Nom, Salaire
                      FROM employes WHERE NumDept = 10;
    tuple dept_10%ROWTYPE;
BEGIN
    OPEN dept_10;
    LOOP
        FETCH dept_10 INTO tuple;
        .....
    EXIT WHEN (dept_10%NOTFOUND) END LOOP;
    CLOSE dept_10;
END;
```

Curseurs PL/SQL : manipulation (2)

- Manipulation simplifiée
 - Déclaration implicite du curseur
 - Déclaration implicite de l'enregistrement récepteur
 - Ouverture et fermeture du curseur implicites
 - Ordre de lecture pas à pas *fetch* implicite
 - Condition de sortie implicite

```
BEGIN
  FOR tuple IN
    (SELECT Nom, Salaire FROM employes WHERE NumDept = 10)
  LOOP
    .....
  END LOOP;
END;
```


Curseurs PL/SQL : exemple

- Augmente de 5% le salaire des employés du service compta...

Exceptions prédéfinies (1)

- Levées automatiquement par le moteur PL/SQL
 - CURSOR_ALREADY_OPEN, INVALID_CURSOR
 - NO_DATA_FOUND, LOGIN_DENIED, PROGRAM_ERROR
 - ZERO_DIVIDE, DUP_VAL_ON_INDEX...
- Traitées par la procédure PL/SQL section *begin end*

```
BEGIN
    ...
    EXCEPTION
        WHEN CURSOR_ALREADY_OPEN THEN
            BEGIN Affiche_Err (-20000, 'problème...') ; END;
END;
```

Exceptions prédéfinies (2)

- Exemple de traitement lors de la levée de l'exception oracle DUP_VAL_ON_INDEX
 - Détecte les doublons sur la clé primaire dans la table

```
BEGIN
    INSERT INTO employes VALUES (num, nom, salaire);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        INSERT INTO doublons VALUES (num, nom);
END;
```

Exceptions définies par l'utilisateur (1)

- Déclaration dans la section *declare*
- Levée dans la section *begin end* par l'instruction *raise*
- Traitement de l'exception dans la section *begin end*
 - Dans la sous section *exception*
 - Lors de la détection dans la clause *when*
- La clause *when others* traite les exceptions non traitées

```
DECLARE
    sal_nul EXCEPTION;
    ...
BEGIN
    ...
    IF ... THEN RAISE sal_nul;
    END IF;
    EXCEPTION
        WHEN sal_nul THEN ...
        WHEN OTHERS THEN ...
END;
```

Exceptions définies par l'utilisateur (2)

- Exemple
 - Lève une exception quand la température du four dépasse 1000°...

```
DECLARE  
Err_Temp EXCEPTION;  
MaxTemp NUMBER;  
BEGIN  
    SELECT Max(temp) INTO MaxTemp FROM Four;  
    IF MaxTemp > 1000 THEN RAISE Err_Temp END IF;  
    EXCEPTION  
        WHEN Err_Temp THEN  
            BEGIN Affiche_Err (-200, 'le four va exploser') ; END;  
END;
```

Procédures et fonctions PL/SQL (1)

- Stockées sous forme compilée dans la base de données, de la même façon qu'un objet de la base
 - soumise aux mécanismes de sécurité ou de confidentialité
- Partagées par plusieurs applications et utilisateurs
 - à condition d'avoir le privilège EXECUTE
- Procédure
 - unité de traitement qui peut contenir des instructions
 - SQL (sauf DDL), PL/SQL, variables, constantes, curseurs et gestionnaire d'erreurs
- Fonction
 - procédure qui retourne une valeur

Procédures et fonctions PL/SQL (2)

- Création de procédure

```
CREATE [OR REPLACE] PROCEDURE nom_procédure [(liste_arguments)]  
    <IS|AS> declaration_variables  
    bloc_PLSQL  
liste_arguments ::=      nom_argument_1 {IN | OUT | IN OUT} type,  
                        nom_argument_2 {IN | OUT | IN OUT} type,  
                        ....  
                        nom_argument_n {IN | OUT | IN OUT} type
```

- Création de fonction

```
CREATE [OR REPLACE] FUNCTION nom_fonction [(liste_arguments)]  
    RETURN type {IS | AS} declaration_variables  
    bloc_PLSQL
```

- Re-compilation de procédure et fonction en cas de modification du schéma de la BD

```
ALTER <PROCEDURE | FUNCTION> nom COMPILE;
```

- Suppression de procédure et fonction

```
DROP {PROCEDURE | FUNCTION} nom;
```

Procédures et fonctions PL/SQL (3)

- Exemple de procédure
 - Modifie le prix d'un article d'un certain taux

```
CREATE PROCEDURE modif_prix (id IN NUMBER, taux IN NUMBER)
IS
BEGIN
    UPDATE article a
    SET a.prix_unitaire = a.prix_unitaire*(1+taux)
    WHERE a.id_article = id;
END;
```

- Exemple de fonction
 - Calcule le chiffre d'affaire

```
CREATE FUNCTION chiffre_affaire (id IN NUMBER) RETURN NUMBER
IS
    ca NUMBER;
BEGIN
    SELECT SUM(montant) INTO ca FROM vendeurs WHERE id_vendeur = id;
    RETURN ca;
END;
```


Package PL/SQL

- Package
 - regroupement de programmes dans un objet de la BD

```
CREATE [OR REPLACE] PACKAGE nom_Package  
<IS|AS>  
    déclaration_variables  
    déclaration_exceptions  
    déclaration_procédures  
    déclaration_fonctions  
  
END nom_Package;
```

Visible par
l'application
(public)

```
CREATE [OR REPLACE] PACKAGE BODY nom_Package  
<IS|AS>  
    déclaration_variables_globales  
    corps_procédures  
    corps_fonctions  
  
END nom_Package
```

Interne au
package
(privé)

Package PL/SQL : exemple

```
CREATE PACKAGE traitements_vendeurs IS
    FUNCTION chiffre_affaire (id_Vendeur IN NUMBER) RETURN NUMBER;
    PROCEDURE modif_com (id IN NUMBER, tx IN NUMBER);
END traitements_vendeurs;
```

```
CREATE PACKAGE BODY traitements_vendeurs IS
    FUNCTION chiffre_affaire (id_Vendeur IN NUMBER) RETURN NUMBER
    IS
        ca NUMBER;
    BEGIN
        SELECT SUM(montant) INTO ca FROM vendeurs WHERE id_vendeur = id;
        RETURN ca;
    END;

    PROCEDURE modif_com (id IN NUMBER, taux IN NUMBER)
    IS
    BEGIN
        UPDATE vendeur v
        SET v.com = v.com*(1+taux)
        WHERE v.id_vendeur = id;
    END;
END traitements_vendeurs;
```

Conclusion langages dédiés

- Les langages de type PL/SQL constituent le mode d'utilisation le plus courant des bases de données relationnelles
 - Utilisé pour programmer des procédures stockées ou des triggers
 - Performant (réduit le transfert réseau)
 - Bien adapté aux clients légers (ex: smartphone ...) car déporte des traitements sur le serveur
- Utilisé par les outils de développement de plus haut niveau (SQLforms)

OCI (Oracle) et OCILIB

Introduction (projet/TP)

Introduction à OCILIB

- Une librairie **open source**
- Pour coder une application en **langage C...**
... qui interagit avec une base **Oracle**
- Pour toute plateforme (Windows, Unix, Linux, Mac....)
- La librairie encapsule les **API OCI** (Oracle)...
... dans des fonctions simples
- Documentation en ligne :
<http://orclib.sourceforge.net/doc/html/modules.html>

1) Intitiliser la librairie

Importer la librairie

```
#include "ocilib.h"
int main(void)
{
```

Initialiser la librairie

```
    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;
```

```
    /* ... application code here ... */
```

Libérer la librairie

```
    OCI_Cleanup();
    return EXIT_SUCCESS;
}
```

2) Ouvrir une connexion à la base

Déclarer une connexion

```
#include "ocilib.h"
int main(void)
{
    OCI_Connection *cn;
```

Ouvrir cette connexion

Afficher les propriétés
de la connexion

```
        OCI_Connection *cn;

        if (!OCI_Initialize(NULL, "XE", "login", "pwd", OCI_SESSION_DEFAULT))
            return EXIT_FAILURE;

        cn = OCI_ConnectionCreate("XE", "login", "pwd", OCI_SESSION_DEFAULT);

        printf(OCI_GetVersionServer(cn));
        printf("Server major version : %i\n", OCI_GetServerMajorVersion(cn));
        printf("Server minor version : %i\n", OCI_GetServerMinorVersion(cn));
        printf("Server revision version : %i\n", OCI_GetServerRevisionVersion(cn));
        printf("Connection version : %i\n", OCI_GetVersionConnection(cn));

        OCI_Cleanup();
        return EXIT_SUCCESS;
}
```

Nom du service Oracle (voir la doc Oracle)

Login/mdp Oracle

3) Exécuter une requête SQL

```
#include "ocilib.h"
int main(void)
{
    OCI_Connection *cn;
    OCI_Statement* st;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("XE", "login", "pwd", OCI_SESSION_DEFAULT);
    printf(OCI_GetVersionServer(cn));
    printf("Server major    version : %i\n", OCI_GetServerMajorVersion(cn));
    printf("Server minor    version : %i\n", OCI_GetServerMinorVersion(cn));
    printf("Server revision version : %i\n", OCI_GetServerRevisionVersion(cn));
    printf("Connection    version : %i\n", OCI_GetVersionConnection(cn));

    st = OCI_StatementCreate(cn);
    OCI_ExecuteStmt(st, "SELECT * FROM CLI");

    OCI_Cleanup();
    return EXIT_SUCCESS;
}
```

Déclarer une requête SQL

Créer la requête au travers
de la connexion
Exécuter la requête
au travers de la connexion

La requête à exécuter

4) Parcourir le résultat de la requête

```
#include "ocilib.h"
int main(void)
{
    OCI_Connection *cn;
    OCI_Statement* st;
    OCI_Resultset* rs;
    int nb_col; int i=0; OCI_Column *col;
    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;
    cn = OCI_ConnectionCreate("XE", "login", "pwd", OCI_SESSION_DEFAULT);
    printf(OCI_GetVersionServer(cn));
    printf("Server major    version : %i\n", OCI_GetServerMajorVersion(cn));
    printf("Server minor    version : %i\n", OCI_GetServerMinorVersion(cn));
    printf("Server revision version : %i\n", OCI_GetServerRevisionVersion(cn));
    printf("Connection      version : %i\n", OCI_GetVersionConnection(cn));
    st = OCI_StatementCreate(cn);
    OCI_ExecuteStmt(st, "SELECT * FROM CLI");
    rs = OCI_GetResultset(st);
    nb_col = OCI_GetColumnCount (rs);
    for(i=0; i<nb_col; i++) {
        col = OCI_GetColumn (rs, i+1);
        printf("%s | ", OCI_ColumnGetName (col));
    } printf("\n");
    while (OCI_FetchNext(rs)) {
        for(i=0; i<nb_col; i++)
            printf("%s | ", OCI_GetSt:
            printf("\n");
    }
    OCI_Cleanup();
    return EXIT_SUCCESS;
}
```

Déclaration du résultat

Déclaration d'une colonne

Stockage du résultat
de la requête

Récupération du
nb de colonnes du résultat

Afficher les noms des colonnes

Affichage des lignes du résultat

Récupérer la i+1ème colonne

Récupérer le nom de la ième colonne

Récupérer la valeur de la ième
colonne
sous forme de chaîne de
caractère

TD – Programmation SQL

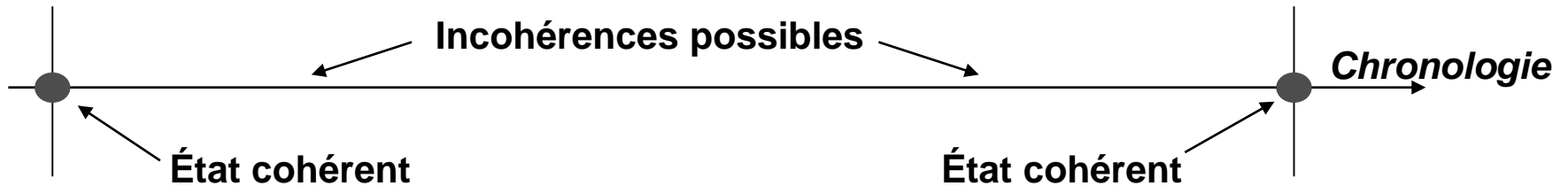
PL/SQL et
interrogation depuis C

Gestion de transactions

Problème de la concurrence d'accès

Terminologie des transactions BD

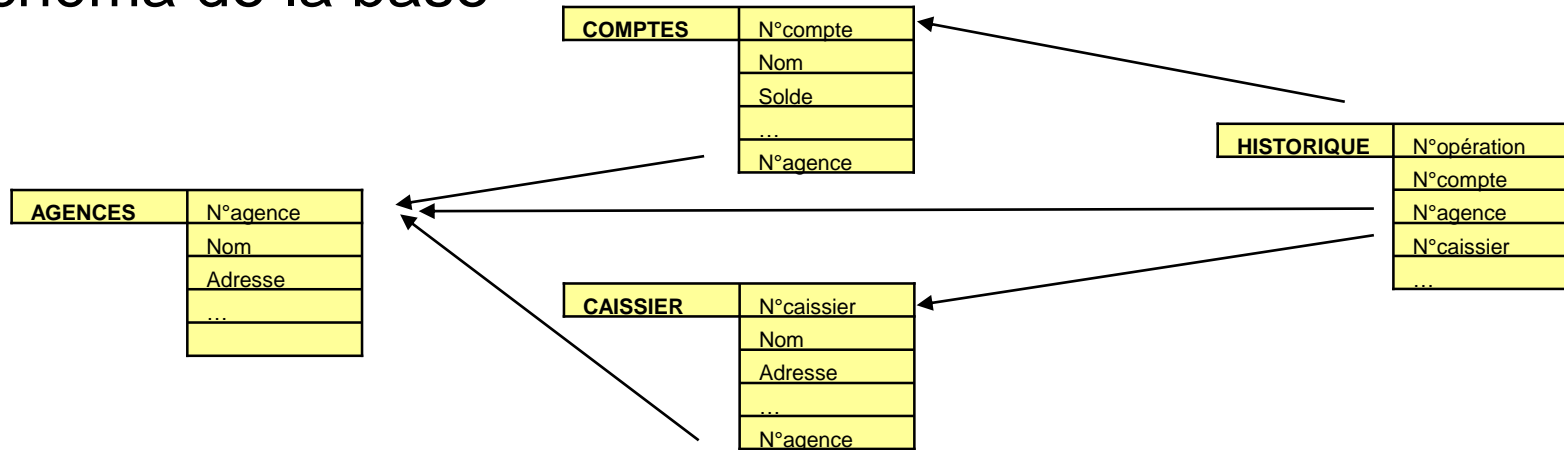
- Définition
 - unité de traitement faisant passer la base d'un état cohérent à autre



- Regroupe des opérations unitaires formant une action logique
- L'ensemble des opérations formant la transaction est délimité
 - Begin transaction, commit, abort
 - Fin avec succès : validation de la transaction
 - Toutes les opérations unitaires ont été validées
 - La base de donnée passe dans un nouvel état cohérent
 - Les effets de la transaction ne seront jamais perdus
 - Fin avec échec : abandon de la transaction
 - Annuler tous les effets de la transaction

Exemple de transaction (BD bancaires)

- Évaluation du système : opérations bancaires par seconde
- Schéma de la base



- L'opération bancaire représentative à effectuer...
dépôt de Δ sur le compte X par le caissier Y de l'agence Z à la date D

```
Update COMPTE set Solde = Solde +  $\Delta$  where N°compte = X ;  
Insert into HISTORIQUE (X, Z, Y, Bid,  $\Delta$ , D) ;  
Update CAISSIER set Solde = Solde +  $\Delta$  where N°caissier = Y ;  
Update AGENCE set Solde = Solde +  $\Delta$  where N°agence = Z ;
```

... est une transaction

Les propriétés des transactions

→ Atomicité

- Toutes les mises à jour doivent être effectuées ou aucune

→ Cohérence

- La base de donnée doit passer d'un état cohérent à un autre

→ Isolation

- Les résultats d'une transaction sont visibles une fois validés

→ Durabilité

- Les résultats de transactions validées ne sont jamais perdus

• Assurer l'**ACID**ité revient à résoudre les problèmes précédents

- Passer d'un état **cohérent** à un autre \Leftrightarrow assurer la **Cohérence**
- Résoudre le problème de **concurrency** \Leftrightarrow assurer l'**Isolation**
- Résister aux **pannes** \Leftrightarrow assurer l'**Atomicité** et la **Durabilité**

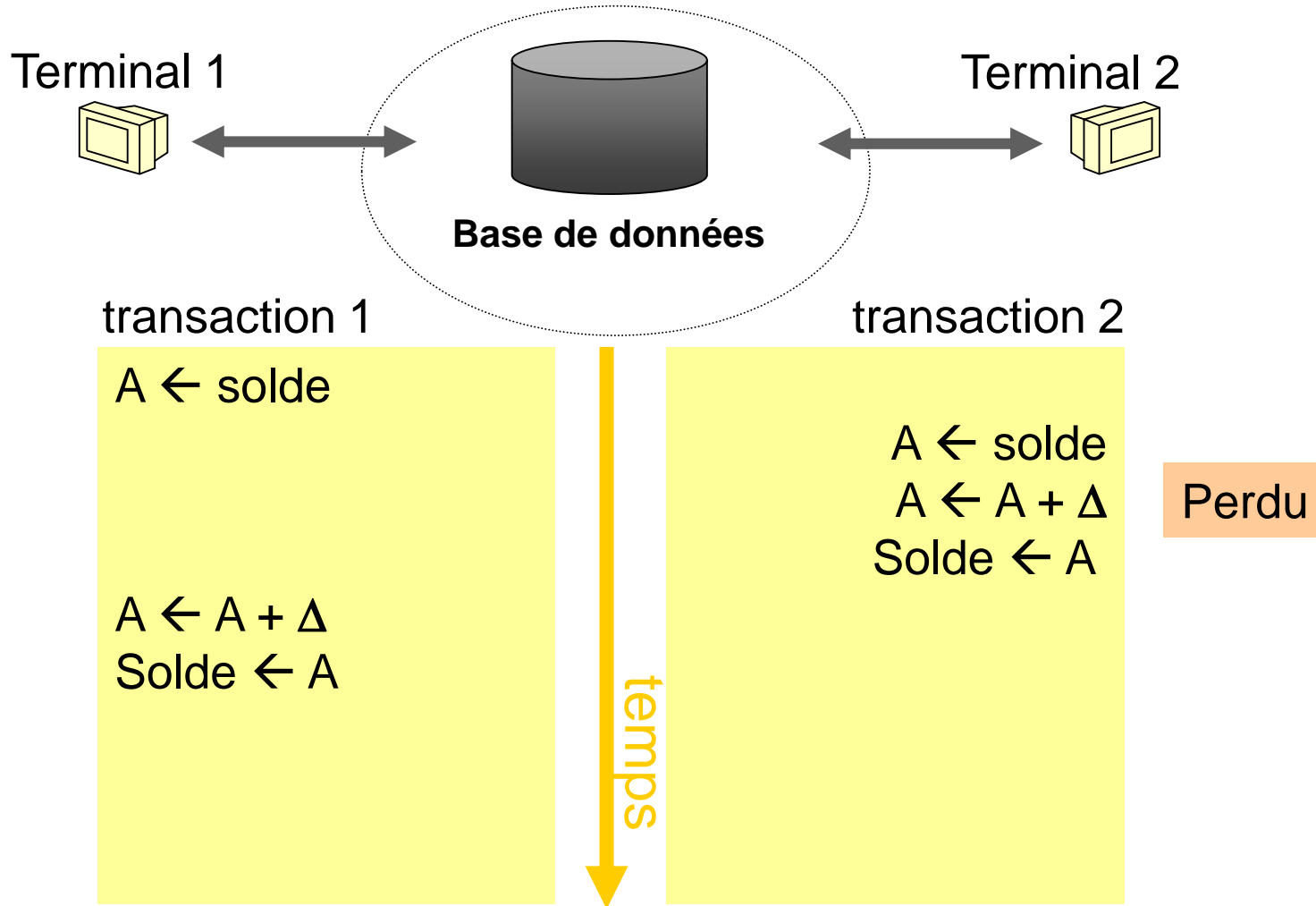
Problème de la concurrence d'accès

- Un problème transactionnel
- Les protocoles de concurrence sont nécessaires
 - Sinon conduit le SGBD à des erreurs
- La difficulté : des protocoles performants
- Les techniques
 - Degrés d'isolation
 - Multi-versions
 - Modification des programmes
- Leur utilisation sur des exemples

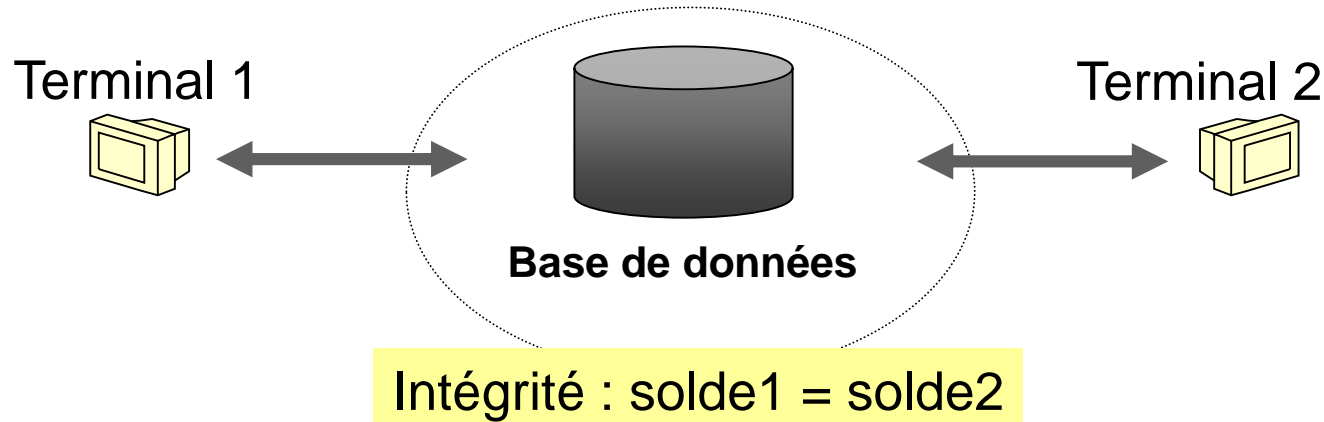
Gestion du contrôle de concurrence

- Problèmes de concurrence de transactions
 - Perte d'opération
 - Observation d'incohérences
 - Introduction d'incohérences
 - Lectures non reproductibles
- Objectif : assurer l'Isolation des transactions tout en
 - Permettant l'exécution simultanée d'un grand nombre de transactions
 - Gardant de très bonne performance
 - Évitant les blocages
- Pour cela on définit des protocoles de gestion de concurrence
 - Verrouillage
 - Estampillage

Problème 1 : perte d'opération



Problème 2 : observation d'incohérence



transaction 1

$A \leftarrow \text{solde1}$
 $A \leftarrow A - \Delta$
 $\text{Solde1} \leftarrow A$

$A \leftarrow \text{Solde2}$
 $A \leftarrow A - \Delta$
 $\text{Solde2} \leftarrow A$

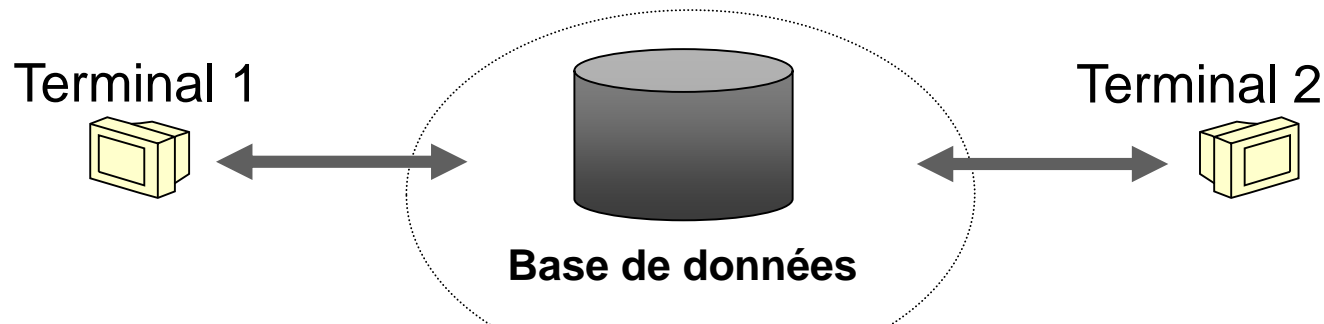
transaction 2

$A \leftarrow \text{solde1}$
 $B \leftarrow \text{solde2}$
 $\text{solde1} \leftarrow A + 1$
 $\text{solde2} \leftarrow B + 1$

$\text{solde1} \neq \text{solde2}$

temps

Problème 3 : introduction d'incohérence



Intégrité : $\text{solde1} = \text{solde2}$

transaction 1

$A \leftarrow \text{solde1}$
 $A \leftarrow A + \Delta$
 $\text{Solde1} \leftarrow A$

$A \leftarrow \text{Solde2}$
 $A \leftarrow A + \Delta$
 $\text{Solde2} \leftarrow A$

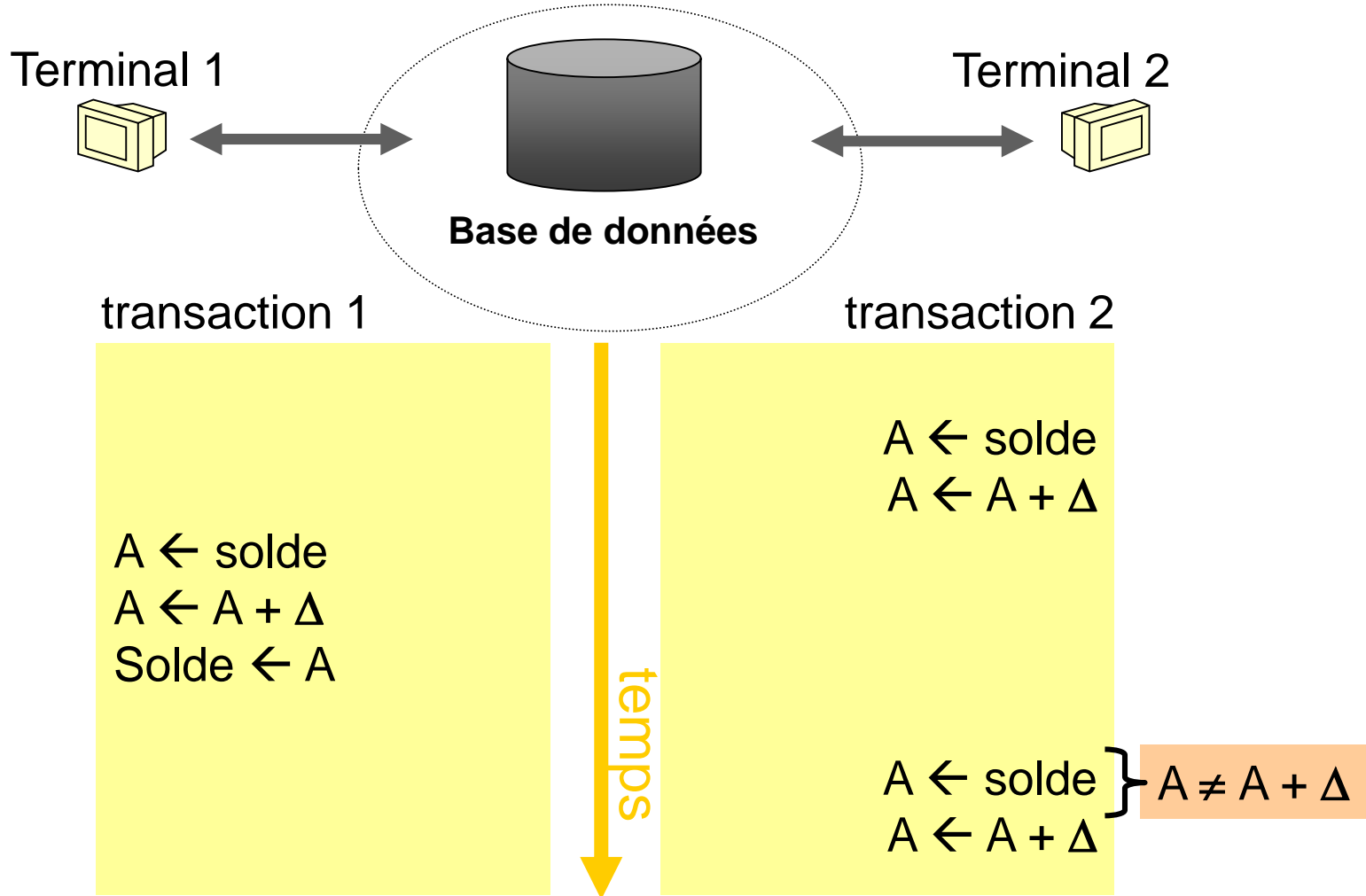
transaction 2

$A \leftarrow \text{solde1}$
 $B \leftarrow \text{solde2}$
 $\text{solde1} \leftarrow A \times 2$
 $\text{solde2} \leftarrow B \times 2$

temps

$\text{solde1} \neq \text{solde2}$

Problème 4 : lectures non reproductibles

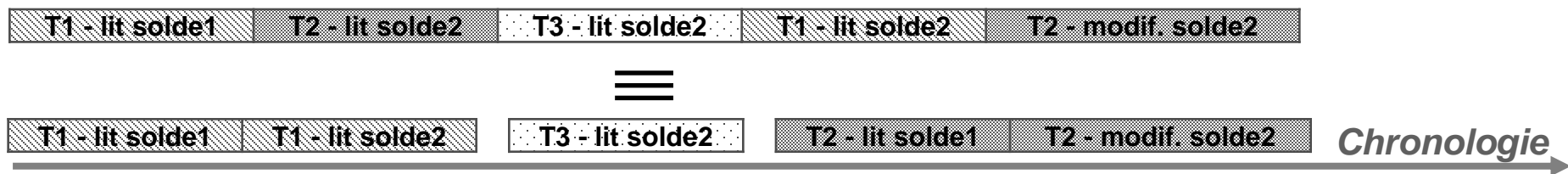


Problème : transactions en parallèle

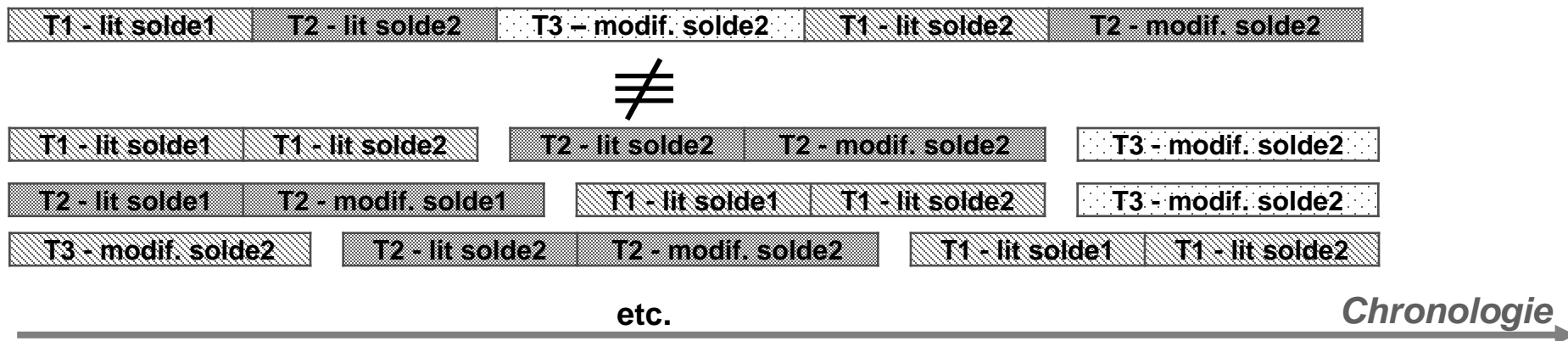
- Pas de problème si les transactions sont sérielles
→ Comment avoir une exécution \Leftrightarrow exécution en série ?
- Exécution sérialisable
 - une exécution en parallèle des transactions T_1, \dots, T_n est dite sérialisable si son résultat est équivalent à une exécution en série de ces mêmes transactions.
- Actions permutable (ou commutables)
 - les actions A et B sont permutable si toute exécution de A suivie par B donne le même résultat que l'exécution de B suivie par A.
- Si une exécution de transactions est transformable par permutations successives en une exécution en série, alors elle est sérialisable.

Exécution sérialisable ou non ?

- Exemple d'exécution sérialisable



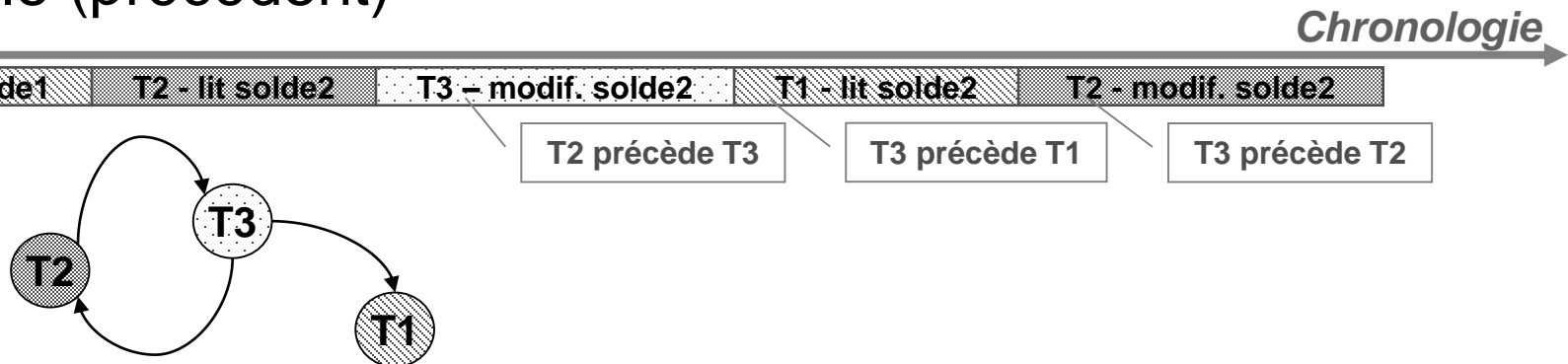
- Exemple d'exécution non sérialisable



→ Comment détecter une exécution non sérialisable ?

Détection de la non sérialisabilité

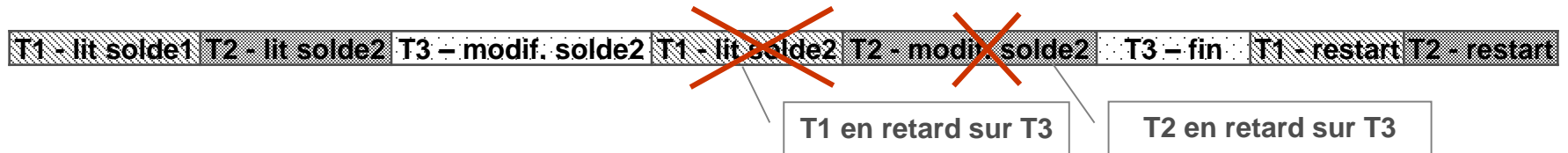
- Graphe de précédence de l'exécution → sérialisable ou non
 - Sur chaque objet, deux actions non permutables impliquent une relation de précédence entre les transactions correspondantes
 - Définition de la précédence
 - Si :
 - { – Une transaction T1 applique une action A1 avant que T2 applique A2
 - { – A1 et A2 sont non permutables
 - Alors : T1 précède T2 (i.e., arc de précédence de T1 vers T2)
- Graphe de précédence sans boucle → exécution sérialisable
- Exemple (précédent)



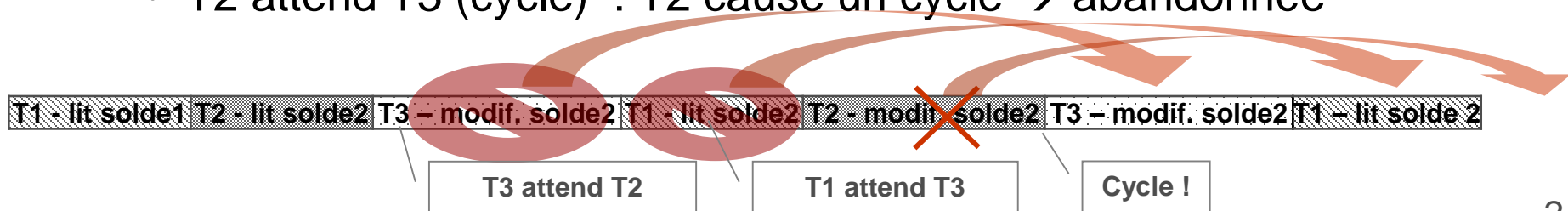
→ Comment faire pour rendre cette exécution sérialisable ?

Rendre l'exécution sérialisable

- Protocole d'estampillage : stratégie curative
 - Exécute les transactions en avance, annule celles en retard
 - T3 attend T2 (en avance) → T3 s'exécute
 - T1 attend T3 et T2 attend T3 (en retard) → T1 et T2 abandonnées

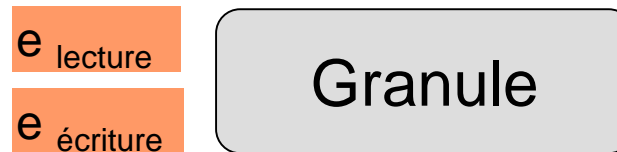


- Protocole de verrouillage : stratégie préventive
 - Exécute les transactions non conflictuelles, bloque les transactions conflictuelles, annule les cycles
 - T3 attend T2 et T1 attend T3 (conflits) → T3 et T1 bloquées
 - T2 attend T3 (cycle) : T2 cause un cycle → abandonnée



L'estampillage : principe

- Associer un numéro à chaque transaction (une estampille)
 - Attribuée au lancement de la transaction avec un compteur
 - Plus le numéro est petit, plus la transaction est 'vieille'
 - Chaque élément accédé (granule) conserve l'estampille du lecteur et de l'écrivain les plus récents



- Assurer que les accès non permutables aux granules se font dans l'ordre des estampilles
 - Éliminer les transactions qui veulent traiter un granule en cours de traitement par une transaction plus récente

L'estampillage : bilan ?

- Arcs du graphe de précédence dans l'ordre des estampilles
➔ Graphe de précédence sans circuit
- Ordonnancement équitable
 - Dans l'ordre d'arrivée des transactions
- Algorithme simple
- Problèmes
 - Abandon en cascade (solution : mises à jour différées)
 - une transaction ayant fait une modification est abandonnée, toutes celles qui on vu cette modification le sont aussi...
 - Abandon inutile (optimisations : règle de Thomas, estamp. multiversion)
 - On abandonne une transaction vieille qui veut écrire une donnée lue par une plus jeune. Pourtant, la jeune allait terminer, l'exécution était sérialisable...
 - Risque de privation
 - Une transaction longue indéfiniment abandonnée...

Le verrouillage

- Exécution simultanée des opérations commutables
 - Des opérations sur des objets différents sont commutables
 - Pour des opérations concurrentes sur un même objet :

Operation2 \ Operation1	Lecture	Écriture
Lecture	Commutable	Non commutable
Écriture	Non commutable	Non commutable

- Mise en œuvre par pose de verrous
 - Les transactions posent des verrous sur chaque objet accédé
 - En lecture : partagé ou S (Shared), autorise les lectures concurrentes
 - En écriture : exclusif ou X

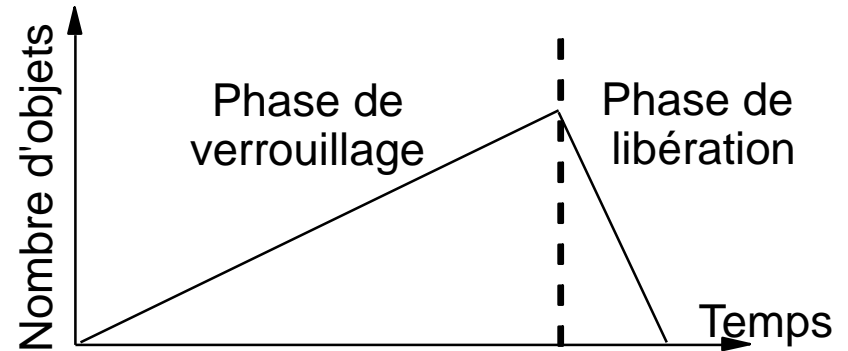
Verrou demandé \ Verrou détenu	S	X
S	Accordé	Mise en attente
X	Mise en attente	Mise en attente

- Les transactions effectuant une opération non commutable attendent...
- Les verrous sont relâchés à la fin de la transaction

Les 2 phases du verrouillage

- 2 phases → bon fonctionnement

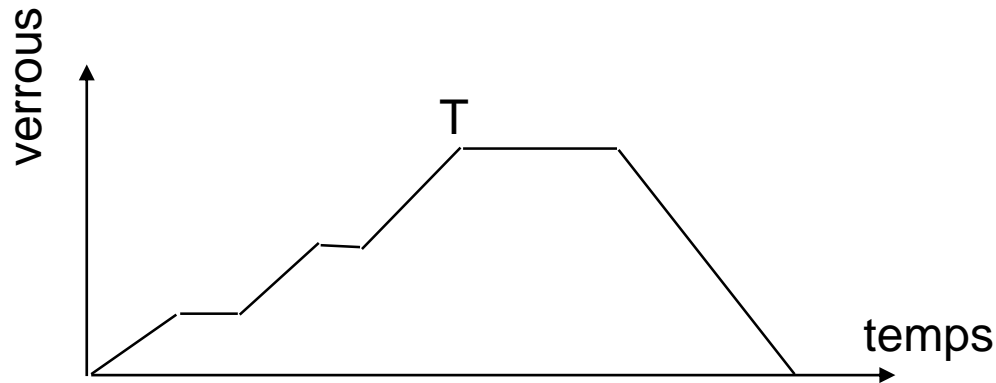
1. La transaction pose des verrous
2. La transaction libère les verrous (plus de pose possible)



- Permet d'assurer l'isolation
- Mais engendre des problèmes
 - Verrou mortel (cycle)
 - Transactions inter-bloquées (s'attendent mutuellement)
 - Performance
 - Petit granule (tuple)
 - verrouillage coûteux
 - Gros granule (page, table)
 - temps d'attente important
 - Cohabitation avec le décisionnel

- Solutions
 - Au problème du verrou mortel
 - Prévenir, détecter, éviter
 - Au problème de performance
 - Degrés d'isolation
 - Verrouillage hiérarchique
 - Granule variable
 - Protocoles multi-versions

Protocole de Verrouillage à 2 Phases

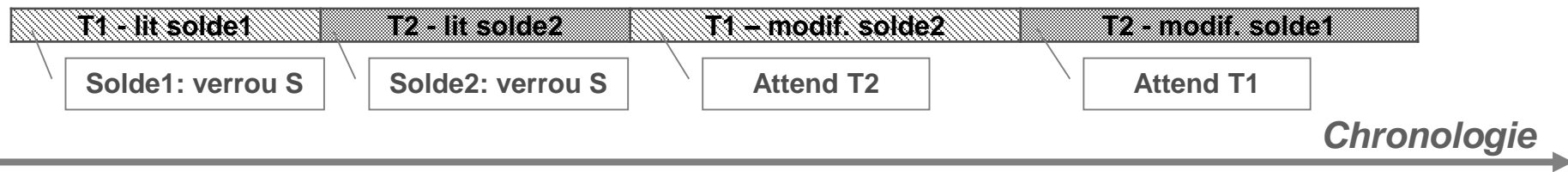


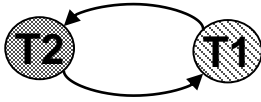
	Lecture	Ecriture
Lecture		
Ecriture		

- Règle 1 : verrouillage
 - Avant d'accéder à une donnée D, une transaction doit acquérir un verrou sur D. Si D est déjà verrouillée dans un mode non compatible, la transaction attend.
- Règle 2 : deux phases
 - Dès qu'une transaction relâche un verrou, elle ne peut plus acquérir de nouveau verrou => bon fonctionnement, assure l'isolation
- Verrouillage 2 phases stricts
 - Les verrous sont gardés jusqu'au commit
- Des problèmes : verrou mortel, performances

Le verrou mortel – détection

- Exemple



- Détection : graphe des attentes
 - Ti attend Tj si Ti demande un verrou détenu par Tj
 - Des qu'un cycle apparaît dans le graphe → verrou mortel
 - Sur l'exemple : 
- Comment résoudre le problème ?

Le verrou mortel – résolution

- Par prévention :

1. Attente ou mort (« wait-die »)

- Si T_i demande un verrou accordé à T_j et T_i est plus vieille
 - alors T_i peut attendre
 - sinon elle est annulée
- les jeunes ne peuvent attendre les vieux

2. Assassinat ou attente (« wound-wait »)

- Si T_i demande un verrouillage accordé à T_j et T_i est plus jeune
 - alors T_i peut attendre
 - sinon T_j est annulée
- les jeunes ne peuvent bloquer les vieux

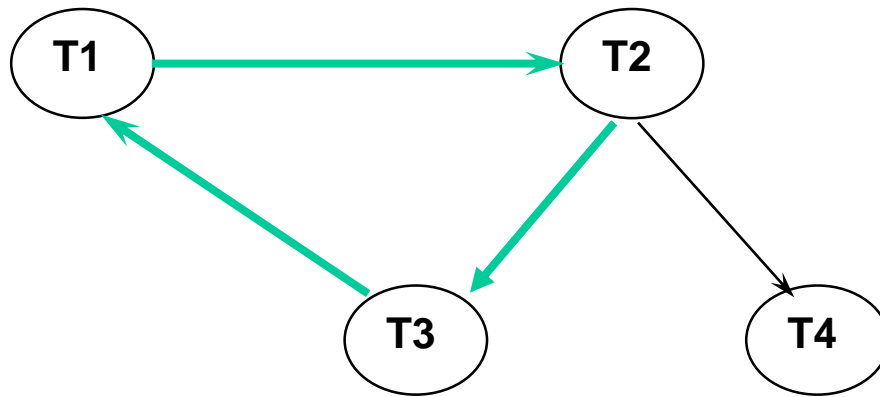
En quoi différent de l'estampillage?

- Par abandon :

- Abandon de transactions une fois le verrou mortel détecté

Optimisation des transactions

- L'objectif, en termes de performances, est de réduire :
 - les blocages : une transaction est bloquée en attente d'un verrou
 - les inter-blocages : un ensemble de transactions s'attendent mutuellement



- Solutions existantes
 - Degrés d'isolation
 - Protocoles multiversions
 - Modification des données / programmes

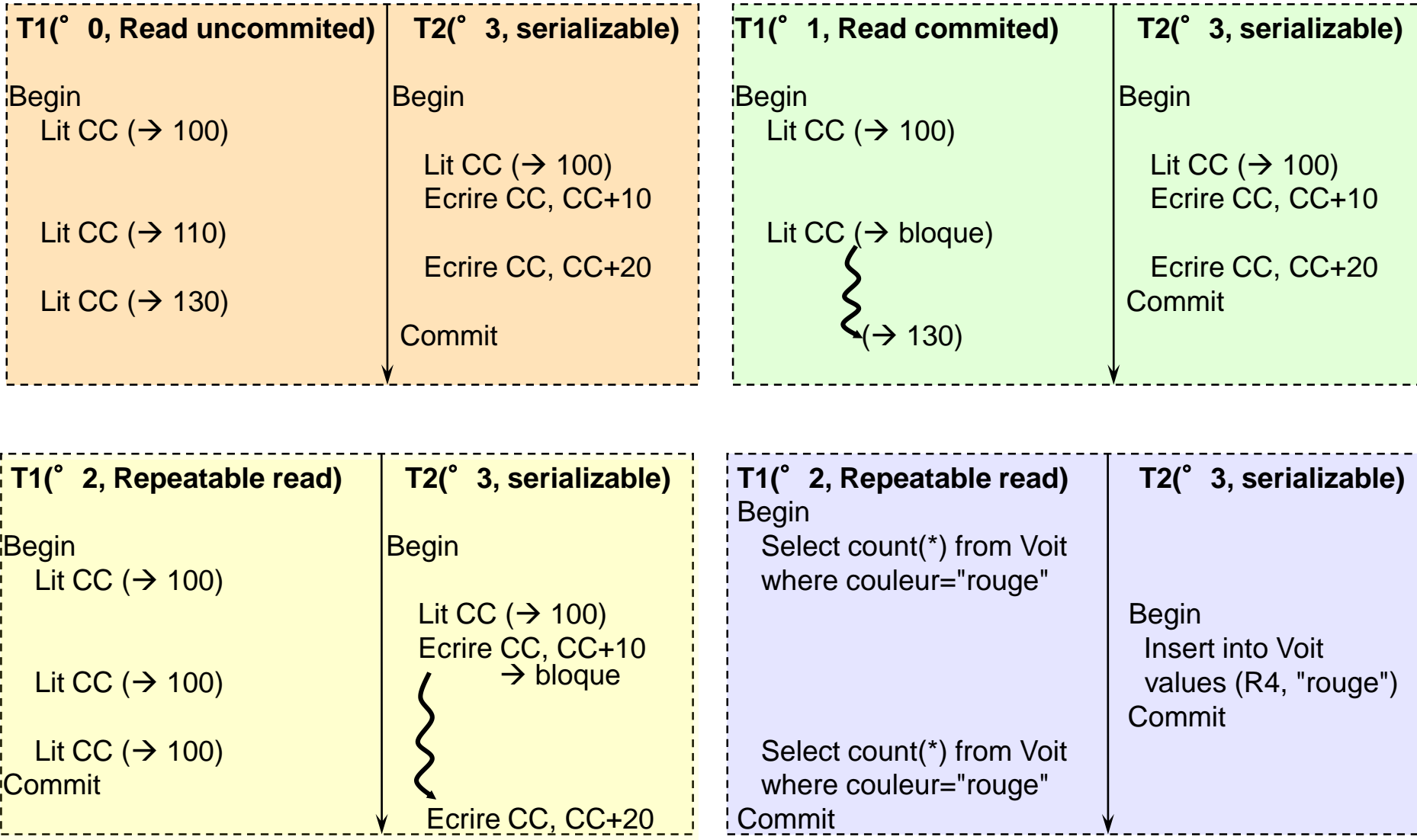
L'optimisation de la concurrence par l'exemple

- Scénario 1 :
 - Joe consulte son compte (lectures) au moment où sa banque le crédite de 500 euros (écritures)
→ Blocker le crédit ? Le montrer à Joe ?
- Scénario 2 :
 - Joe retire 200 euros (écritures) au même moment où sa banque le crédite de 500 euros (écritures)
→ Blocker le crédit ? Ne pas le perdre !
- Scénario 3 :
 - Une analyse des ventes (lectures) est faite dans un supermarché alors que les caisses sont ouvertes (écritures)
→ La lecture ne doit pas bloquer les caisses !
- Scénario 4 :
 - Les places de la finale de la coupe du monde sont mises en vente sur Internet le lundi à 8h (conflits massifs)
→ conflits massifs !
- Scénario 5 :
 - Réservation SNCF (lectures (choix) puis écritures (achat))
→ Comment éviter de bloquer tout le monde ?
- **Objectif :**
Chacun doit travailler en isolation i.e. comme si il était seul utilisateur de la base de données

Degrés d'isolation SQL – normalisés SQL2

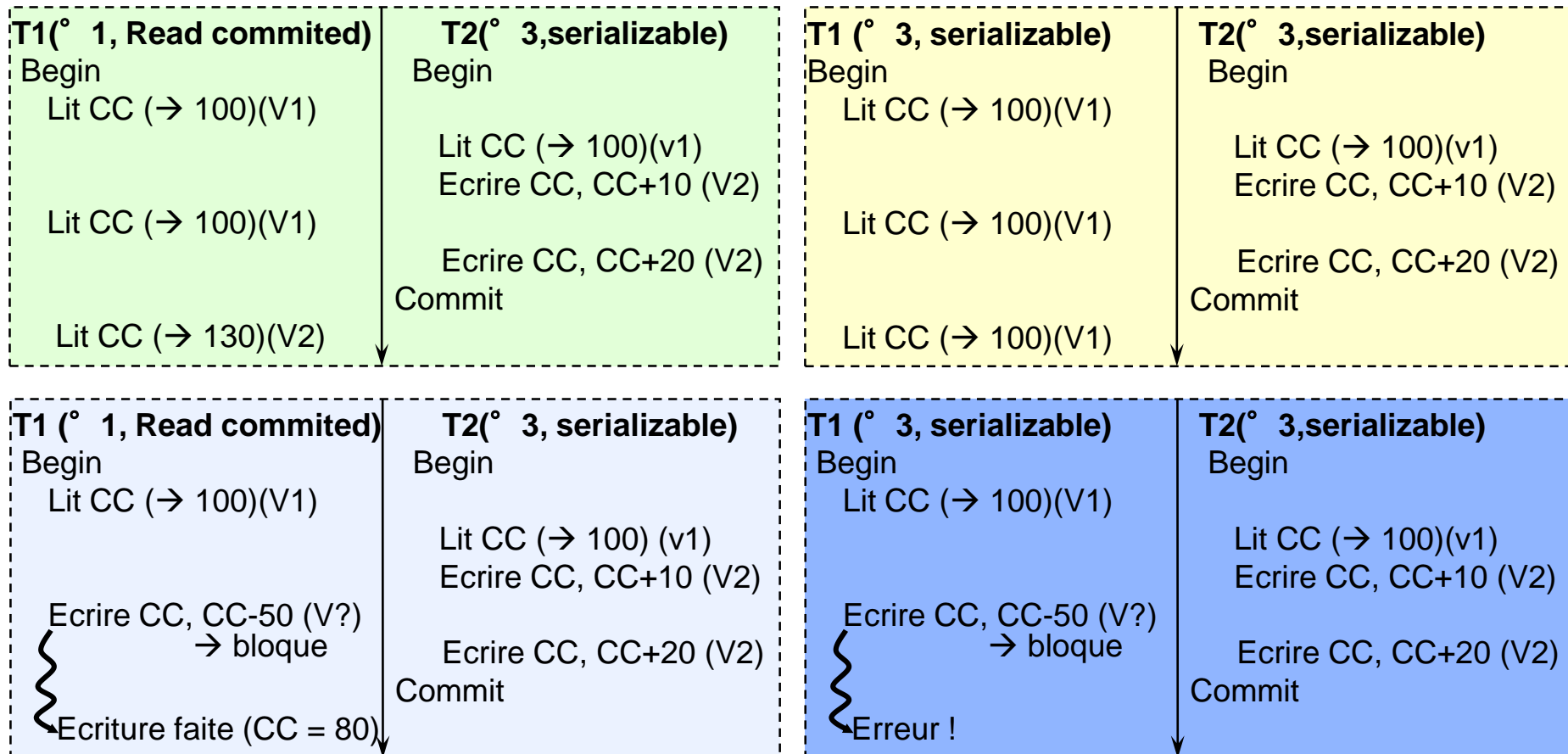
- Objectif: accroître le parallélisme en autorisant **certaines transactions** à violer la règle d'isolation
- Degrés standardisés SQL2
(Set Transaction Isolation Level)
 - **Degré 0 (Read Uncommitted)**
 - Lecture de données sales – Interdiction d'écrire.
 - *Ex. lecture sans verrouillage*
 - **Degré 1 (Read Committed)**
 - Lecture de données propres – Ecritures autorisées
 - *Ex. Verrous court en lecture, long en écriture*
 - **Degré 2 (Repeatable Read)**
 - Pas de lecture non reproductible
 - *Ex. Verrous longs en lecture et en écriture*
 - **Degré 3 (Serializable)**
 - Pas de requête non reproductible (fantôme)

Degrés d'isolation SQL : exemples



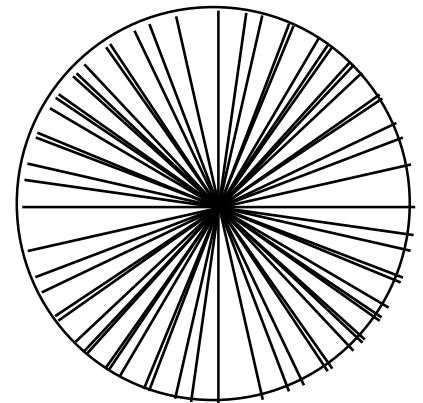
Protocoles multiversions (Oracle)

- Objectif : faire cohabiter sans blocage des transactions conflictuelles en les faisant travailler sur différentes versions des données



Modifications des données / Programmes

- **Scénario 4** : Les places de la finale de la coupe du monde sont mises en vente sur Internet le lundi à 8h
- Une seule donnée → Trop de conflits
- Exemple de solution :
 - Idée : partitionner le stade :
 - Chaque utilisateur se connectant est aléatoirement dirigé vers 1 des partitions.
 - Le degré de parallélisme est multiplié par le nombre de partitions.
- **Scénario 5** : Réservation SNCF (choix puis achat)
 - Si verrouillage dès le début, beaucoup de conflits
 - Sinon, risque de perdre la place convoitée
 - C'est le deuxième choix qui est implanté ...



Conclusion sur le verrouillage

- Scénario 1 : Compte Joe : lecture / écriture
 - ➔ Multiversion + Joe en degré Repeatable Read ou Read Committed
 - Scénario 2 : Compte Joe : écriture / écriture
 - ➔ Conflit inévitable quelque soit la méthode (mais rare)
 - Scénario 3 : lectures massives / écritures (supermarché)
 - ➔ Multiversion + degré Read Committed (peu de versions)
 - Scénario 4 : écritures massives (coupe du monde)
 - ➔ Partitionnement du stade
 - Scénario 5 : réservations SNCF (lectures puis écritures)
 - ➔ Transaction 1 = Phase de choix en Multiversion et Read Committed
 - ➔ Transaction 2 = Phase d'achat en Multiversion et Repeatable Read
- ➔ Il est donc particulièrement important de maîtriser les degrés d'isolations et les notions transactionnelles

Vrai ou faux ?

- Les protocoles de concurrence assurent entre autre l'Atomicité et la Cohérence des données
- On a besoin de protocole de concurrence seulement si plusieurs utilisateurs sont connectés en même temps à la base
- Le verrouillage assure la sérialisabilité des transactions
- Le verrouillage deux phases génère des verrou mortel
- Si un protocole de concurrence abandonne ma transaction, le système doit lui-même la relancer et la faire passer

Vrai ou faux ?

- T1 a un verrou en lecture sur D. T2 veut modifier cette donnée. Il demande un verrou en écriture. T1 est alors mise en attente et T2 obtient le verrou
- T1 démarre avant T2 mais se termine après T2. En cas de conflit, un algorithme de verrouillage sérialise T1 avant T2.
- Un verrou mortel ne met en jeu que deux transactions (et jamais plus)
- Un verrou mortel peut apparaître si une seule transaction lit et modifie des données, les autres ne faisant que lire
- En verrouillage deux phases, l'abandon d'une transaction peut entraîner l'abandon d'autres transactions

TD – Expérimentation sur la concurrence

Panorama des SGBD NoSQL



Panorama des SGBD NoSQL

Source : nosqldatabase.org

Hadoop / HBase	Couchbase Server	Riak	HyperDex	ArangoDB,
MapR	CouchDB	Redis	SharedHashFile	OrientDB
Hortonworks	ToroDB	Aerospike	Symas LMDB	Infinite Graph
Cloudera	SequoiaDB	LevelDB	Sophia	Sparksee
Cassandra	NosDB	RocksDB	NCache	TITAN
Scylla	RavenDB	Berkeley DB	TayzGrid	InfoGrid
Hypertable	MarkLogic Server	GenieDB	PickleDB	HyperGraphDB
Accumulo	Clusterpoint Server	BangDB	Mnesia	GraphBase
Amazon SimpleDB	JSON ODM	Chordless	LightCloud	Trinity
Cloudata	NeDB	Scalaris	Hibari	AllegroGraph
MonetDB	Terrastore	Tokyo Cabinet / Tyrant	OpenLDAP	BrightstarDB
HPCC	AmisaDB:	Scalien	Genomu	Bigdata
Apache Flink	JasDB	Voldemort	BinaryRage	Meronymy
IBM Informix	RaptorDB	Dynomite	Elliptics	WhiteDB
Splice Machine	djondb	KAI	DBreeze	Onyx Database
eXtremeDB	EJDB	MemcachedDB	TreodeDB	Virtuoso
ConcourseDB	densodb	Faircom C-Tree	BoltDB	VertexDB
Druid	SisoDB	LSM	Serenety	FlockDB
KUDU	SDB	KitaroDB	Cachelot	weaver
Elassandra	NoSQL embedded db	upscaledb	filejson	BrightstarDB
Document Store	ThruDB	STSdb	InfinityDB	Execom IOG
Elastic	iBoxDB	Tarantool/Box	KeyVast	Fallen 8
ArangoDB	BergDB	Chronicle Map	SCR Siemens	ArangoDB
OrientDB	ReasonDB	Maxtable	IOWOW	OrientDB
gunDB	IBM Cloudant	Pincaster	BBoxDB	FoundationDB
MongoDB	BagriDB	RaptorDB	NuSTER	Datomic
Cloud Datastore	DynamoDB	TIBCO Active Spaces	JDX	gunDB
Azure DocumentDB	Azure Table Storage	allegro-C	Antidote	CortexDB
RethinkDB		nessDB	Neo4J	Oracle NOSQL Database



Panorama des SGBD NoSQL

- **NoSQL = Not Only SQL** → Au-delà du SQL
- De quoi veut-on s'affranchir ?
 - Des lignes ?
 - Intérêt des bases de données orientées colonnes
 - Des colonnes ?
 - Systèmes clés-valeurs
 - De la structure ?
 - Document peu structurés (JSON)
 - Des transactions ACID ?
 - Principes BASE
 - De l'administration, de l'expertise ?
- Quelques produits
- Les tendances actuelles

Principes communs au NoSQL

- Partitionnement horizontal des données (Sharding)
 - Plusieurs milliers de nœuds
 - Architecture 'Shared nothing' / MPP
 - Ajout/suppression dynamique des nœuds
 - Equilibrage de charge automatique
- Réplication des données
 - Résolution des conflits (cohérence à terme)
- Parallélisme des traitements
 - Map/reduce (BigData de Google, Hadoop...)
 - Map: analyse/découpe le problème
 - Reduce: remonte des résultats

Théorème CAP

- *Proposé par Eric A. Brewer, Prof à Berkeley*

Référence : <https://people.eecs.berkeley.edu/~brewer/PODC2000.pdf>

- **Borne sup. de ce que peut faire un syst. distribué**
- Ce qu'un système distribué doit savoir faire :
 - **Consistency**
 - Tous les clients voient les mêmes données
 - /!\ \neq C de ACID (contraintes d'intégrité)
mais \approx A et I (les R/W sont atomiques et sérialisables)
 - **Availability**
 - Toute nœud du système, s'il n'est pas en panne, accepte toujours les requêtes R/W des clients
 - **Partition tolerance**
 - Resistance aux pertes de messages entre les partitions
 - NB : quid d'un SGBD relationnel ?

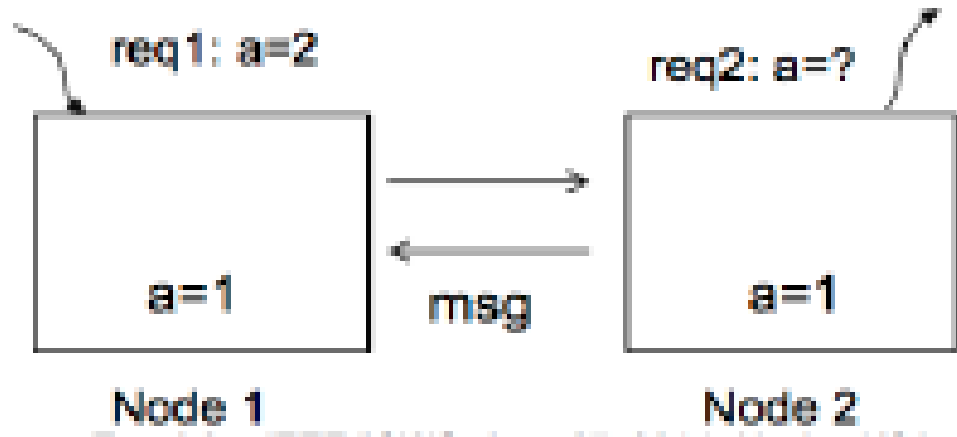
C&A&P non réalisable

- Req1 sur nœud 1 :

Ecrit a=2

- Req2 sur nœud 2 :

Lit a



- C&A ?
 - A → répondre à Req2 → renvoyer a=1
 - Mais C → attendre le message de nœud 1...
- A&P ?
 - A → répondre à Req2 → renvoyer a=1
 - Mais P → msg perdu → a=1 sur nœud 2...

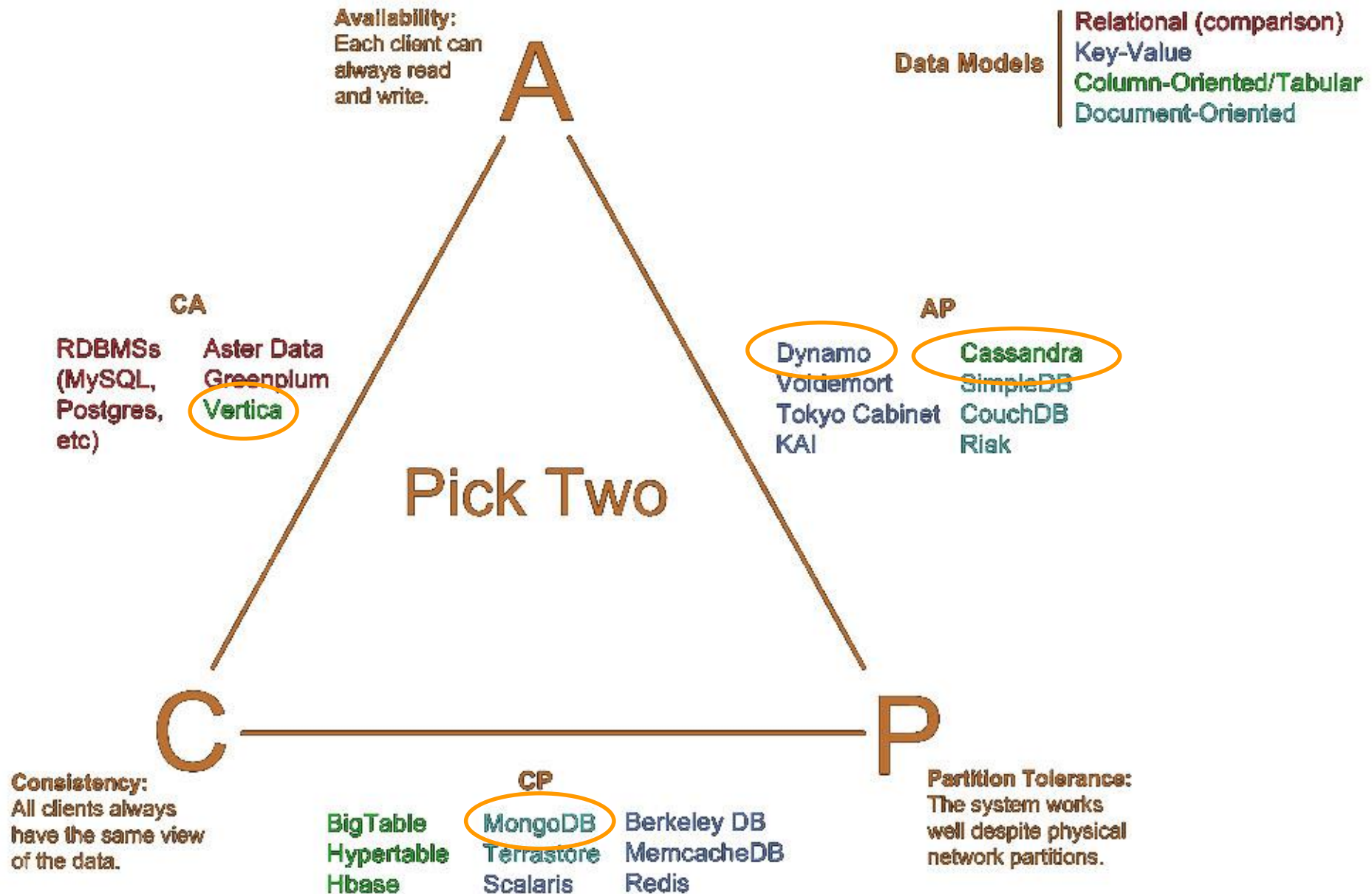
ACID vs BASE

- **ACID** = **A**tomique, **C**ohérent, **I**solé, **D**urable
 - Concept introduit par **Jim Grey** (Prix Turing)
- **CAP** = **C**onsistency, **A**vailability, **P**artition tolerance
 - Théorème CAP : impossible d'assurer les 3 !
 - Introduit par Eric Brewer (Prof. At Berkeley) en 2000
 - ➔ Systèmes **BASE** = **B**asically **A**vailable, **S**oft state, **E**ventual consistency

Classification des SGBD NoSQL

- SGBD colonnes (issus du relationnel)
 - MonetDB, VectorWise, C-Store, Vertica, Cassandra, ...
- SGBD documents
 - CouchDB, MongoDB, ElasticSearch...
- SGBD clé-valeur
 - Dynamo, Oracle NoSQL Database, Redis, ...
- Et bien d'autres aussi :
 - SGBD graphes (Apache Giraph, Neo4J, ...)
 - SGBD MultiValue (U2, Zoebase, ...)
 - Etc.

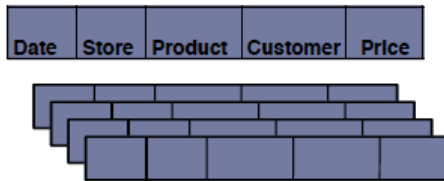
Visual Guide to NoSQL Systems



Principe des SGBD « colonnes »

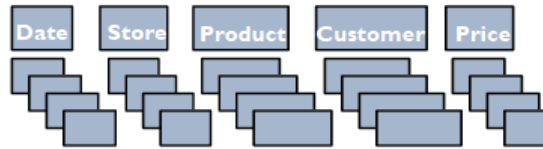
- Le stockage en colonne est adapté à des charges
 - Lecture intensive (voire lecture seule)
 - Grand volume de données

Stockage en ligne



- + Ajout/suppression simple
- Lecture de données inutiles

Stockage en colonnes

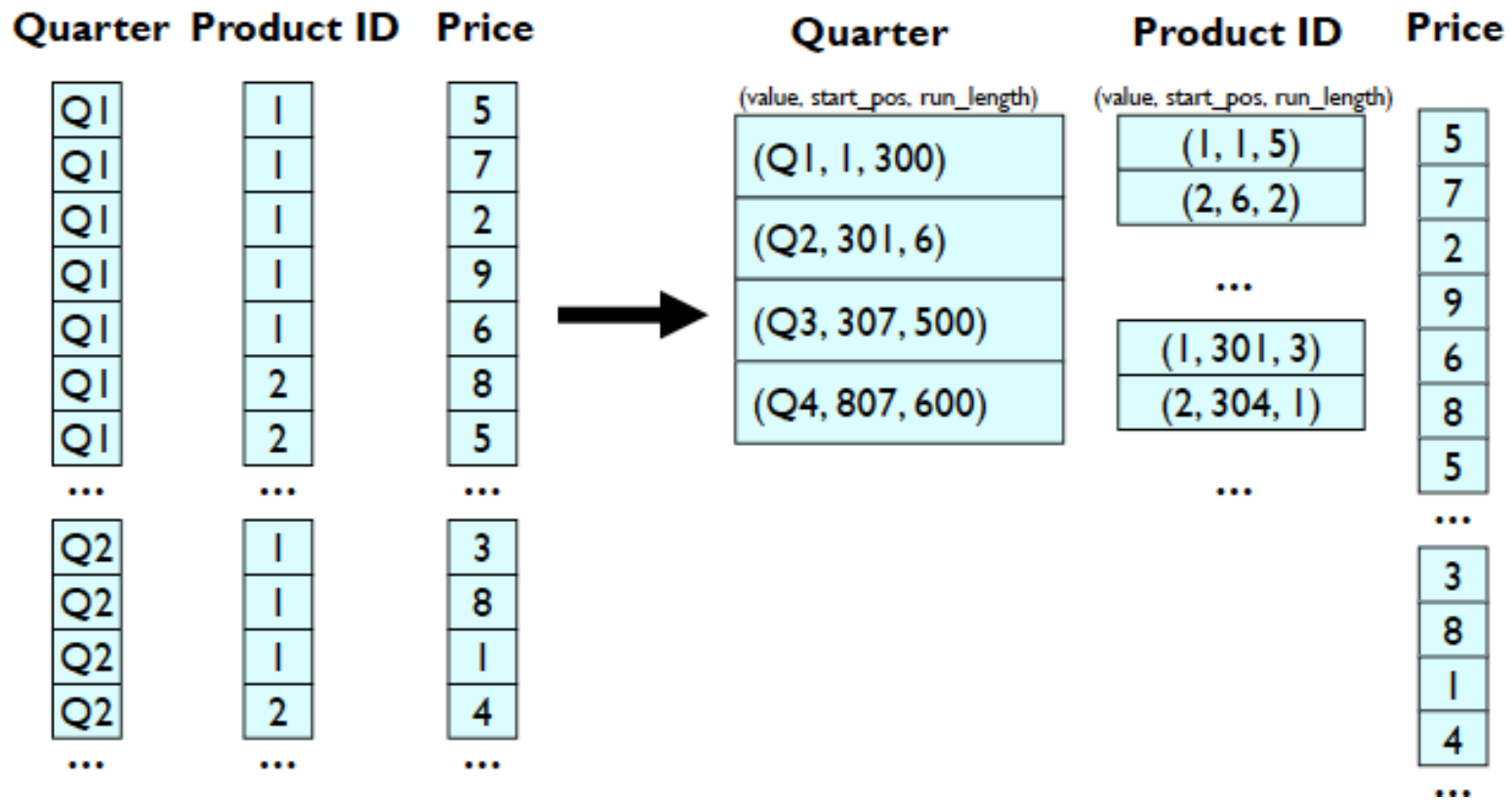


- + Accès aux données utiles
- Insertion d'un tuple → plusieurs accès

- OLAP, pas OLTP !

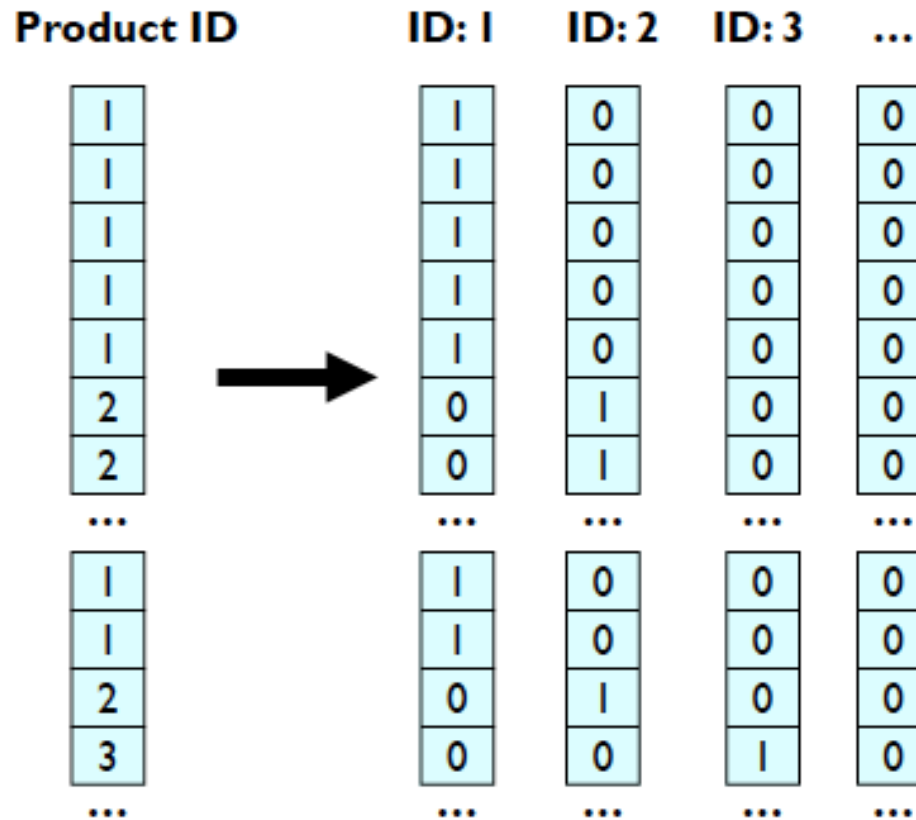
SGBD colonnes : compression (1)

- Run-Length Encoding (RLE)



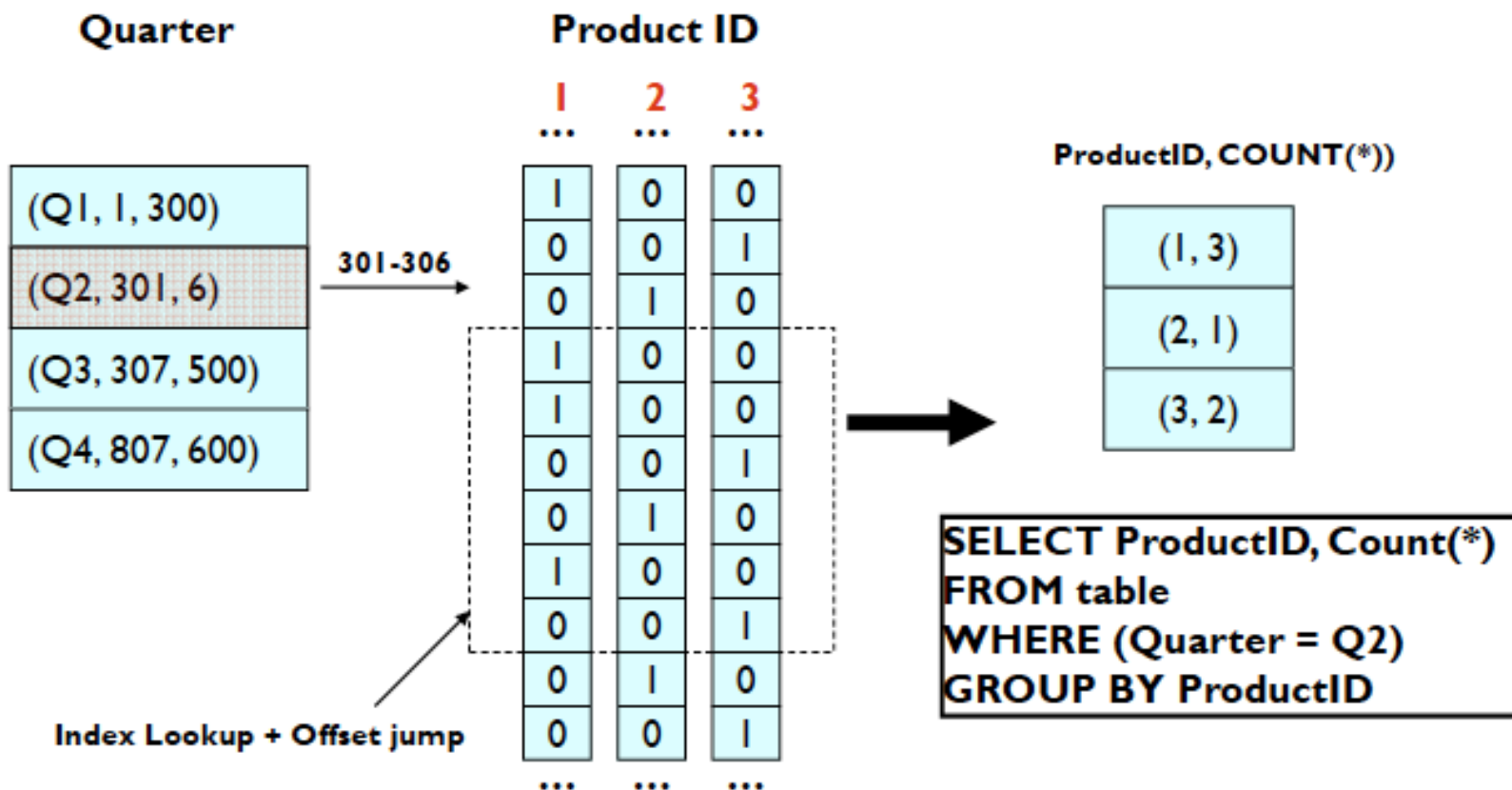
SGBD colonnes : compression (2)

- Vecteurs de bits




SGBD colonnes : exécution

- Exécution sur les données compressées



MonetDB / VectorWise



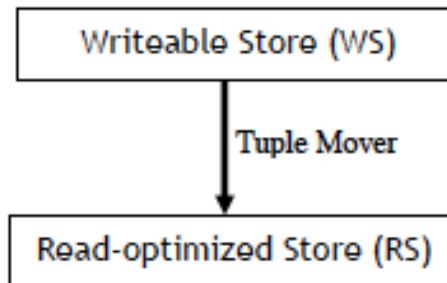
- Premier SGBD(R) **orienté colonnes**
- Support Relationnel (SQL) puis XML/Xquery
- Origine
 - Prototype de recherche très utilisé dans la communauté
 - Développé par le CWI (Amsterdam)
 - Open source en 2004, 10 year Best Paper Award en 2009
- Particularités
 - Efficacité (Colonnes, compression, maximisation caches CPU, traitements par « lots de tuples », ...)
- Base du produit VectorWise...
 - Startup du CWI (MonetDB ☺)
 - **Première place pour le Benchmark TPC-H !**
 - Racheté par INGRES (ACTIAN) en 2009

C-Store / Vertica

- Premier SGBD(R) orienté colonnes **sur le cloud**
- Travaux de recherche (VLDB'05) de M. Stonebraker
 - Développé par MIT, Yale, UMass ...
 - Version commerciale : Vertica racheté par HP



- Particularités



- C-Store : 2 'stores'
- shared nothing/MPP → cloud (AWS, Google, Azure)
- Scalabilité extrême (exascale)
- Hadoop nodes (MapReduce)

Cassandra (1)

- SGBD colonne NoSQL (A&P)
- Développé par Facebook
 - open source en 2008, projet apache en 2010
 - Utilisé par Netflix, Twitter, Spotify, etc.
- Réplication et partitionnement : basé sur l'anneau Dymano
- Modèle de données : basé sur BigTable
 - Des 'tables' peuvent être créées, supprimées ou modifiées pendant l'exécution, sans bloquer les modifications et les requêtes.
 - Les données sont stockées selon leur 'clé' dans les 'tables'
 - premier élément de la clé = clé de partition
 - dans chaque partition, les données sont groupées selon les autres colonnes composant la clé.
- Article de référence :
 - ["Cassandra - A Decentralized Structured Storage System", Avinash Lakshman et Prashant Malik, Facebook, 2009.](#)

Cassandra (2)

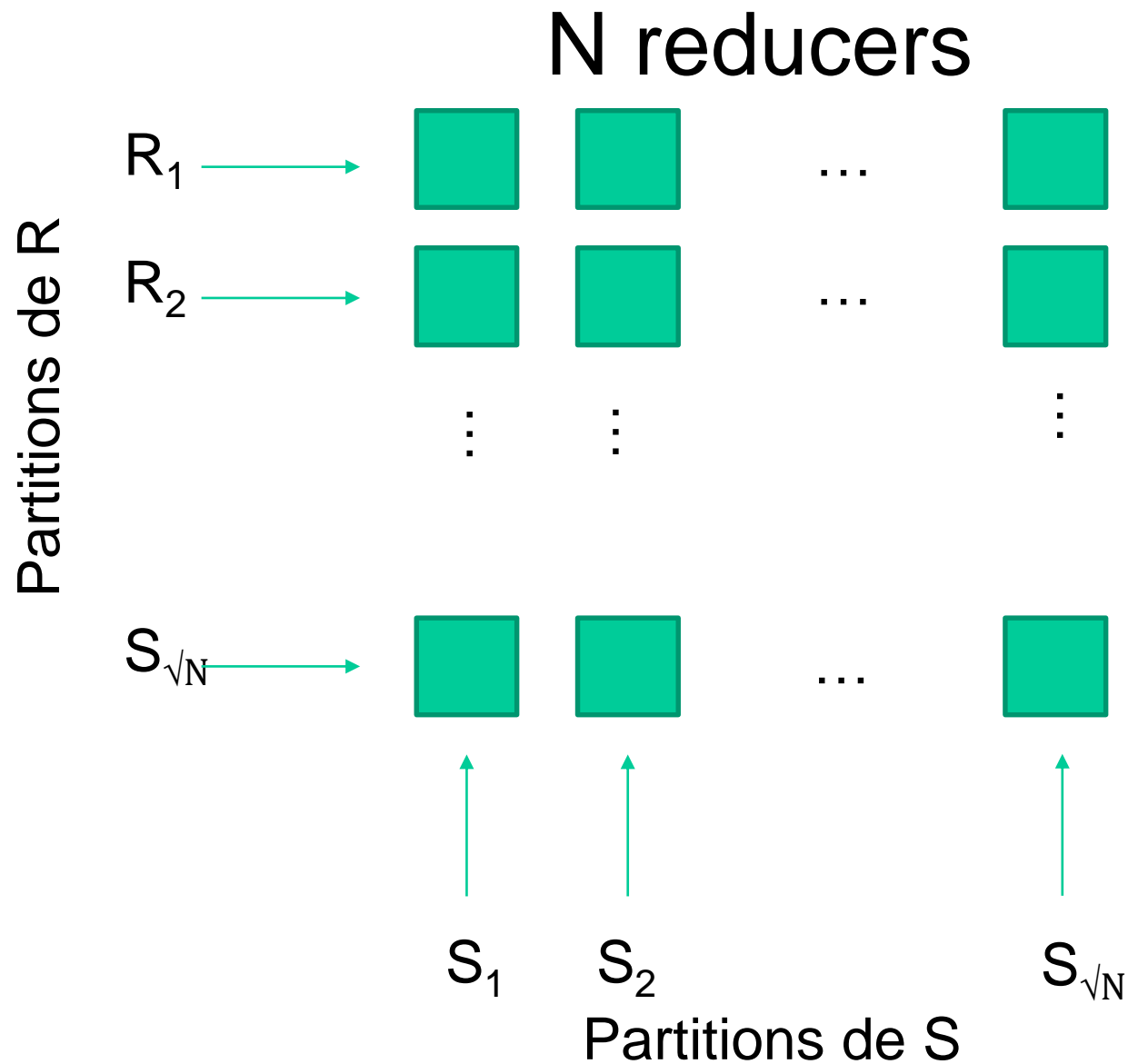
- Eventual consistency (Dynamo)
 - Données répliquées sur un anneau de serveurs
 - Gestion des conflits pour forcer la convergence
 - Échange des modifications entre les serveurs
 - Réconciliation en cas de reception de modification incohérentes
 - Ex: *last writer wins* basé sur des timestamps
 - Réconciliation asynchrone, ou en *Read repair* / *Write repair*
 - i.e., lors d'une lecture / écriture incohérente
- Langage CQL -SQL simplifié- :

```
CREATE COLUMNFAMILY MesColonnes (id text, Nom text,  
                                Prenom text, PRIMARY KEY(id));  
INSERT INTO MesColonnes (id, Nom, Prenom)  
VALUES ('1', 'Doe', 'John');  
SELECT * FROM MesColonnes;
```
- Pas de jointures ni sous-requêtes, mais support MapReduce

MapReduce ? un exemple...

- Ex : faire un produit cartésien $R \times S$ en Map/Réduce
- R et S sur un anneau de serveurs Cassandra
- Hypothèses
 - Des Mappers distribuent les tuples de R et de S
 - N Reducers 'collent' les tuples de R avec ceux de S
- Comment organiser le traitement pour le faire de façon optimale en 1 round?
 - But : minimiser les échanges de données entre mappers et reducers
 - Exemple de solution sous optimale :
 - Envoyer chaque élément de R à tous les reducers, envoyer chaque élément de S à un seul reducer

Solution

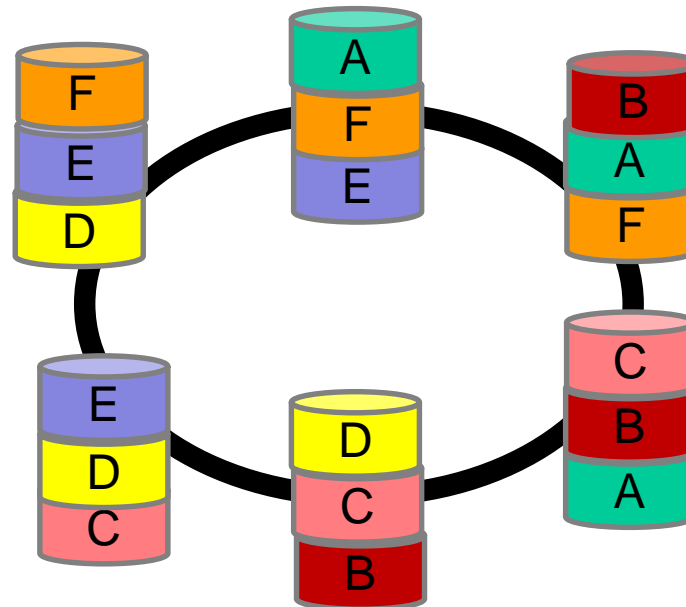
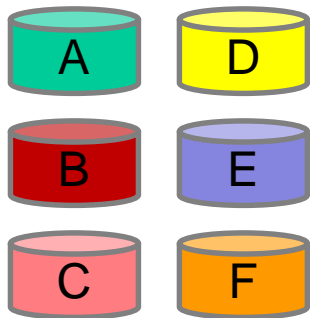


Principes des SGBD clé-valeur

- Modèle de données : (clé) → valeur
 - Users:2:friends → {23, 76, 233, 11}
 - Users:2:inbox → {234, 3476, 33, 55}
 - Users:2:settings → theme=dark, cookies=false
- La valeur est vue comme un blob opaque
- Interface CRUD (create, read, update, delete)
- Exemples : Amazon Dynamo (A&P), Redis (C&P), ...

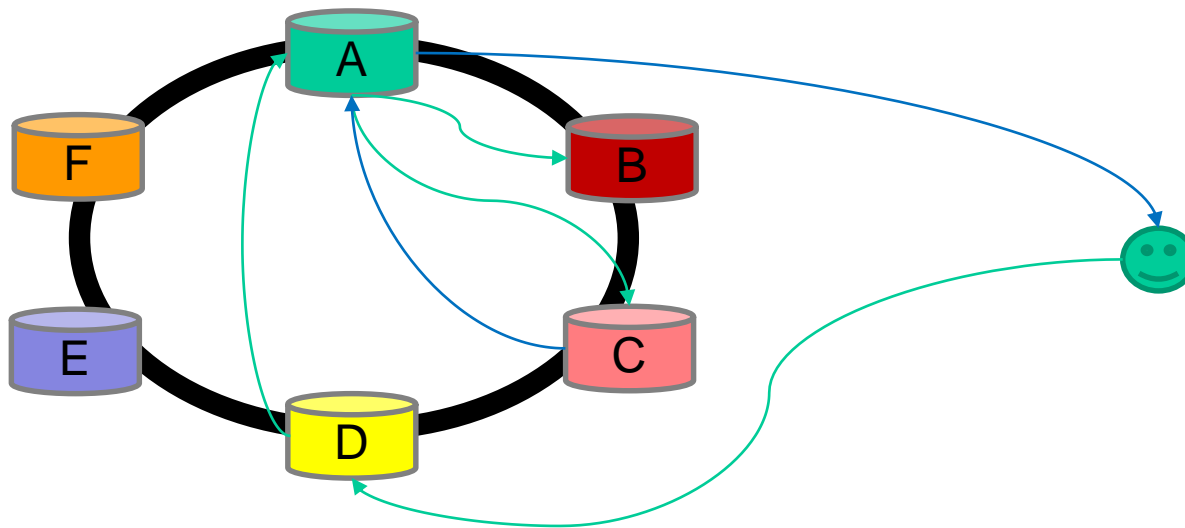
Dynamo

- Développé par Amazon (2007)
- Partitionnement des données sur un anneau de serveurs
- Chaque noeud stocke plusieurs partitions
- Chaque partition est répliquée N fois (ici 3)



Lecture

- Un noeud arbitraire sert de coordinateur
- Paramètres : N , R , W
 - N : nombre de replicas (ici 3)
 - R : nombre de noeuds devant confirmer la lecture (ici 2)
 - W : nombre de noeuds devant confirmer une écriture (ici 1)



Principes des SGBD documents

- Modèle de données : (collection, clé) → documents
- Exemple :
 - Order-12338 →

```
{  
  order-id: 23,  
  customer : { name : "Felix Marx", age : 25 }  
  line-items : [ {product-name = "x", ... } , ... ]  
}
```
- Interface CRUD, langage de requêtes, MapReduce
- Exemples : CouchDB (A&P), MongoDB (C&P)

MongoDB (1)

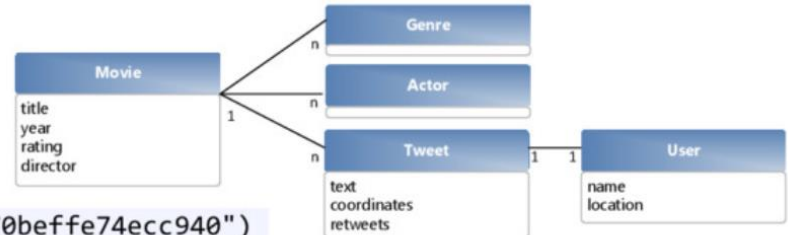
- Vient de “humongous” \approx gigantesque
- Partitionnement par intervalle ou basé sur du hachage
- Replication (synchrone ou asynchrone)
- Modèle de données : Json
 - Tout est clé valeur \Rightarrow “**clé**” : “**valeur**”
 - Un document est encapsulé par des accolades {...}
 - valeur : une valeur, une collection [...], un document
- Le principe est donc :
 - **Schema libre** : attributs par documents
 - **Dénormaliser** plutôt que de réaliser des jointures
 - “**Imbriquer**” pour remplacer les associations 1:n et 1:1
 - **Grain d’atomicité** : le document

MongoDB (2)

- Exemple de document

```
{
  "_id" : ObjectId("51a5d316d70beffe74ecc940")
  title : "Iron Man 3",
  year : 2013,
  rating : 7.6,
  director: "Shane Block",
  genre : [ "Action",
            "Adventure",
            "Sci -Fi"],
  actors : ["Downey Jr., Robert",
            "Paltrow , Gwyneth"],
  tweets : [ {
    "user" : "Franz Kafka",
    "text" : "#nowwatching Iron Man 3",
    "retweet" : false,
    "date" : ISODate("2013-05-29T13:15:51Z")
  } ]
}
```

Movie Document



- Interface : langage proche de javascript
 - db.movie.selectOne()
 - db.movie.find({ title : "Iron Man 3" }) ;
 - db.movie.find({ title : "Iron Man 3" }).count();

En conclusion

- S'affranchir des contraintes des systèmes SQL
 - Schéma statique, transactions ACID
 - Installation difficile, expertise extrême
 - Performance => cluster dédiés / mainframe
- Réponse NoSQL poussée par les géants du Web
 - Big Data : énormes quantité de données, générées par des machines
 - Faiblement structuré, scalabilité extrême
 - Mais : de l'expertise, des performances faibles vs les ressources...

Réf.: Gessert, F. et al. Scalable data management: NoSQL data stores in research and practice. *ICDE'16*
- Tendances : performances 'sub-second' + ACID + cloud
 - Convergence des systèmes : SGBD-R & NoSQL
 - Besoin d'acidité : MongoDB tourne ACID en 2018...
 - Dynamisme des ressources (allocation à la demande)
 - BFM Awards 2018 : Snowflake computing

TD – Expérimentation sur MongoDB