

Licence Mathématiques et Informatique - 3ème année
2022-2023

Bases de données relationnelles

Maude Manouvrier

- **Modèle relationnel**
- **Algèbre relationnelle**
- **Calcul relationnel**
- **SQL**
- **Modélisation Entité/Association et UML et passage au relationnel**
- **Dépendances fonctionnelles**
- **Décomposition de schéma**
- **Formes Normales**

BIBLIOGRAPHIE

Ouvrages de référence utilisés pour le cours et disponibles à la BU

J-L. Hainaut Bases de données - Concepts, utilisation et développement - 4e éd., InfoSup, Dunod, 2018, ISBN : 978-2100784608

T. Connolly, C. Begg et A. Strachan, *Database Systems A Pratical Approach to Desigh, Implementation and Management*, 6^{ème} édition, 2014, ISBN: 9780132943260

F. Brouad, R. Bruchez, C. Soutou, *SQL*, Syntex Informatique, Pearson, 2012, ISBN: 978-2-7440-7630-5

R. Ramakrishnan et J. Gehrke, *Database Management Systems*, Second Edition; McGraw-Hill, 2002, ISBN: 0-07-232206-3

A. Silberschatz, H.F. Korth et S. Sudarshan, *Database System Concepts*, McGraw-Hill, 7^{ème} édition, 2019, ISBN: 978-0073523323

J.D. Ullman et J. Widom, *A first Course in Database Systems*, Prentice Hall, 3eme édition, 2014, ISBN: 978-9332535206

BIBLIOGRAPHIE

Autres ouvrages de référence, disponibles à la BU :

C.J. Date, *An Introduction to Database Systems*, Addison Wesley

C.J. Date, *A Guide to SQL Standard*, Addison Wesley

R.A. El Masri et S.B. Navathe, *Fundamentals of Database Systems*, Prentice Hall

Ouvrages pédagogiques contenant des exercices corrigés :

Philip J. Pratt, *Initiation à SQL - Cours et Exercices corrigés*, Eyrolles, 2001

F. Brouard, C. Soutou, *ULM 2 pour les bases de données : Modélisation, normalisation, génération, SQL, outils*, Eyrolles, 2012

F. Brouard, C. Soutou, *SQL (Synthèse de cours et exercices corrigés)*. Pearson Education 2008

R. Stephens, R. Plew, A. Jones, Adapté par Nicolas Larrousse, *SQL*, Coll. Synthex, Pearson Education, 2012

Cours en ligne (avec vidéo) : <http://sql.bdpedia.fr/>

Chap. I - Introduction

Bases de données :

- **Collection homogène et structurée d'informations** ou de données qui existent sur une **longue période de temps** et qui décrivent les activités d'une ou plusieurs organisations
- Ensemble de données **modélisant les objets d'une partie du monde réel** et servant de support à une application informatique

Exemple 1 :

Organisation : une bibliothèque

Données : les livres, les emprunts, les emprunteurs

Exemple 2 :

Organisation : une Université

Données : les étudiants, les enseignants, les cours, etc.

SGBD (1/3)

Systemes de Gestion de Bases de Données (*DataBase Management Systems - DBMS*) :

Ensemble de logiciels systèmes permettant aux utilisateurs **d'insérer, de modifier, et de rechercher** efficacement des données spécifiques dans **une grande masse d'informations** (pouvant atteindre plusieurs milliards d'octets) **partagée par de multiples utilisateurs**

Exemples : MySQL, PostgreSQL (utilisé en TP), Oracle, Microsoft SQLServer, etc.

cf. <https://db-engines.com/en/ranking/relational+dbms>

Classement des SGBD en 2022

395 systems in ranking, August 2022

Rank			DBMS	Database Model	Score		
Aug 2022	Jul 2022	Aug 2021			Aug 2022	Jul 2022	Aug 2021
1.	1.	1.	Oracle	Relational, Multi-model	1260.80	-19.50	-8.46
2.	2.	2.	MySQL	Relational, Multi-model	1202.85	+7.98	-35.37
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	944.96	+2.83	-28.39
4.	4.	4.	PostgreSQL	Relational, Multi-model	618.00	+2.13	+40.95
5.	5.	5.	MongoDB	Document, Multi-model	477.66	+4.68	-18.88
6.	6.	6.	Redis	Key-value, Multi-model	176.39	+2.77	+6.51
7.	7.	7.	IBM Db2	Relational, Multi-model	157.23	-3.99	-8.24
8.	8.	8.	Elasticsearch	Search engine, Multi-model	155.08	+0.75	-2.01
9.	9.	10.	Microsoft Access	Relational	146.50	+1.41	+31.66
10.	10.	9.	SQLite	Relational	138.87	+2.20	+9.06

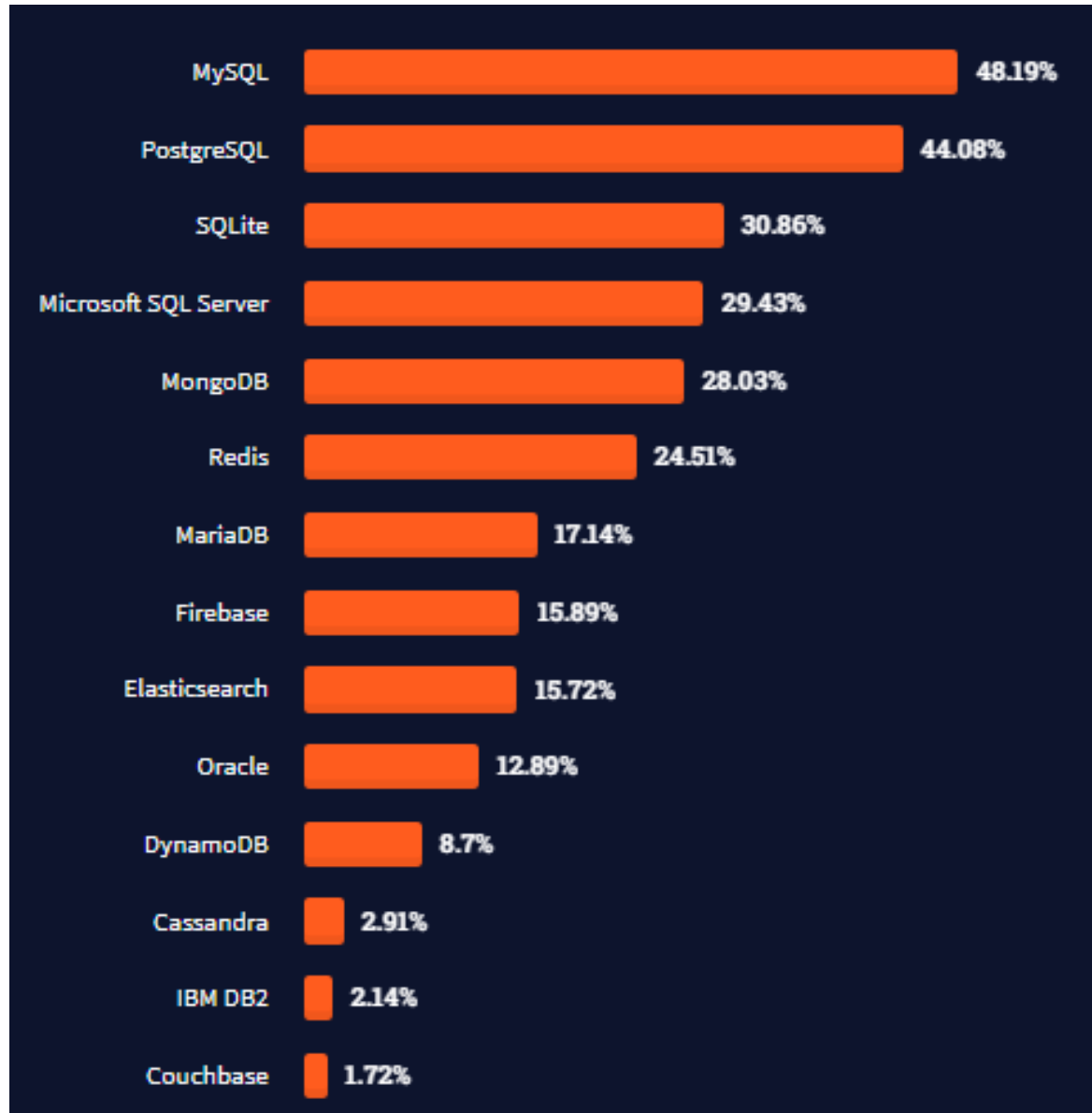
<https://db-engines.com/en/ranking>

Classement en fonction de la popularité (en fonction du résultats des recherches sur internet, des offres d'emploi, des questions sur les forum, des compétences sur les réseaux sociaux professionnels etc.)

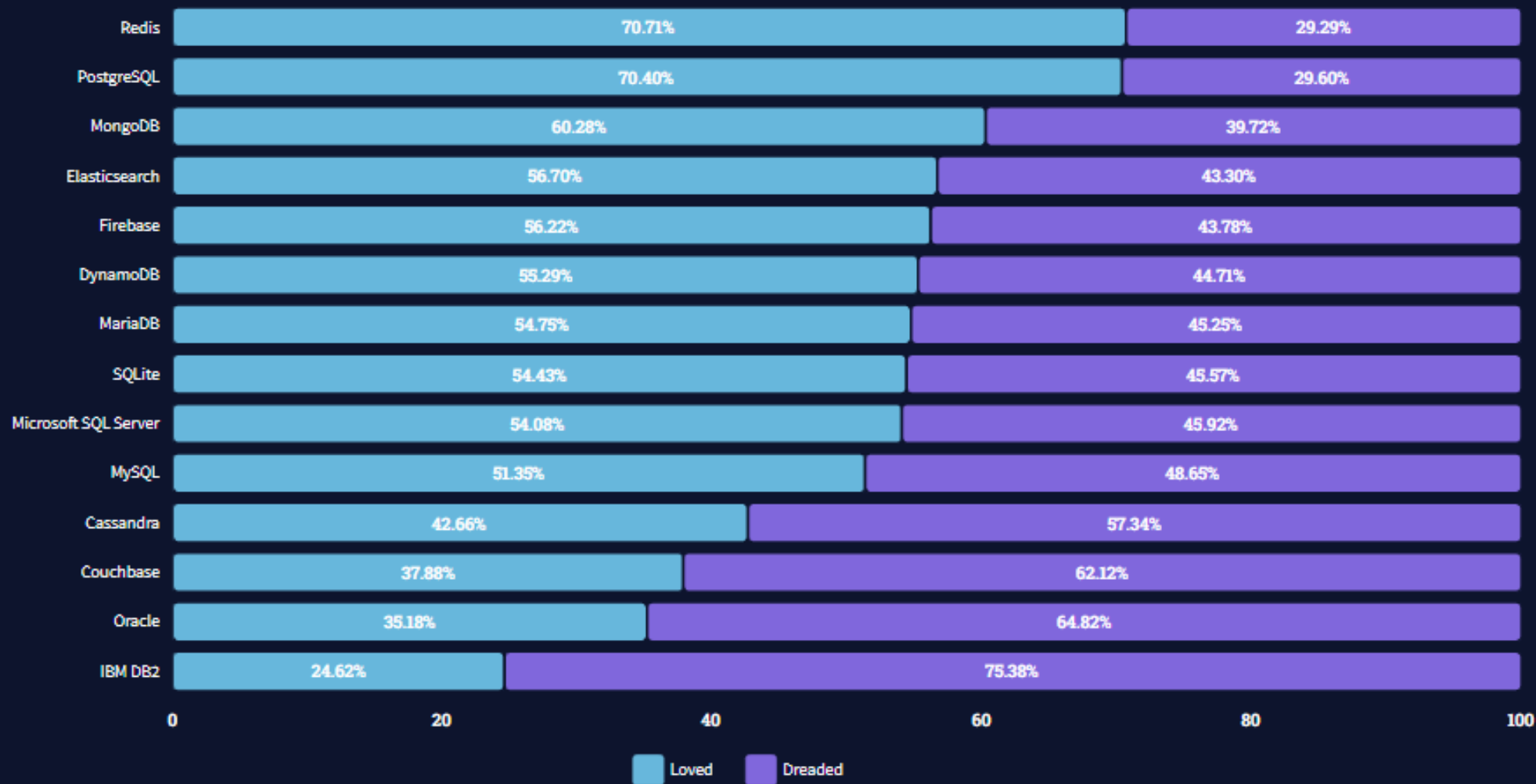
Bases de données les plus populaires en 2021

Etude *stackoverflow* basée sur les réponses de 53 312 réponses de développeurs issues de 180 pays

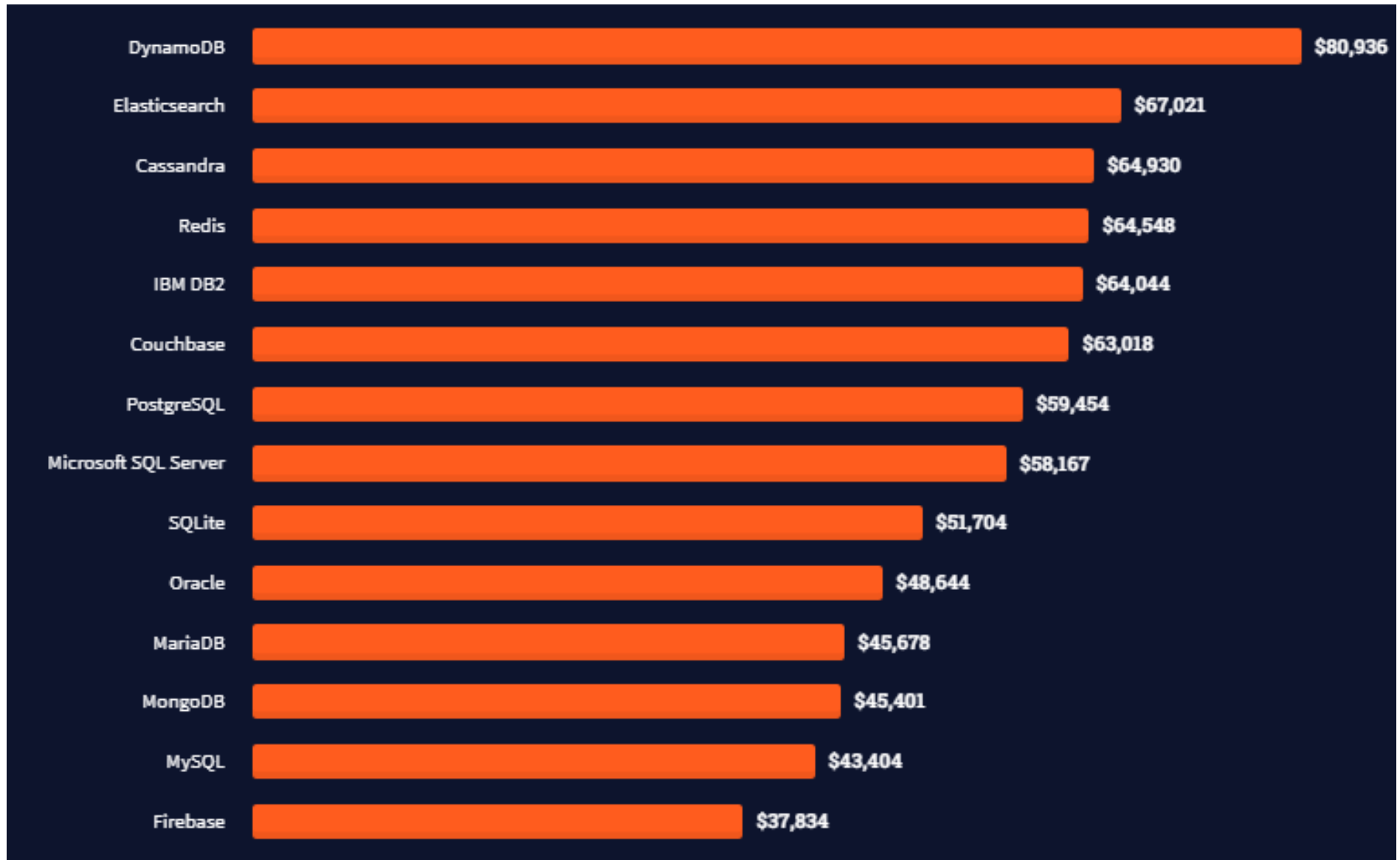
<https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies>



Appréciation/Détestation des bases de données en 2021



Profils BD les mieux payés en 2021



SGBD (2/3)

Principales fonctionnalités d'un SGBD :

- **Création et mises à jour de la structure de la base de données** (par le **concepteur** et/ou le **DBA *DataBase Administrator***)
- **Administration de la base de données** : gestion des utilisateurs, des droits d'accès etc. (par l'**administrateur – DBA**)
- **Saisie et mises à jour des données** (par le **concepteur** et/ou les **utilisateurs**)
- **Interrogation des données** selon différents critères et/ou en effectuant des calculs (par les **utilisateurs**)

SGBD (3/3)

Principaux composants :

- **Système de gestion de fichiers**
- **Gestionnaire de requêtes**
- **Gestionnaire de transactions**

Principales fonctionnalités :

- **Contrôle de la redondance d'information**
- **Partage des données**
- **Gestion des autorisations d'accès**
- **Vérifications des contraintes d'intégrité**
- **Sécurité et reprise sur panne**

Exemple de transaction

Virement d'une somme X d'un compte A vers un compte B :

1. Vérifier que $SoldeA \geq X$ (Lecture)
2. $SoldeA = SoldeA - X$ (Ecriture)
3. $SoldeB = SoldeB + X$ (Ecriture)

Atomicité : les 3 opérations seront effectuées ou aucune

Cohérence : la base est cohérente au début de la transaction et à la fin de son exécution

Isolation : les mises à jour de la transaction ne sont visibles qu'à la fin de son exécution

Durabilité : une fois validées, les mises à jours doivent être pérennes même en cas de panne.

Abstraction des données

- **Niveau interne ou physique :**
 - plus bas niveau
 - indique **comment** (avec quelles structures de données) sont stockées physiquement les données
- **Niveau logique ou conceptuel :**
 - décrit par un **schéma conceptuel**
 - indique quelles sont les données stockées et quelles sont leurs relations **indépendamment de l'implantation physique**
- **Niveau externe ou vue :**
 - propre à chaque groupe d'utilisateurs
 - décrit par un ou plusieurs **schémas externes**

Instances et schéma

- **Instances de base de données :**
 - données de la base à un instant donné
 - manipulées par un **langage de manipulation de données (DML - *Data Manipulation Language*)**
- **Schéma de base de données :**
 - description de la structure des données
 - ensemble de définitions exprimées en **langage de description de données (DDL - *Data Definition Language*)**

Petit historique

- **1960** : systèmes de gestion de fichiers
- **1970** : début des SGBD réseaux et hiérarchiques proches des systèmes de gestion de fichiers \Rightarrow pas d'interrogation sans savoir où est l'information recherchée ("navigation") et sans écrire de programmes
- **1970** : papier fondateur de CODD sur la théorie des relations
fondement de la théorie des bases de données relationnelles
INGRES à Berkeley - langage QUEL
System R IBM à San Jose, Ca. - langages SEQUEL et QBE
- **1980** : **Apparition des SGBD relationnels sur le marché** (Oracle, Ingres, Informix, Sybase, DB2 ...)
- **1990** : **début des SBGD orientés objet** (Gemstone, O₂, Orion, Objectstore, Versant, Matisse...).
- **Aujourd'hui** : **relationnel-objet, NoSQL et NewSQL**
- **cf . L'histoire des bases de données** <https://www.youtube.com/watch?v=iu8z5QtDQhY>

Chap II - Modèle relationnel

- **Données** : Ce que l'on stocke
- **Modèle relationnel** : Modèle permettant d'organiser les données en une **représentation schématique** qui autorisera son exploitation par le SGBD relationnel

Un livre de la BU (ayant un titre, un premier auteur et un ISBN) peuvent être empruntés par les étudiants (ayant un numéro de carte d'étudiant) etc.

Modèle relationnel

livre_id [PK] serial	titre character vai	isbn character vai
1	Livre BD	12-1334134
2	SQL	23-45546645

personne_id [PK] serial	nom character vai	prenom character	date_naissan date	adresse character vai	ville character varyin	code_postal character vai
1	GAMOTTE	Albert	03/12/1989	Rue des Alo	Paris	75012
2	COMPUTING	Claude	21/03/1999	Avenue Mach	Boulogne-Bill	92100

livre_id [PK] integer	personne_id [PK] integer	date_debut date	date_fin date
1	1	2014-10-11	14/10/2014
2	1	2014-11-03	

Relations

- Collection de **nuplets** (*tuples* en anglais) décrivant des données de même structure
- Tableau à deux dimensions composé d'**attributs** (ou champs - en colonnes) et de **nuplets** (ou enregistrements - en ligne)

enseignant_id [PK] serial	departement_id integer	nom character varying(25)	prenom character varyin	grade character vai	telephone character vai	fax character var	email character varying(100)
1	1	MANOUVRIER	Maude	MCF			manouvrier@lamsade.dauphine.fr
2	1	NEGRE	Elsa	MCF			negre@lamsade.dauphine.fr
3	1	BELHADJJAME	Khalid	MCF			Khalid.Belhajjame@dauphine.fr



Dans une relation :

- Pas de doublon
- Pas deux attributs de même nom

Modèle relationnel

- **Domaine** : ensemble de valeurs que peut prendre un attribut
- **Relation** : sous-ensemble du produit cartésien d'une liste de domaines caractérisé par **un nom unique**
 - représentée sous forme de **table à deux dimensions**
 - colonne = un domaine du produit cartésien
 - **un même domaine peut apparaître plusieurs fois**
 - ensemble de **nuplets sans doublon**
- **Attribut** : une colonne dans une relation
 - caractérisé par un nom et dont **les valeurs appartiennent à un domaine**
 - **de valeur atomique** (un attribut à une seule valeur / nuplet)
- **Nuplet** : une ligne d'une relation
 - correspondant à un enregistrement
 - **Pas de doublon** (les nuplets d'une relation sont tous différents)

Instances et schéma

- **Instances de base de données :**
 - les nuplets (les valeurs) contenus dans la base à un instant donné
- **Schéma de base de données :**
 - ensemble de **schémas de relation**
 - modélisation logique de la base de données à l'aide du modèle relationnel
- **Schéma de relation :**
 - liste d'attributs et leurs domaines

Clé

Attribut (ou ensemble d'attributs) permettant d'**identifier de manière unique** les nuplets de la relation

Exemples :

- *L'attribut ISBN pour une relation Livre*
- *L'attribut NuméroImmatriculation pour une relation Voiture*
- *L'attribut NuméroCarte pour une relation Emprunteur*

Par défaut : Création d'un attribut numérique s'incrémentant automatiquement

Clé artificielle
(*surrogate key*)

departement_id [PK] serial	nom_departement character varying(25)
1	MIDO
2	LSO
3	MSO



Une clé est unique (pas deux fois la même valeur) et a forcément une valeur (pas de valeur *null*)

Intégrité structurelle

■ Unicité des clés

- **Ensemble minimal d'attributs** dont la connaissance des valeurs permet d'identifier un nuplet unique de la relation considérée
- **R a pour clé K si :** $\forall t_1, t_2$ nuplets d'une instance de R
 $t_1.K \neq t_2.K$
- **Clé minimale :** si K est une clé minimale alors $\nexists K' \subset K$
tel que K' est une clé
- **Clé primaire :** une clé parmi toutes les clés minimales

Clé / Clé minimale

Personne (PersonneID, NSS, Nom, Prénom, Adresse)

Clés primaires possibles : PersonneID ou NSS

Clés non minimales : PersonneID + d'autres att. ou NSS + d'autres att.

Voiture (Immatriculation, Marque, Puissance, Type, Année, ProprioID)

Clé primaire : Immatriculation

Clés non minimales : (Immatriculation, Marque, Puissance, Type, Année, ProprioID)

Location (PersonneID, Immatriculation, Date)

Clé primaire : (Personne_ID, Immatriculation, Date)

Clé étrangère (FK – Foreign Key) (1/9)

Attribut (ou ensemble d'attributs) d'une relation qui fait (font) référence à la clé primaire d'une autre relation

A quoi cela sert ? *Exemple d'une mauvaise relation :*

NomDeFamille	Prénom	Adresse	DateDeNaissance	Type	ISBN	Titre
MANOUVRIER	Maude	Bureau P409bis, Univ. Paris-Dauphine	19/08/1973	Enseignant	2-3456-4567-7	Vives les Bases de Données
GAMOTTE	Albert	45, rue des Alouettes 75019 Paris	09/08/1989	Etudiant	2-7298-2012-4	Bases de données - Implémentation avec Access
SLATABLE	Deborah	24, avenue des Lilas 91650 Corbeil	31/03/1991	Etudiant	2-7298-2012-4	Bases de données - Implémentation avec Access
MANOUVRIER	Maude	Bureau P409bis, Univ. Paris-Dauphine	19/08/1973	Enseignant	2-7298-2012-4	Bases de données - Implémentation avec Access
MANOUVRIER	Maude	Bureau P409bis, Univ. Paris-Daphine	19/08/1973	Enseignant	2-6345-6567-6	Vives les bases de données

Problèmes :

- Répétition des noms, prénoms, dates de naissances, ISBN, etc.
autant de fois qu'il y a d'emprunts = **Redondance d'information**
- Comment identifier les nuplets ?



⇒ Ne pas mettre toutes les données dans une seule relation !!!

Clé étrangère (FK – Foreign Key) (2/9)

La solution ? Diviser les données en plusieurs relations

⇒ Division en 3 relations associées : *Personne*, *Emprunt* et *Livre*

⇒ Stockage unique des informations de chaque livre

livre_id [PK] serial	titre character var	isbn character var
1	Livre BD	12-1334134
2	SQL	23-45546645

⇒ Stockage unique des informations de chaque emprunteur

personne_id [PK] serial	nom character var	prenom character	date_naissan date	adresse character var	ville character varyin	code_postal character var
1	GAMOTTE	Albert	03/12/1989	Rue des Alo	Paris	75012
2	COMPUTING	Claude	21/03/1999	Avenue Mach	Boulogne-Bill	92100

⇒ Stockage unique des informations de chaque emprunt

livre_id [PK] integer	personne_id [PK] integer	date_debut date [PK]	date_fin date
1	1	2014-10-11	14/10/2014
2	1	2014-11-03	

Clé étrangère (FK – Foreign Key) (3/9)

Relation *Livre*

livre_id [PK] serial	titre character vari	isbn character vari
1	Livre BD	12-1334134
2	SQL	23-45546645

Relation *Emprunt*

livre_id [PK] integer	personne_id [PK] integer	date_debut date [PK]	date_fin date
1	1	2014-10-11	14/10/2014
2	1	2014-11-03	

Relation *Personne*

personne_id [PK] serial	nom character vari	prenom character	date_naissan date	adresse character vari	ville character varyin	code_postal character vari
1	GAMOTTE	Albert	03/12/1989	Rue des Alo	Paris	75012
2	COMPUTING	Claude	21/03/1999	Avenue Mach	Boulogne-Bill	92100

- L'attribut *livre_id* de la relation *Emprunt* est une clé étrangère qui fait référence à l'attribut *livre_id* de la relation *Livre*
- L'attribut *personne_id* de la relation *Emprunt* est une clé étrangère qui fait référence à l'attribut *personne_id* de la relation *Personne*

Clé étrangère (FK – Foreign Key) (4/9)

Intégrité référentielle : Ensemble de règles garantissant la **cohérence** (l'intégrité) des données réparties dans plusieurs relations

- Insertion dans la relation *Emprunt* : autorisée uniquement si on spécifie une valeur *personne_id* qui existe dans la relation *Personne* et une valeur *livre_id* qui existe dans la relation *Livre*

Relation *Livre*

livre_id [PK] serial	titre character var	isbn character var
1	Livre BD	12-1334134
2	SQL	23-45546645

Relation *Emprunt*

livre_id [PK] integer	personne_id [PK] integer	date_debut date [PK]	date_fin date
1	1	2014-10-11	14/10/2014
2	1	2014-11-03	

Relation *Personne*

personne_id [PK] serial	nom character var	prenom character	date_naissan date	adresse character var	ville character varyin	code_postal character var
1	GAMOTTE	Albert	03/12/1989	Rue des Alc	Paris	75012
2	COMPUTING	Claude	21/03/1999	Avenue Mach	Boulogne-Billi	92100

Clé étrangère (FK – Foreign Key) (5/9)

- Suppression d'un nuplet/Mise à jour de la valeur de clé primaire dans la relation *Personne* :
 - Non autorisée si un nuplet dans *Emprunt* fait référence au nuplet supprimé/mis à jour dans *Personne*

Suppression du nuplet 1 de Personne impossible car il existe des nuplets correspondants dans Emprunt

Relation *Emprunt*

livre_id [PK] integer	personne_id [PK] integer	date_debut date	date_fin date
1	1	2014-10-11	14/10/2014
2	1	2014-11-03	

Relation *Personne*

personne_id [PK] serial	nom character varying	prenom character varying	date_naissan date	adresse character varying	ville character varying	code_postal character varying
1	GAMOTTE	Albert	03/12/1989	Rue des Alo	Paris	75012
2	COMPUTING	Claude	21/03/1999	Avenue Mach	Boulogne-Bill	92100

Clé étrangère (FK – Foreign Key) (6/9)

- Suppression d'un nuplet/Mise à jour de la valeur de clé primaire dans la relation *Personne* :
 - Non autorisée si un nuplet dans *Emprunt* fait référence au nuplet supprimé/mis à jour dans *Personne*
 - Ou autorisée mais avec suppression/mise à jour en cascade des nuplets correspondant dans *Emprunt*

Suppression du nuplet 1 de Personne et en cascade des nuplets correspondants dans Emprunt

Relation *Emprunt*

livre_id [PK] integer	personne_id [PK] integer	date_debut date	date_fin date
1	1	2014-10-11	14/10/2014
2	1	2014-11-03	

Relation *Personne*

personne_id [PK] serial	nom character varying	prenom character varying	date_naissan date	adresse character varying	ville character varying	code_postal character varying
1	GAMOTTE	Albert	03/12/1989	Rue des Ais	Paris	75012
2	COMPUTING	Claude	21/03/1999	Avenue Mach	Boulogne-Bill	92100

Clé étrangère (FK – Foreign Key) (7/9)

- Suppression d'un nuplet/mise à jour de la valeur de la clé primaire dans la relation référencée autorisée avec affectation d'une valeur NULL ou d'une valeur par défaut à la clé étrangère des nuplets correspondants dans la relation référençant

Relation *Enseignant*

EnseignantID	DepartementID	Nom	Prenom	email
1	1	Bertrand	Patrice	Patrice.Bertrand@dauphine.fr
2	1	Hoffmann	Marc	Marc.Hoffmann@dauphine.fr
3	1	Belhajjame	Khalid	Khalid.Belhajjame@dauphine.fr
4	1	Manouvrier	Maude	manouvrier@lamsade.dauphine.fr
5	1	Negre	Elsa	negre@lamsade.dauphine.fr

Relation *Master*

MasterID	NomMaster	ResponsableID	DepartementID
1	ISF	1	1
2	Actuariat	2	1
3	MIAGE-SITN App.	3	1

Clé étrangère (FK – Foreign Key) (8/9)

- Suppression d'un nuplet/mise à jour de la valeur de la clé primaire dans la relation référencée autorisée avec affectation d'une valeur NULL ou d'une valeur par défaut à la clé étrangère des nuplets correspondants dans la relation référençant

Relation *Enseignant*

EnseignantID	DepartementID	Nom	Prenom	email
1	1	Bertrand	Patrice	Patrice.Bertrand@dauphine.fr
2	1	Hoffmann	Marc	Marc.Hoffmann@dauphine.fr
3	1	Belhajjame	Khalid	Khalid.Belhajjame@dauphine.fr
4	1	Manouvrier	Maude	manouvrier@lamsade.dauphine.fr
5	1	Negre	Elsa	negre@lamsade.dauphine.fr

Relation *Master*

MasterID	NomMaster	ResponsableID	DepartementID
1	ISF	1	1
2	Actuariat	2	1
3	MIAGE-SITN App.	NULL	1

Clé étrangère (FK – Foreign Key) (9/9)

- Suppression d'un nuplet/mise à jour de la valeur de la clé primaire dans la relation référencée autorisée avec affectation d'une valeur NULL ou d'une valeur par défaut à la clé étrangère des nuplets correspondants dans la relation référençant

Relation *Enseignant*

EnseignantID	DepartementID	Nom	Prenom	email
1	1	Bertrand	Patrice	Patrice.Bertrand@dauphine.fr
2	1	Hoffmann	Marc	Marc.Hoffmann@dauphine.fr
3	1	Belhajjame	Khalid	Khalid.Belhajjame@dauphine.fr
4	1	Manouvrier	Maude	manouvrier@lamsade.dauphine.fr
5	1	Negre	Elsa	negre@lamsade.dauphine.fr

Relation *Master*

MasterID	NomMaster	ResponsableID	DepartementID
1	ISF	1	1
2	Actuariat	2	1
3	MIAGE-SITN App.	-1	1

Contraintes d'intégrité :

toutes règles implicites ou explicites que doivent suivre les données

- **Contraintes d'identité**: tout nuplet doit posséder une valeur de clé primaire unique
- **Contraintes de domaine** : les valeurs de certains attributs doivent être prises dans un ensemble donné
- **Contraintes d'unicité** : une valeur d'attribut ne peut pas être affectée deux fois à deux entités différentes
- **Contraintes générales** : règle permettant de conserver la cohérence de la base de manière générale

Exemples de contraintes

– Contraintes de domaine :

"La fonction d'un enseignant à l'Université prend sa valeur dans l'ensemble {vacataire, moniteur, ATER, MCF, Prof., PRAG, PAST}."

– Contraintes d'unicité :

"Un département, identifié par son numéro, a un nom unique (il n'y a pas deux départements de même nom)."

– Contraintes générales :

"Un même examen ne peut pas avoir lieu dans deux salles différentes à la même date et à la même heure "

"La date de début d'emprunt doit être antérieure à la date de fin d'emprunt"

Intégrité référentielle et valeur NULL

■ Contraintes d'intégrité référentielle

Contrainte d'intégrité portant sur une relation R qui consiste à imposer que la valeur d'un groupe d'attributs apparait comme valeur de clé primaire dans une autre relation

■ Valeur nulle (NULL)

- valeur conventionnelle introduite dans une relation pour représenter une information inconnue ou inapplicable
- tout attribut peut prendre une valeur nulle **exceptés les attributs de la clé primaire** (contrainte d'entité)
- toute clé étrangère peut prendre une valeur nulle

Création d'une base de données (1/2)

Étape N° 1 : Concevoir la base de données

= Réfléchir à ce que va contenir la base de données et comment structurer les données

= Modélisation de la base de données

⇒ **Modèle conceptuel de données**

(Modèle Entité/Association ou UML - hors programme)

Démarche :

- Établir la liste des données devant être stockées dans la base
- Définir la structure des données

Création d'une base de données (2/2)

Étape N° 2 : Définir le modèle relationnel

= le **schéma** des relations de la base de données

Démarche :

- Pour chaque relation :
 - Définir les différents attributs
 - **Définir la clé primaire**
- Pour chaque attribut de chaque relation
 - **Définir le type et le domaine**
 - Préciser les propriétés (taille, format, etc.)
- Quand il y a plusieurs relations : **définir les clés étrangères**

Quelques règles

- Bien réfléchir aux schémas des relations et vérifier qu'ils sont corrects avant d'y insérer des données
- Utiliser des noms de relations et d'attributs compréhensibles (penser aux utilisateurs!!)
- Choisir le type de données adéquate pour chaque attribut
- Ne pas créer d'attribut de trop grande taille
- Ne pas créer d'attribut ayant des valeurs trop variables (ex. Age)
- Spécifier toutes les contraintes de domaines (en particulier quand l'ensemble de valeurs est limité), d'unicité etc.
- Préférer les clés primaires de type entier et en particulier des clés artificielles

Exemple de modèle relationnel (1/2) :

On veut créer une base de données stockant des enseignants (avec leur nom, prénom etc.) et des départements, chaque enseignant appartenant à un et un seul département.

Modèle relationnel correspondant :

Departement (DeptID, NomDept)

NomDept est unique et non NULL

Enseignant (EnsID, NUMEN, Nom, Prénom ..., #DeptID, Grade)

- *NUMEN est unique et non NULL*
- *#DeptID est une clé étrangère faisant référence à la clé primaire de Departement*
- *Grade \in {"Professeur", "MCF", "ATER" ...}*
- *Seuls Tel, Fax et Email peuvent prendre la valeur NULL*

Relation *Departement*

departement_id [PK] serial	nom_departement character varying(25)
1	MIDO
2	LSO
3	MSO



enseignant_id [PK] serial	departement_id integer	nom character varying(25)	prenom character varying(25)	grade character vai	telephone character vai	fax character vai	email character varying(100)
1	1	MANOUVRIER	Maude	MCF			manouvrier@lamsade.d
2	1	NEGRE	Elsa	MCF			negre@lamsade.dauphi
3	1	BELHADJAME	Khalid	MCF			Khalid.Belhajjame@da
4	1	COMPUTING	Claude	Moniteur			

Relation *Enseignant*

Exercice

On veut créer une base de données gérant des énoncés d'examens et des exercices.

Quelles sont les différences entre les modèles relationnels ?

Modèle relationnel 1 : Énoncé est UNIQUE

Examen (ExamID, Date, Heure)

Exercice (ExoID, Énoncé)

Contenu_Exam (#ExamID, #ExoID, Position)

Modèle relationnel 2 : Énoncé est UNIQUE

Examen (ExamID, Date, Heure)

Exercice (ExoID, Énoncé, #ExamID, Position)

Modèle relationnel 3 : Énoncé est UNIQUE

Examen (ExamID, Date, Heure)

Exercice (ExoID, #ExamID, Énoncé)

Rappel sur les clés étrangères

Une clé étrangère contient autant d'attributs que la clé primaire à laquelle elle fait référence.

Relation *Salle*

batiment [PK] character varying	numero_salle [PK] character varying	capacite integer
A	208	30
B	020	30
B	022	15
B	026	31
C	405	32
D	120	34



Reservation* contient une clé étrangère à 2 attributs faisant référence à la clé primaire de *Salle

Relation *Reservation*

reservation_id [PK] serial	batiment character varying	numero_salle character varying	enseignement_id integer	master_id integer	enseignant_id integer	date_resa date	heure_debut time without time zone	heure_fin time without time zone	nombre_heures integer
1	B	022	1	2	1	2012-01-15	08:30:00	11:45:00	3
2	B	020	2	2	1	2011-11-17	13:45:00	17:00:00	3

Langages d'interrogation

- **Algèbre relationnelle**

Opérations utilisées par le SGBD pour évaluer les requêtes

- **Calcul relationnel à variable nuplet**

Fondement logique du langage SQL

- **Calcul relationnel à variable domaine**

Fondement logique des langages de requêtes graphiques

- **SQL (*Structured Query Langage*)** - standard

Ces langages sont équivalents : ils permettent de désigner les mêmes ensembles de données

Requêtes et langage d'interrogation

- **Différents types de requête :**
 - Requêtes d'interrogation (*Data Manipulation Language*)
 - Requêtes d'insertion, de mise à jour et de suppression des données (*Data Manipulation Language*)
 - Requêtes de définition de schéma (*Data Definition Language*)

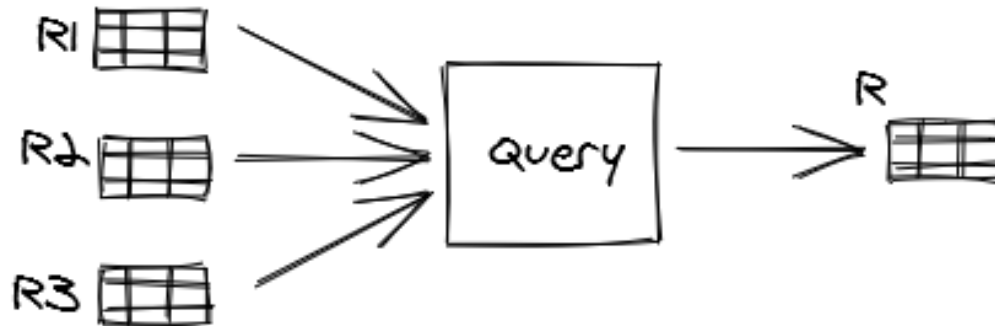
Chap III - Algèbre relationnelle

Requêtes d'interrogation

- Question sur les données
- Moyen d'extraction des données de la base en fonction de plusieurs critères

Requêtes d'interrogation :

- Entrée : une ou plusieurs instances de relations [+ prédicats]
- Sortie : une instance de relation temporaire (en mémoire)



Opérations unaires

- **Sélection** des nuplets satisfaisant un certain prédicat

Etudiant (Etudiant_ID, Nom, Prénom, Rue, Ville, Code-Postal, Téléphone, Fax, Email, NumAnnées)

$\sigma_{(Ville='Paris')}$ (Etudiant)

$\sigma_{(Ville='Paris') \wedge (NumAnnées \geq 2)}$ (Etudiant)

- **Projection** : élimination de certains attributs d'une relation

$\Pi_{Nom, Prénom}$ (Etudiant)

$\Pi_{Nom, Prénom} (\sigma_{(Ville='Paris')} (Etudiant))$

Exemples de projection et de sélection

Relation *Enseignant*

L..	enseignant_id (...)	departement_id ...	nom (varchar)	prenom ...	grade (...)	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
3	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr
4	4	1	LIMAM	Medhi	ATER			
5	5	5	MyTaylor	IsRich	Vacataire			
6	6	1	RIGAUX	Philippe	PROF			
7	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
8	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

Résultat de la sélection $\sigma_{(grade='MCF')}$ (*Enseignant*) :

L..	enseignant_id...	departement_id...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

Résultat de la projection

$\Pi_{Nom, Prenom}$ (*Enseignant*) :

L..	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	NAIJA	Yosr
3	BAHRI	Afef
4	LIMAM	Medhi
5	MyTaylor	IsRich
6	RIGAUX	Philippe
7	CHAKHAR	Salem
8	MURAT	Cécile

Résultat de la requête

$\Pi_{Nom, Prenom} (\sigma_{(grade='MCF')} (\textit{Enseignant}))$:

Ligne	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	MURAT	Cécile

Exemples de projection et de sélection

$$r : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array}$$

$$\pi_{B,A}(r) : \begin{array}{c|c} B & A \\ \hline b & a \\ a & d \\ b & c \end{array}$$

$$r : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array}$$

$$\sigma_{B=b}(r) : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ c & b & d \end{array}$$

Opérateurs binaires

- **Union** : { nuplets de 2 relations schéma-compatibles }

Enseignant (Enseignant_ID, Département_ID, Nom, Prénom, Grade, Téléphone, Fax, Email)

$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) \cup \Pi_{\text{Nom,Prénom}}(\text{Enseignant})$

- **Différence** : des nuplets de 2 relations schéma-compatibles

$\Pi_{\text{Nom,Prénom}}(\text{Enseignant}) - \Pi_{\text{Nom,Prénom}}(\text{Etudiant})$

- **Produit cartésien** : concaténation couples nuplets de 2 relations

Département (Département_ID, Nom_Département)

Schéma du Produit Cartésien de Enseignant \times Département :

(Enseignant_ID, Enseignant.Département_ID, Nom, Prénom, Grade, Téléphone, Fax, Email, Département.Département_ID, Nom_Département)



Exemple d'union

$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) :$

nom character varying (25)	prenom character varying (25)
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal
DEBECE	Gill
DEBECE	Aude

$\Pi_{\text{Nom,Prénom}}(\text{Enseignant}) :$

nom character varying (25)	prenom character varying (25)
MANOUVRIER	Maude
BELHAJJAME	Khalid
NEGRE	Elsa
MURAT	Cecile
DEBECE	Aude

$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) \cup \Pi_{\text{Nom,Prénom}}(\text{Enseignant}) :$

nom character varying (25)	prenom character varying (25)
DEBECE	Gill
MURAT	Cecile
GAMOTTE	Albert
BELHAJJAME	Khalid
ODENT	Jamal
MANOUVRIER	Maude
DEBECE	Aude
NEGRE	Elsa
HIBULAIRE	Pat

Exemple de différence

$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) :$

nom character varying (25)	prenom character varying (25)
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal
DEBECE	Gill
DEBECE	Aude

$\Pi_{\text{Nom,Prénom}}(\text{Enseignant}) :$

nom character varying (25)	prenom character varying (25)
MANOUVRIER	Maude
BELHAJJAME	Khalid
NEGRE	Elsa
MURAT	Cecile
DEBECE	Aude

$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) - \Pi_{\text{Nom,Prénom}}(\text{Enseignant}) :$

nom character varying (25)	prenom character varying (25)
DEBECE	Gill
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal

Produit cartésien

NSS	Nom	Prénom	Grade	Dept
12345	Manouvrier	Maude	MCF	1
45678	Toto	Titi	Prof	2

La relation Enseignant

Dept_ID	Nom_Dept_
1	Info
2	Math

La relation Département

NSS	Nom	Prénom	Grade	Dept	Dept_ID	Nom_Dept
12345	Manouvrier	Maude	MCF	1	1	Info
45678	Toto	Titi	Prof	2	1	Info
12345	Manouvrier	Maude	MCF	1	2	Math
45678	Toto	Titi	Prof	2	2	Math

La relation Enseignant \times Département

Autres opérateurs binaires

- **Renommage :**

$$\Pi_{A',B', \dots} (\mathbf{r}_{A \rightarrow A', B \rightarrow B', \dots})$$

$r(R)$: instance r de schéma R
 $s(S)$: instance s de schéma S

- **Intersection :**

$$\mathbf{r} \cap \mathbf{s} = \mathbf{r} - (\mathbf{r} - \mathbf{s})$$

- **Théta-jointure :**

$$\mathbf{r} \infty_{\Theta} \mathbf{s} = \sigma_{\Theta} (\mathbf{r} \times \mathbf{s})$$

- **Jointure naturelle :** $r(R)$ et $s(S)$ avec $R \cap S = \{A_1, A_2, \dots, A_n\}$

$$\mathbf{r} \infty \mathbf{s} = \sigma_{(R.A1=S.A1) \wedge (R.A2=S.A2) \wedge \dots \wedge (R.An=S.An)} (\mathbf{r} \times \mathbf{s})$$

Exemple de renommage et d'intersection

$\Pi_{\text{Last_Name, First_Name}}(\text{Enseignant}_{\text{Nom} \rightarrow \text{Last_Name}, \text{Prénom} \rightarrow \text{First_Name}})$:

last_name character varying (25)	first_name character varying (25)
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal
DEBECE	Gill
DEBECE	Aude

$\Pi_{\text{Nom, Prénom}}(\text{Enseignant}) \cap \Pi_{\text{Nom, Prénom}}(\text{Etudiant})$:

nom character varying (25)	prenom character varying (25)
DEBECE	Aude

Exemple d'union, intersection et différences (1/2)

$$r : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array}$$

$$s : \begin{array}{c|c|c} A & B & C \\ \hline b & g & a \\ d & a & f \end{array}$$

$$r \cup s : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \\ b & g & a \end{array}$$

$$r - s : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ c & b & d \end{array}$$

$$r \cap s : \begin{array}{c|c|c} A & B & C \\ \hline d & a & f \end{array}$$

Exemple d'union, intersection et différences (2/2)

Attention à l'ordre des attributs pour les opérateur d'union, de différence et d'intersection !

$$r : \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array}$$

$$s : \begin{array}{c|c|c} B & C & A \\ \hline g & a & b \\ a & f & d \end{array}$$

$$r \cup s \begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \\ g & a & b \\ a & f & d \end{array}$$

$r - s = r$

$$\begin{array}{c|c|c} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array}$$

$$r \cap s : \begin{array}{c|c|c} A & B & C \\ \hline & & \end{array}$$

Exemple de produit cartésien et de jointure naturelle

$$r : \begin{array}{c|c} A & B \\ \hline a & b \\ d & e \end{array}$$

$$s : \begin{array}{c|c} C & D \\ \hline c & d \\ f & g \end{array}$$

$$r : \begin{array}{c|c} A & B \\ \hline a & b \\ d & e \end{array}$$

$$s : \begin{array}{c|c} B & C \\ \hline b & f \\ g & h \end{array}$$

$$r \times s : \begin{array}{c|c|c|c} A & B & C & D \\ \hline a & b & c & d \\ a & b & f & g \\ d & e & c & d \\ d & e & f & g \end{array}$$

$$r \bowtie s : \begin{array}{c|c|c|c} A & B & B & C \\ \hline a & b & b & f \end{array}$$

Exemple de produit cartésien

La relation *Enseignant* :

Ligne	enseignant_id ...	departement_id ...	nom (varchar)	prenom ...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	4	1	LIMAM	Medhi	ATER			
3	5	5	MyTaylor	IsRich	Vacataire			
4	6	1	RIGAUX	Philippe	PROF			
5	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr
6	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
7	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
8	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr

La relation *Departement* :

Ligne	departement_id ...	nom_departement...
1	1	INFO
2	2	MATHS
3	3	GESTION
4	4	ECO
5	5	LANGUES
6	7	COMM
7	8	OPTION

Enseignant × *Departement* :

L...	enseignant_id...	departement_id (...	nom (varchar)	prenom...	grade...	telephone ...	fax ...	email (va...	departement_id...	nom_departement ...
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	1	INFO
2	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	2	MATHS
3	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	3	GESTION
4	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	4	ECO
5	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	5	LANGUES
6	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	7	COMM
7	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	8	OPTION
8	4	1	LIMAM	Medhi	ATER				1	INFO
9	4	1	LIMAM	Medhi	ATER				2	MATHS

Exemple de jointure

La relation *Enseignant* :

Ligne	enseignant_id ...	departement_id ...	nom (varchar)	prenom ...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	4	1	LIMAM	Medhi	ATER			
3	5	5	MyTaylor	IsRich	Vacataire			
4	6	1	RIGAUX	Philippe	PROF			
5	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr
6	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
7	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
8	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr

La relation *Departement* :

Ligne	departement_id ...	nom_departement...
1	1	INFO
2	2	MATHS
3	3	GESTION
4	4	ECO
5	5	LANGUES
6	7	COMM
7	8	OPTION

Enseignant \bowtie *Departement* :

L...	enseignant_id...	departement_id ...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (va...	departement_id...	nom_departement ...
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	1	INFO
2	4	1	LIMAM	Medhi	ATER				1	INFO
3	5	5	MyTaylor	IsRich	Vacata...				5	LANGUES
4	6	1	RIGAUX	Philippe	PROF				1	INFO
5	8	1	MURAT	Cécile	MCF			murat@la...	1	INFO
6	7	1	CHAKHAR	Salem	ATER			chakhar...	1	INFO
7	2	1	NAIJA	Yosr	Moniteur			naija@lm...	1	INFO
8	3	1	BAHRI	Afef	Moniteur			bahri@lm...	1	INFO

Théta-jointure

NSS	Nom	Prénom	Grade	Dept
12345	Manouvrier	Maude	MCF	1
45678	Toto	Titi	Prof	2

La relation Enseignant

Dept_ID	Nom_Dept_
1	Info
2	Math

La relation Département

NSS	Nom	Prénom	Grade	Dept	Dept_ID	Nom_Dept
12345	Manouvrier	Maude	MCF	1	1	Info
45678	Toto	Titi	Prof	2	2	Math

La relation Enseignant $\bowtie_{Dept=Dept_ID}$ *Département*

Exemple de Théta-jointure

r :	A	B	C
	1	2	3
	4	5	6
	7	8	9

s :	D	E
	3	1
	6	2

$r \bowtie_{B < D} s$:

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

Division (1/3)

Requête qui contient le terme « pour tous »

Soient $r(R)$ et $s(S)$ avec $S \subset R$

la relation $r \div s$ a pour schéma $R - S$

un nuplet t appartient à $r \div s$ si :

① $t \in \Pi_{R-S}(r)$

② $\forall t_s$ nuplet de s , $\exists t_r$ dans r qui satisfait :

◆ $t_r(S) = t_s(S)$ ou $\Pi_S(t_r) = \Pi_S(t_s)$

◆ $t_r(R-S) = t$ ou $\Pi_{R-S}(t_r) = t$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S} [(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)]$$

Exemple de division (1/2)

$$r : \begin{array}{c|c|c|c} A & B & C & D \\ \hline a & b & c & d \\ a & b & e & f \\ g & h & c & d \\ i & j & k & \ell \end{array}$$
$$s : \begin{array}{c|c} C & D \\ \hline c & d \\ e & f \end{array}$$
$$r \div s : \begin{array}{c|c} A & B \\ \hline a & b \end{array}$$

Exemple de division (2/2)

$$r : \begin{array}{c|c|c|c} A & B & C & D \\ \hline a & b & c & d \\ a & b & e & f \\ g & h & c & d \\ i & j & k & \ell \end{array}$$

$$s : \begin{array}{c|c} C & D \\ \hline c & d \\ e & f \end{array}$$

$$r \div s : \begin{array}{c|c} A & B \\ \hline a & b \end{array}$$

2eme exemple de division

Relation *Match* :

MID	DateMatch	Heure	Lieu
1	"21/01/17"	17h	Boulogne
2	"22/01/17"	15h	Asnières

Relation *Convocation* :

MID	JID
1	2
2	2

Relation *Joueur* :

JID	Nom	Prenom	AnneeNaissance
1	Comptuting	Claude	2005
2	Debece	Gilles	2008

Numéro des joueurs convoqués à tous les matchs :

$$\text{Convocation} \div \Pi_{MID}(\text{Match}) = \begin{array}{|c|} \hline \text{JID} \\ \hline 2 \\ \hline \end{array}$$

Noms et prénoms des joueurs convoqués à tous les matchs :

JID	Nom	Prenom	AnneeNaissance	MID	JID
2	Debece	Gilles	2008	1	2
2	Debece	Gilles	2008	2	2

$$\Pi_{Nom, Prenom, MID}(\text{Joueur} \bowtie \text{Convocation}) \div \Pi_{MID}(\text{Match}) = \begin{array}{|c|c|} \hline \text{Nom} & \text{Prenom} \\ \hline \text{Debece} & \text{Gilles} \\ \hline \end{array}$$

3^{ème} exemple de division

La relation *Enseignement* :

L.	enseignement_id ...	departement_id ...	intitule (varchar)	description (varchar)
1	①	①	Bases de Données	Niveau Licence : Modélisation E/A et UML, Modèle relationnel, Algèbre Relation...
2	②	①	Mise à Niveau Informatique	Pour les étudiants de GMI entrant directement en IUP2: Architecture, Algorithm...
3	③	①	Mise à Niveau Bases de Données	Pour les étudiants de DESS ID ou DEA127 - Programme Licence et Maîtrise en ...
4	①	⑤	Anglais	

La relation
Inscription :

Ligne	etudiant_id (int4)	enseignement_id (int4)	departement_id (int4)	date_inscription (date)
1	①	①	①	2004-02-25
2	①	②	①	2004-07-22
3	3	2	1	2004-07-22
4	5	2	1	2004-07-22
5	4	2	1	2004-07-22
6	①	③	①	2004-07-22
7	①	①	⑤	2004-07-22
8	2	1	5	2004-07-22

$\Pi_{Etudiant_ID, Enseignement_ID, Departement_ID} (Inscription) \div \Pi_{Enseignement_ID, Departement_ID} (Enseignement) :$

Ligne	etudiant_id (int4)
1	1

Contraintes et DF

- **Expressions des contraintes d'intégrité référentielle :**

$$\Pi_{\text{Departement_ID}}(\text{Enseignant}) \subseteq \Pi_{\text{Departement_ID}}(\text{Departement})$$

$$\Pi_{\text{Departement_ID}}(\text{Enseignant}) - \Pi_{\text{Departement_ID}}(\text{Departement}) = \emptyset$$

- **Expressions des dépendances fonctionnelles :**

$X \rightarrow Y \iff$ à une valeur de X est associée 1 et 1 seule valeur de Y

\iff si on connaît X on peut en déduire Y

$\iff \forall r$, une instance et $\forall t_1, t_2 \in r$, 2 nuplets de r , on a :

$$\Pi_X(t_1) = \Pi_X(t_2) \Rightarrow \Pi_Y(t_1) = \Pi_Y(t_2)$$

Chap IV - Calcul relationnel

- Langage déclaratif (ou non-procédural): le quoi
- Algèbre relationnelle (procédural) : le comment
- **Calcul à variable nuplet :**

$\{ \mathbf{t} / P(\mathbf{t}) \}$ signifie la requête renvoie un ensemble de nuplets \mathbf{t} vérifiant le prédicat $P(\mathbf{t})$ avec $P(\mathbf{t})$ contenant des conditions telles que :

- $r(\mathbf{t})$ indiquant que \mathbf{t} est un nuplet d'une instance r
- $\mathbf{t}.att_1 = valeur_1$
- $\mathbf{t}.att_1 > \mathbf{s}.att_2 \dots$

Exemples de requêtes en calcul

$$\sigma_{(\text{Ville}=' \text{ Paris } ') \wedge (\text{NumAnnées} \geq 2)} (\text{Etudiant})$$

$$\Leftrightarrow$$

$$\{t / \text{Etudiant}(t) \wedge (t.\text{Ville}=' \text{ Paris } ') \wedge (t.\text{NumAnnees} \geq 2) \}$$

$$\Pi_{\text{Nom,Prénom}} (\sigma_{(\text{Ville}=' \text{ Paris } ') } (\text{Etudiant}))$$

$$\Leftrightarrow$$

$$\{t.\text{Nom}, t.\text{Prenom} / \text{Etudiant}(t) \wedge (t.\text{Ville}=' \text{ Paris } ') \}$$

Quantificateurs

- $\exists \mathbf{t} (Q(\mathbf{t}))$: il existe un nuplet \mathbf{t} dans la base qui vérifie le prédicat $Q(\mathbf{t})$
- $\forall \mathbf{t} (Q(\mathbf{t}))$: pour tous les nuplets \mathbf{t} , le prédicat $Q(\mathbf{t})$ est vrai
- **Variable liée** : précédée d'un quantificateur
- **Variable libre** : non précédée d'un quantificateur

Expressions saines

- **Expression saine : ayant un sens en base de données**

`{t / Etudiant(t) ∧ (t.Ville=' Paris ') ∧ (t.NumAnnees ≥ 2)}`

- **Expression non saine : sans sens en bases de données**

`{t / ¬ Etudiant(t)}`

Expression des opérateurs algébriques (1/2)

- $\sigma_{\Theta}(\mathbf{r}) \Leftrightarrow \{ \mathbf{t} / \mathbf{r}(\mathbf{t}) \wedge \Theta(\mathbf{t}) \}$
- $\prod_{A1, A2, \dots, An}(\mathbf{r}) \Leftrightarrow \{ \mathbf{t}.A1, \mathbf{t}.A2, \dots, \mathbf{t}.An / \mathbf{r}(\mathbf{t}) \}$
avec $A1, A2, \dots, An \dots$ des attributs de R

Soient $R(A1, \dots, An)$ et $S(B1, \dots, Bn)$ schéma-compatibles

- $r \cup s \Leftrightarrow \{ \mathbf{t} / \mathbf{r}(\mathbf{t}) \vee \mathbf{s}(\mathbf{t}) \}$
- $r - s \Leftrightarrow \{ \mathbf{t} / \mathbf{r}(\mathbf{t}) \wedge \neg [\exists u \mathbf{s}(u) \wedge (\mathbf{t}.A1 = u.B1) \wedge \dots \wedge (\mathbf{t}.An = u.Bn)] \}$

Exemple d'écriture d'union et de différence

$$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) \cup \Pi_{\text{Nom,Prénom}}(\text{Enseignant})$$



$$\{ \mathbf{t.Nom, t.Prenom / Etudiant(t) \vee Enseignant(t)} \}$$

$$\Pi_{\text{Nom,Prénom}}(\text{Etudiant}) - \Pi_{\text{Nom,Prénom}}(\text{Enseignant}) :$$

$$\Leftrightarrow \{ \mathbf{t.Nom, t.Prenom / Etudiant(t) \wedge$$

$$\neg [\exists u \text{ Enseignant}(u) \wedge (t.Nom = u.Nom) \wedge$$

$$(t.Prenom = u.Prenom)$$

$$\mathbf{] } \}$$

Expression des opérateurs algébriques (2/2)

Soient $R(A_1, \dots, A_n)$ et $S(B_1, \dots, B_p)$

- $R \times S \iff \{t \mid \exists u, v \ r(u) \wedge s(v) \wedge (t.A_1 = u.A_1) \wedge \dots$
 $(t.A_n = u.A_n) \wedge (t.B_1 = v.B_1) \wedge \dots \wedge (t.B_p = v.B_p) \}$

Soient $R(A_1, \dots, A_n, C_1, \dots, C_m)$ et $S(B_1, \dots, B_p, C_1, \dots, C_m)$

- $R \circ S \iff \{t \mid \exists u, v \ r(u) \wedge s(v) \wedge (t.A_1 = u.A_1) \wedge \dots$
 $(t.A_n = u.A_n) \wedge (t.B_1 = v.B_1) \wedge \dots \wedge (t.B_p = v.B_p) \wedge (t.C_1 = u.C_1)$
 $\dots \wedge (t.C_m = u.C_m) \wedge (u.C_1 = v.C_1) \wedge \dots \wedge (u.C_m = v.C_m) \}$

Exemples d'écriture de produit cartésien et de jointure

Enseignant \times Departement

$$\Leftrightarrow \{t \mid \exists e, d \text{ Enseignant}(e) \wedge \text{Departement}(d) \wedge \\ (t.\text{EnseignantID}=e.\text{EnseignantID}) \wedge \dots \\ \wedge (t.\text{DepartementID}=d.\text{DepartementID}) \wedge \dots \}$$

Enseignant \bowtie Departement

$$\Leftrightarrow \{t \mid \exists e, d \text{ Enseignant}(e) \wedge \text{Departement}(d) \wedge \\ (t.\text{EnseignantID}=e.\text{EnseignantID}) \wedge \dots \\ \wedge (t.\text{DepartementID}=d.\text{DepartementID}) \wedge \dots \\ \wedge (e.\text{DepartementID}=d.\text{DepartementID}) \}$$

Exemple d'écriture de théta-jointure

$$r : \begin{array}{c|c|c} A & B & C \\ \hline 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}$$

$$s : \begin{array}{c|c} D & E \\ \hline 3 & 1 \\ 6 & 2 \end{array}$$

$$r \bowtie_{B < D} s : \begin{array}{c|c|c|c|c} A & B & C & D & E \\ \hline 1 & 2 & 3 & 3 & 1 \\ 1 & 2 & 3 & 6 & 2 \\ 4 & 5 & 6 & 6 & 2 \end{array}$$

$$r \infty_{B < D} s$$

$$\Leftrightarrow \{ t \mid \exists u, v \ r(u) \wedge s(v) \wedge (t.A = u.A) \wedge (t.B = u.B) \wedge (t.C = u.C) \wedge (t.D = v.D) \wedge (t.E = v.E) \wedge (u.B < v.D) \}$$

Division (1/3)

Exemple :

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID, ISBN, DateEmprunt, EtudiantID)

Etudiant(EtudiantID, Nom, Prenom)

« Quels sont les noms et prénoms des étudiants ayant emprunté tous les livres ? »

La requête retourne les valeurs des att. Nom et Prenom des nuplets t de la relation Etudiant tq :

$$\left\{ \overbrace{t.\text{Nom}, t.\text{Prenom}} \mid \text{Etudiant}(t) \wedge \left[\forall u \text{ Livre}(u) \left(\begin{array}{l} \Rightarrow (\exists v \text{ Emprunt}(v) \wedge \\ (v.\text{Etudiant_ID}=t.\text{Etudiant_ID}) \wedge (v.\text{ISBN}=u.\text{ISBN}) \end{array} \right) \right] \right\}$$

Quel que soit un nuplet u si c'est un nuplet de Livre (i.e. quel que soit un livre) alors

Il existe un nuplet v dans Emprunt associant le livre u à l'étudiant t

Division (2/3)

Exemple :

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID, ISBN, DateEmprunt, EtudiantID)

Etudiant(EtudiantID, Nom, Prenom)

« Quels sont les noms et prénoms des étudiants ayant emprunté tous les livres ? »

La requête retourne les valeurs des att. Nom et Prenom des nuplets t de la relation Etudiant tq :

$$\{ \text{t.Nom, t.Prenom} / \text{Etudiant}(t) \wedge [\forall u \neg (\text{Livre}(u)) \vee$$

$$(\exists v \text{Emprunt}(v) \wedge$$

$$(\text{v.Etudiant_ID}=\text{t.Etudiant_ID}) \wedge (\text{v.ISBN}=\text{u.ISBN})$$

$$)]$$

$$\}$$

Quel que soit u, u n'est pas dans Livre ou (u est dans Livre et) il existe v ...

Il existe un nuplet v dans Emprunt associant le livre u à l'étudiant t

Division (3/3)

$$\begin{array}{l}
 r : \begin{array}{c|c|c|c}
 A & B & C & D \\
 \hline
 a & b & c & d \\
 a & b & e & f \\
 g & h & c & d \\
 i & j & k & \ell
 \end{array}
 \quad
 s : \begin{array}{c|c}
 C & D \\
 \hline
 c & d \\
 e & f
 \end{array}
 \quad
 r \div s : \begin{array}{c|c}
 A & B \\
 \hline
 a & b
 \end{array}
 \end{array}$$

\mathbf{t} \mathbf{v}_1 \mathbf{v}_2 \mathbf{u}_1 \mathbf{u}_2

$r \div s$

$$\Leftrightarrow \{ \mathbf{t.A}, \mathbf{t.B} / \mathbf{r(t)} \wedge [\forall \mathbf{u} \mathbf{s(u)} \Rightarrow \exists \mathbf{v} \mathbf{r(v)} \wedge (\mathbf{t.A=v.A}) \wedge (\mathbf{t.B=v.B}) \wedge (\mathbf{v.C=u.C}) \wedge (\mathbf{v.D=u.D})] \}$$

$$\Leftrightarrow \{ \mathbf{t.A}, \mathbf{t.B} / \mathbf{r(t)} \wedge [\forall \mathbf{u} \neg (\mathbf{s(u)}) \vee (\exists \mathbf{v} \mathbf{r(v)} \wedge (\mathbf{t.A=v.A}) \wedge (\mathbf{t.B=v.B}) \wedge (\mathbf{v.C=u.C}) \wedge (\mathbf{v.D=u.D}))] \}$$

Expression de la division

Soient $R(A1, \dots, An, C1, \dots, Cm)$ et $S(C1, \dots, Cm)$

$$\bullet \quad R \div S \Leftrightarrow \{t.A1, \dots, t.An \mid r(t) \wedge$$

$$\quad [\forall u \ s(u) \Rightarrow \exists v \ r(v) \wedge (v.C1=u.C1) \wedge \dots \wedge$$

$$\quad (v.Cm=u.Cm) \wedge (t.A1=v.A1)$$

$$\quad \wedge \dots \wedge (t.An=v.An)$$

$$\quad] \}$$

$$\Leftrightarrow \{t.A1, \dots, t.An \mid r(t) \wedge$$

$$\quad [\forall u \neg (s(u) \vee (\exists v \ r(v) \wedge (v.C1=u.C1) \wedge \dots \wedge$$

$$\quad (v.Cm=u.Cm) \wedge (t.A1=v.A1)$$

$$\quad \wedge \dots \wedge (t.An=v.An))] \}$$

Expression des contraintes en logique des prédicats

Agence (nom_banque, ville ...)

Emprunt (num_emprunt, nom_banque, num_client, montant ...)

Compte (nom_banque, num_client, num_compte, solde ...)

« Chaque emprunteur possède un compte en banque dans l'agence dont le solde est au minimum égal à la moitié de son emprunt »

$$\neg (\exists e \text{ Emprunt}(e) \wedge \neg (\exists c \text{ Compte}(c) \\ \wedge (c.\text{num_client} = e.\text{num_client}) \\ \wedge (c.\text{nom_banque} = e.\text{nom_banque}) \\ \wedge (c.\text{solde} \geq (e.\text{montant} / 2) \\))$$

Chap V - Algèbre relationnelle étendue

- **Projection généralisée :**

ajout d'expressions arithmétiques dans une projection

Compte

nomclient	credit	debit
Gamotte	2000	560
Debece	1352	850

$\Pi_{\text{NomClient}, (\text{Credit} - \text{Debit}) \rightarrow \text{Solde}} (\text{Compte})$

nomclient	solde
Gamotte	1440
Debece	502

Jointure externe

- **Jointure externe** (*outer-join*) :

- jointure externe à gauche : $]^\infty$

- jointure externe à droite : $^\infty[$

- jointure externe : $]^\infty[$

$\mathbf{r}]^\infty \mathbf{s} \Rightarrow \mathbf{r} \infty \mathbf{s}$ et conservation des attributs des nuplets de \mathbf{r} qui ne joignent avec aucun nuplet de \mathbf{s} (les valeurs des attributs de \mathbf{s} sont mises à NULL)

$\mathbf{r} \infty[\mathbf{s} \Rightarrow \mathbf{r} \infty \mathbf{s}$ et conservation des attributs des nuplets de \mathbf{s} qui ne joignent avec aucun nuplet de \mathbf{r} (les valeurs des attributs de \mathbf{r} sont mises à NULL)

$\mathbf{r}]^\infty[\mathbf{s} = (\mathbf{r}]^\infty \mathbf{s}) \cup (\mathbf{r} \infty[\mathbf{s})$

Exemple de jointure externe

Personnel

Nom_Employé	Ville
Tom	Marseille
Jerry	Paris
Alex	Limoges
Marthe	Perpignan

Employé

Nom_Employé	Filiale	Salaire
Tom	SUD_EST	10000
Jerry	IDF	25000
Sophie	IDF	15000
Marthe	SUD_OUEST	12000

Personnel

]∞

Employé

Nom_Employé	Ville	Filiale	Salaire
Tom	Marseille	SUD_EST	10000
Jerry	Paris	IDF	25000
Alex	Limoges	NULL	NULL
Marthe	Perpignan	SUD_OUEST	12000

Personnel

∞[

Employé

Nom_Employé	Ville	Filiale	Salaire
Tom	Marseille	SUD_EST	10000
Jerry	Paris	IDF	25000
Sophie	NULL	IDF	15000
Marthe	Perpignan	SUD_OUEST	12000

Fonctions d'agrégation

- Somme des places disponibles dans l'Université

Sum_{Capacite}(Salle)

- Nombre moyen de places disponibles dans les salles de l'Université

Avg_{Capacite}(Salle)

- Nombre d'étudiants à l'Université

Count_{Etudiant_ID}(Etudiant)

- Capacité de la plus petite/grande salle

Min_{Capacite}(Salle)

Max_{Capacite}(Salle)

- Nombre d'enseignants par départements :

Group Nom_Departement, **Count** Enseignant_ID (Enseignant ∞ Departement)

Exemple de regroupement

Le regroupement des enseignants par département

EnseignantID	DepartmentID	Nom	...	Department_Name
1	2	Toto	...	Math
3	2	Tales	...	Math
2	1	Manou	...	Info

Group $\text{Department_Name, Count}_{\text{Enseignant_ID}}$ (Enseignant ∞ Département)

Department_Name	count
Math	2
Info	1

Vue

Pour le SGBD : c'est un requête

Utilisée pour

- Cacher certaines informations à un groupe d'utilisateurs
- Faciliter l'accès à certaines données

create view nom_vue **as** < requête >

Exemple :

create view Info_Non_Confidentielle_Etudiant

as $\Pi_{\text{Etudiant_ID, Nom, Prénom, Email}}$ (Etudiant)

Chap IV - SQL

- Technologies de bases de données **relationnelles** et **transactionnelles** existantes depuis plus de 50 ans
- **SQL** : langage de requête standardisé en 1986
 - **SQL2/SQL92** : standard adopté en 1992
 - **SQL3/SQL99** : extension de SQL2 avec gestion d' "objets", déclencheurs ...
 - **SQL 2003**: auto incrémentation des clés, colonne calculée, prise en compte de XML, ...
 - **SQL 2008 et 2011**: correction de certains défauts et manques (fonctions, types, curseurs) ...
 - **SQL 2016** : gestion de documents JSON, ...
 - **SQL 2019** : tableaux multidimensionnels (*Multi-Dimensional Arrays* - MDA - 2019), ...
 - **SQL 2020+** : SQL/PGQ (*property graph queries* – en cours) ...

Langage SQL

- **Basé sur la logique du 1^{er} ordre**
- Requête d'interrogation : décrit sous forme logique le résultat attendu de la requête
- Peut être décomposé en 4 parties :
 - **Langage de Manipulation de Données (DML)** : interroger et modifier les données de la base
 - **Langage de Définition de Données (DDL)** : définir le schéma de la base de données
 - **Langage de contrôle d'accès aux données (DCL)** : pour définir les privilèges d'accès des utilisateurs
 - **Langage de contrôle des transactions (TCL)** : pour gérer les transactions.

Exécuter des requêtes SQL en ligne

- Pour créer un schéma de bases de données (sous MySQL, PostgreSQL ...) et l'interroger en ligne :
<http://sqlfiddle.com/> ou <https://www.db-fiddle.com/>
- Pour exécuter des requêtes en ligne sur une base exemple :
 - <http://webtic.free.fr/sql/exint/q1.htm>
 - <https://eric.univ-lyon2.fr/~jdarmont/tutoriel-sql/>
 - <https://sqlbolt.com/lesson/introduction>
- Rappels des commandes SQL : <http://sql.sh/>

Structure d'une requête d'interrogation

```
SELECT [DISTINCT] *  
FROM table_1 [varNuplet_1], table_2 [varNuplet_2], ...  
[WHERE prédicat_1  
AND [ou OR] prédicat_2 ...]
```

```
SELECT [DISTINCT] att_1 [AS nom_1], att_2 ...  
FROM table_1 [varNuplet_1], table_2 [varNuplet_2], ...  
[WHERE prédicat_1  
AND [ou OR] prédicat_2 ...]
```

Exemples de requêtes d'interrogation

```
SELECT Nom, Prenom  
FROM Etudiant  
WHERE Ville = ' Paris ' ;
```

```
SELECT Nom, Prenom  
FROM Etudiant  
WHERE Ville = ' Paris '  
AND Nom  
LIKE ' _AR% ' ;
```

```
SELECT Nom, Prenom  
FROM Etudiant  
WHERE Fax IS NULL;
```

```
SELECT Intitule,  
(NbSeances*3) AS NbHeures  
FROM Cours  
WHERE (NbSeances*3)  
BETWEEN 24 AND 27 ;
```

```
SELECT Nom, Prenom  
FROM Enseignant  
WHERE Departement_ID IN  
( ' INFO ', ' MATH ', ' ECO ' )
```

Exemples de résultats d'opérations unaires

Relation *Enseignant*

L..	enseignant_id (...)	departement_id ...	nom (varchar)	prenom ...	grade (...)	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
3	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr
4	4	1	LIMAM	Medhi	ATER			
5	5	5	MyTaylor	IsRich	Vacataire			
6	6	1	RIGAUX	Philippe	PROF			
7	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
8	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

`SELECT * FROM Enseignant WHERE grade = 'MCF'`

L..	enseignant_id...	departement_id...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

`SELECT Nom, Prenom
FROM Enseignant`

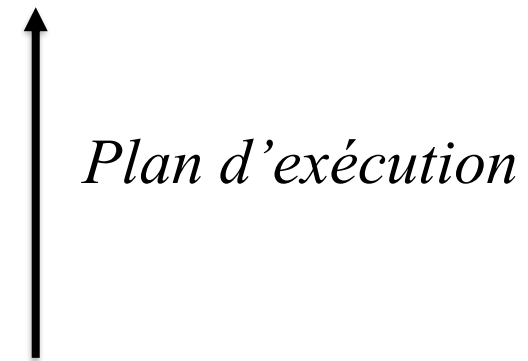
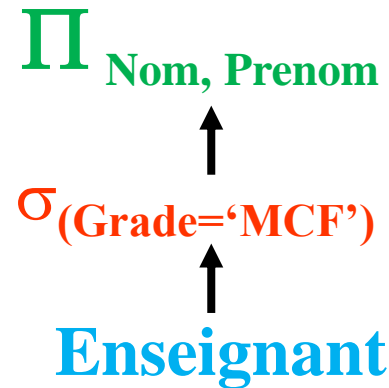
L..	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	NAIJA	Yosr
3	BAHRI	Afef
4	LIMAM	Medhi
5	MyTaylor	IsRich
6	RIGAUX	Philippe
7	CHAKHAR	Salem
8	MURAT	Cécile

`SELECT Nom, Prenom
FROM Enseignant
WHERE Grade = 'MCF'`

Ligne	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	MURAT	Cécile

Lien entre l'algèbre relationnelle et le SQL

```
SELECT Nom, Prenom
FROM Enseignant
WHERE Grade = 'MCF'
```



L..	enseignant_id (...)	departement_id ...	nom (varchar)	prenom ...	grade (...)	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
3	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr
4	4	1	LIMAM	Medhi	ATER			
5	5	5	MyTaylor	IsRich	Vacataire			
6	6	1	RIGAUX	Philippe	PROF			
7	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
8	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

L..	enseignant_id...	departement_id...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

Ligne	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	MURAT	Cécile

Prédicats du WHERE

Prédicats du WHERE de la forme :

exp1 = exp2

exp1 != exp2

exp1 > exp2

exp1 < exp2

exp1 <= exp2

exp1 >= exp2

exp1 **BETWEEN** exp2 **AND** exp3

exp1 **[NOT] LIKE** exp2

exp1 **[NOT] ILIKE** exp2

exp1 **[NOT] IN** (exp2, exp3, ...)

exp1 **IS [NOT] NULL**

exp *op* **ANY** (**SELECT** ...)

exp *op* **ALL** (**SELECT** ...)

avec *op* tel que =, !=, <, >, >=, <=

exp **[NOT] IN** (**SELECT** ...)

exp *op* (**SELECT** ...)

avec *op* tel que =, !=, <, >, >=, <=

SELECT Intitule

FROM Cours

WHERE NbSeances <=

(**SELECT** **AVG**(NbSeances)

FROM Cours);

Exemple avec ANY et ALL

Relation *ReleveNotes*:

nometudiant	intitule	note
Albert	BD	5
Aude	BD	15
Anaïs	BD	18

Noms des étudiants « les moins mauvais » :

```
SELECT NomEtudiant
FROM ReleveNotes
WHERE Note > ANY (SELECT Note FROM ReleveNotes)
```

nometudiant

Aude

Anaïs

Id des étudiants ayant la meilleure note:

```
SELECT NomEtudiant
FROM ReleveNotes
WHERE Note >= ALL (SELECT Note FROM ReleveNotes)
```

nometudiant

Anaïs

EXISTS

Clause EXISTS :

- Retourne VRAI si au moins un nuplet est retourné par la requête
- FAUX si aucun nuplet n'est retourné.
- La valeur NULL n'a aucun effet sur le booléen résultat

```
SELECT Nom, Prenom
FROM Enseignant E
WHERE NOT EXISTS
  ( SELECT *
    FROM Reservation_Salle S
      WHERE S.Enseignant_ID = E.Enseignant_ID
  );
```


Opérateurs d'agregation

Fonctions statistique et de groupage :

COUNT, MIN, MAX, AVG, SUM, EVERY, ...

ORDER BY, GROUP BY

SELECT COUNT(*)

FROM Etudiant ;

SELECT AVG(Capacite), SUM(Capacite)

FROM Salle ;

SELECT Departement_ID, Nom, Prenom

FROM Enseignant

ORDER BY Departement_ID DESC, Nom, Prenom ;

SELECT Departement_Name, COUNT(*)

FROM Reservation_Salle

GROUP BY Departement_Name HAVING COUNT(*) >= 2 ;

Department_Name	count
Math	2

Exemple de tri

numetu	nom	prenom	datenaiss	rue	cp	ville
110	Dupont	Albert	1980-06-01	Rue de Crimée	69001	Lyon
222	West	James	1983-09-03	Studio		Hollywood
300	Martin	Marie	1988-06-05	Rue des Acacias	69130	Ecully
421	Durand	Gaston	1980-11-15	Rue de la Meuse	69008	Lyon
575	Titgoutte	Justine	1985-02-28	Chemin du Château	69630	Chaponost
667	Dupond	Noémie	1987-09-18	Rue de Dôle	69007	Lyon
999	Phantom	Marcel	1960-01-30			

Correction

Saisir une requête SQL

```
SELECT * FROM etudiant ORDER BY ville , Nom DESC
```

Effacer

Exécuter

Résultat

numetu	nom	prenom	datenaiss	rue	cp	ville
999	Phantom	Marcel	1960-01-30			
575	Titgoutte	Justine	1985-02-28	Chemin du Château	69630	Chaponost
300	Martin	Marie	1988-06-05	Rue des Acacias	69130	Ecully
222	West	James	1983-09-03	Studio		Hollywood
421	Durand	Gaston	1980-11-15	Rue de la Meuse	69008	Lyon
110	Dupont	Albert	1980-06-01	Rue de Crimée	69001	Lyon
667	Dupond	Noémie	1987-09-18	Rue de Dôle	69007	Lyon

Exemple de GROUP BY ... HAVING

Modèle conceptuel UML

Modèle logique relationnel

Question

Question 1 : Liste de tous les étudiants

Résultat attendu

numetu	nom	prenom	datenaiss	rue	cp	ville
110	Dupont	Albert	1980-06-01	Rue de Crimée	69001	Lyon
222	West	James	1983-09-03	Studio		Hollywood
300	Martin	Marie	1988-06-05	Rue des Acacias	69130	Ecully
421	Durand	Gaston	1980-11-15	Rue de la Meuse	69008	Lyon
575	Titgoutte	Justine	1985-02-28	Chemin du Château	69630	Chaponost
667	Dupond	Noémie	1987-09-18	Rue de Dôle	69007	Lyon
999	Phantom	Marcel	1960-01-30			

Correction

Saisir une requête SQL

```
select Ville, count(*) from etudiant group by ville having
count(*)>=3
```

Effacer

Exécuter

Résultat

Ville	count(*)
Lyon	3

CASE ... WHEN

CASE ... WHEN: expression conditionnelle générique, similaire aux instructions `if/else` d'autres langages

```
CASE WHEN condition THEN résultat
      [WHEN ...]
      [ELSE résultat]
END [ AS Renommage]
```

Exemple :

a	SELECT a,	a case
---	CASE WHEN a=1 THEN 'un'	---+-----
1	WHEN a=2 THEN 'deux'	1 un
2	ELSE 'autres'	2 deux
3	END	3 autres
	FROM test;	

Exemple de CASE ... WHEN

Personne

nom character varyin	prenom character varyin
Gamotte	Albert
Pabien	Yvon
Computing	Claude
Slatable	Deborah
Suffit	Sam

NbTicketsAchetes

nom character varyin	prenom character varyin	nbtickets bigint
Suffit	Sam	1
Pabien	Yvon	1
Gamotte	Albert	2
Computing	Claude	2

Résultat de la requête

nom character varyin	prenom character varyin	nbtickets bigint
Gamotte	Albert	2
Pabien	Yvon	1
Computing	Claude	2
Slatable	Deborah	0
Suffit	Sam	1

```

SELECT nom, prenom,
       CASE WHEN (nom, prenom) IN (SELECT nom, prenom FROM
NbTicketsAchetes) THEN (SELECT NbTickets FROM
NbTicketsAchetes WHERE nom = Personne.nom AND prenom =
Personne.prenom)
       WHEN (nom, prenom) NOT IN (SELECT nom, prenom FROM
NbTicketsAchetes) THEN 0
       END AS NbTickets
FROM Personne;
    
```

EVERY

Fonctions statistique et de groupage :

EVERY retourne un booléen qui est vrai si toutes les occurrences sont vraies

reservation_id [PK] serial	batiment character varying(10)	numero_salle character varying(10)	enseignement_id integer	master_id integer	enseignant_id integer	date_resa date	heure_debut time without zone	heure_fin time without zone	nombre_heures integer
1	B	022	1	2	1	2012-01-15	08:30:00	11:45:00	3
2	B	020	2	2	1	2011-11-17	13:45:00	17:00:00	3

```
SELECT EVERY(date_resa >= CURRENT_DATE) FROM
Reservation
```

```
every
boolean
```

```
f
```

```
SELECT EVERY(date_resa <= CURRENT_DATE) FROM
Reservation
```

```
every
boolean
```

```
t
```

Jointure en SQL

Jointure : *forme générale*

```
SELECT Nom, Prenom, Nom_Departement  
FROM Enseignant E, Departement D  
WHERE E.Departement_ID = D.Departement_ID ;
```

Jointure interne :

```
SELECT Nom, Prenom, Nom_Departement  
FROM Enseignant INNER JOIN Departement  
ON Enseignant.Departement_ID = Departement.Departement_ID ;
```

Produit cartésien

NSS	Nom	Prénom	Grade	Dept
12345	Manouvrier	Maude	MCF	1
45678	Toto	Titi	Prof	2

La relation Enseignant

Dept_ID	Nom_Dept_
1	Info
2	Math

La relation Departement

NSS	Nom	Prénom	Grade	Dept	Dept_ID	Nom_Dept
12345	Manouvrier	Maude	MCF	1	1	Info
45678	Toto	Titi	Prof	2	1	Info
12345	Manouvrier	Maude	MCF	1	2	Math
45678	Toto	Titi	Prof	2	2	Math

*SELECT * FROM Enseignant, Departement*

Exemple de produit cartésien

La relation *Enseignant* :

Ligne	enseignant_id ...	departement_id ...	nom (varchar)	prenom ...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	4	1	LIMAM	Medhi	ATER			
3	5	5	MyTaylor	IsRich	Vacataire			
4	6	1	RIGAUX	Philippe	PROF			
5	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr
6	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
7	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
8	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr

La relation *Departement* :

Ligne	departement_id ...	nom_departement...
1	1	INFO
2	2	MATHS
3	3	GESTION
4	4	ECO
5	5	LANGUES
6	7	COMM
7	8	OPTION

*SELECT * FROM Enseignant, Departement :*

L..	enseignant_id...	departement_id (...	nom (varchar)	prenom...	grade...	telephone ...	fax ...	email (va...	departement_id...	nom_departement ...
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	1	INFO
2	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	2	MATHS
3	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	3	GESTION
4	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	4	ECO
5	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	5	LANGUES
6	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	7	COMM
7	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	8	OPTION
8	4	1	LIMAM	Medhi	ATER				1	INFO
9	4	1	LIMAM	Medhi	ATER				2	MATHS

Exemple de jointure

La relation *Enseignant* :

Ligne	enseignant_id ...	departement_id ...	nom (varchar)	prenom ...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	4	1	LIMAM	Medhi	ATER			
3	5	5	MyTaylor	IsRich	Vacataire			
4	6	1	RIGAUX	Philippe	PROF			
5	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr
6	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
7	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
8	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr

La relation *Departement* :

Ligne	departement_id ...	nom_departement...
1	1	INFO
2	2	MATHS
3	3	GESTION
4	4	ECO
5	5	LANGUES
6	7	COMM
7	8	OPTION

*SELECT * FROM Enseignant e, Departement d
WHERE e.Departement_ID = d.Departement_ID*

L...	enseignant_id...	departement_id ...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (va...	departement_id...	nom_departement ...
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	1	INFO
2	4	1	LIMAM	Medhi	ATER				1	INFO
3	5	5	MyTaylor	IsRich	Vacata...				5	LANGUES
4	6	1	RIGAUX	Philippe	PROF				1	INFO
5	8	1	MURAT	Cécile	MCF			murat@la...	1	INFO
6	7	1	CHAKHAR	Salem	ATER			chakhar...	1	INFO
7	2	1	NAIJA	Yosr	Moniteur			naija@lm...	1	INFO
8	3	1	BAHRI	Afef	Moniteur			bahri@lm...	1	INFO

Jointure externe et naturelle

Jointure externe :

```
SELECT Nom, Prenom, Nom_Departement  
FROM Enseignant LEFT OUTER JOIN Departement ON  
    Enseignant.Departement_ID = Departement.Departement_ID ;
```

S'il existe des enseignants attaché à aucun département, la valeur de Nom_Departement sera NULL.

En SQL2 : [RIGHT | LEFT | FULL] OUTER JOIN

Jointure naturelle :

```
SELECT Nom, Prenom, Nom_Departement  
FROM Enseignant NATURAL JOIN Departement
```

Produit cartésien:

```
SELECT Nom, Prenom, Nom_Departement  
FROM Enseignant CROSS JOIN Departement
```

Exemple de jointure externe

Personnel

Nom_Employé	Ville
Tom	Marseille
Jerry	Paris
Alex	Limoges
Marthe	Perpignan

Employé

Nom_Employé	Filiale	Salaire
Tom	SUD_EST	10000
Jerry	IDF	25000
Sophie	IDF	15000
Marthe	SUD_OUEST	12000

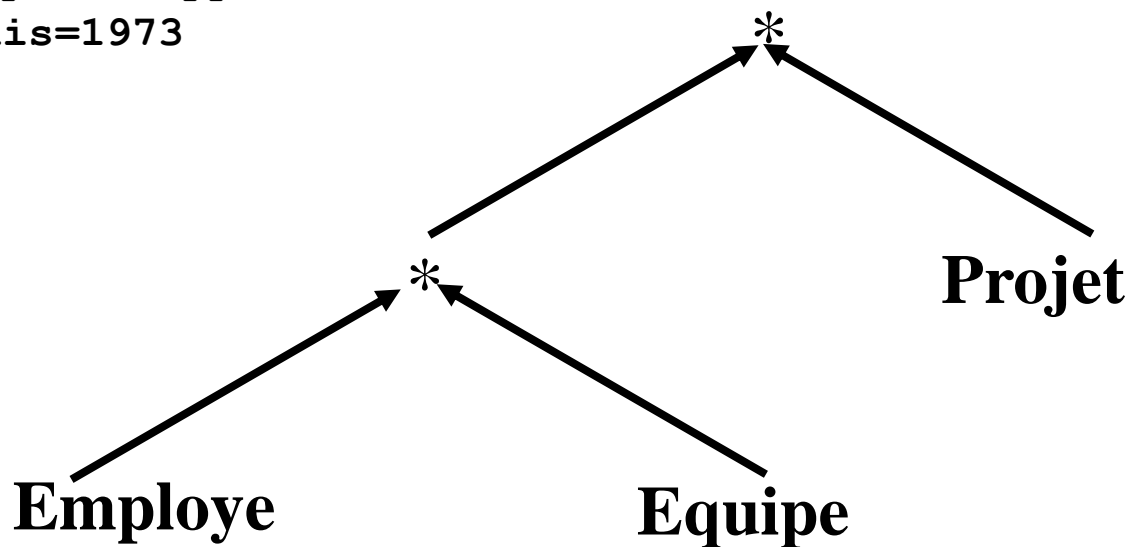
```
SELECT * FROM Personnel
LEFT/RIGHT OUTER JOIN
  Employe ON
Nom_Employe=Employe.Nom
  _Employe
```

Nom_Employé	Ville	Filiale	Salaire
Tom	Marseille	SUD_EST	10000
Jerry	Paris	IDF	25000
Alex	Limoges	NULL	NULL
Marthe	Perpignan	SUD_OUEST	12000

Nom_Employé	Ville	Filiale	Salaire
Tom	Marseille	SUD_EST	10000
Jerry	Paris	IDF	25000
Sophie	NULL	IDF	15000
Marthe	Perpignan	SUD_OUEST	12000

Lien entre l'algèbre relationnelle et le SQL (1/3)

```
SELECT Nom
FROM Employe, Equipe, Projet
WHERE Nom_Projet = 'Sirius'
AND #Projet = #Projet_Equipe
AND #Equipe = #Appartenance
AND DaeNais=1973
```

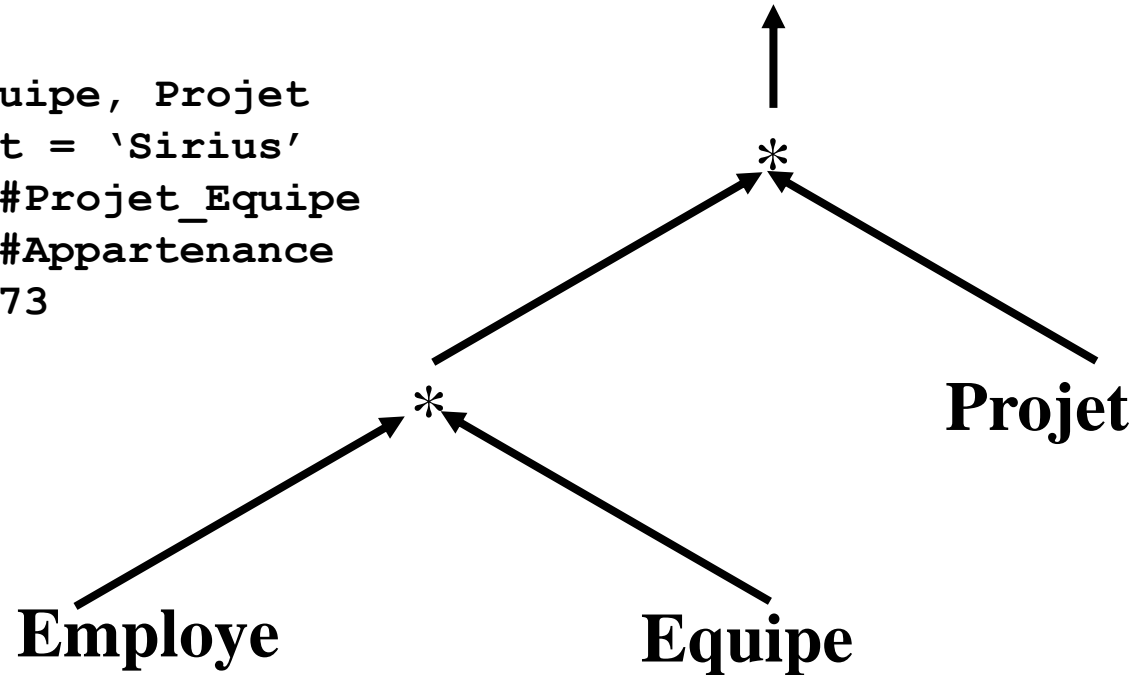


**Analyse du
FROM**

Lien entre l'algèbre relationnelle et le SQL (2/3)

σ (Nom_Projet='Sirius') \wedge (#Projet=#Projet_Equipe) \wedge
 (#Equipe=#Appartenance) \wedge (DateNais=1973)

```
SELECT Nom
FROM Employe, Equipe, Projet
WHERE Nom_Projet = 'Sirius'
AND #Projet = #Projet_Equipe
AND #Equipe = #Appartenance
AND DaeNais=1973
```

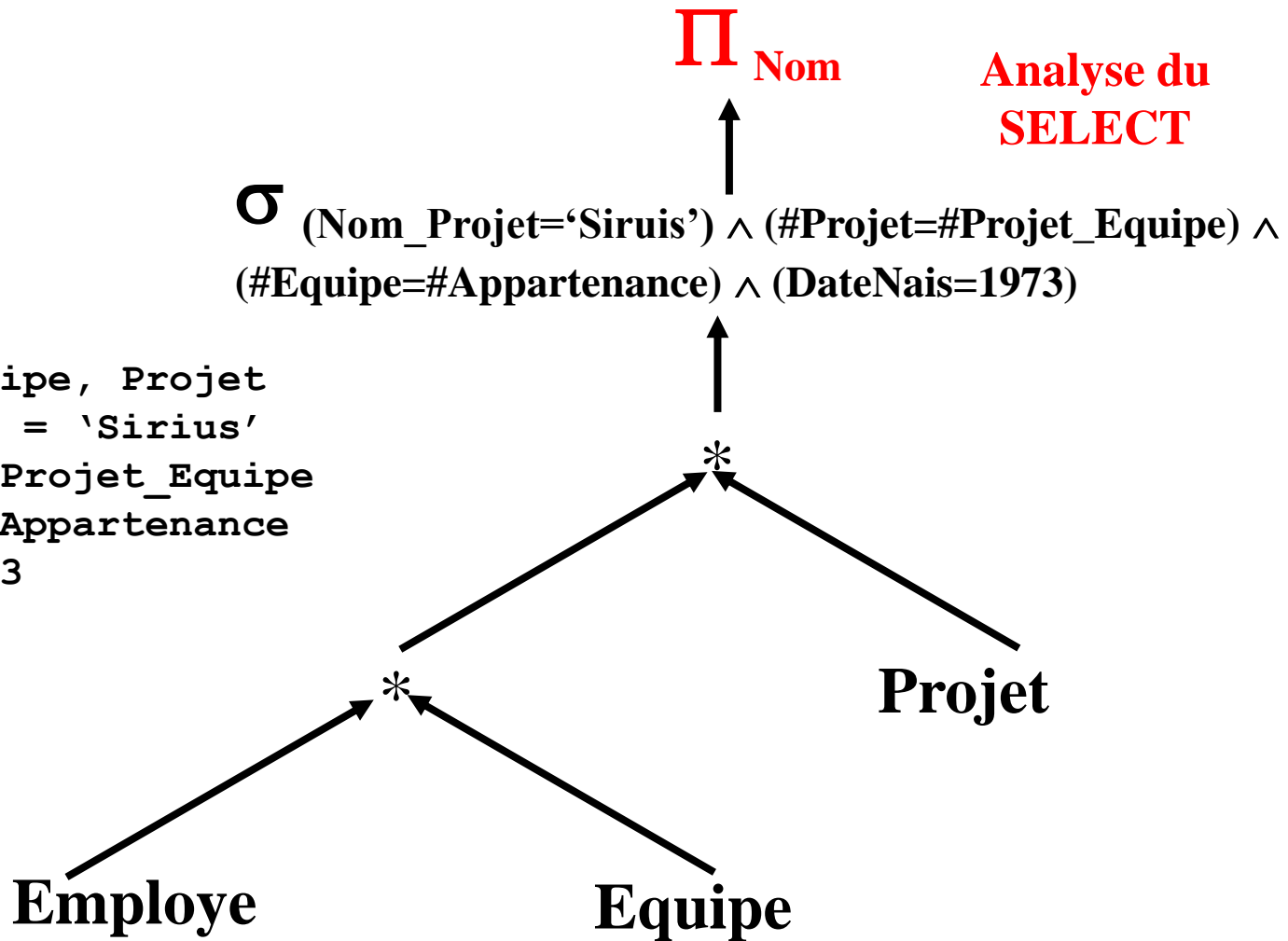


Analyse du
WHERE

Lien entre l'algèbre relationnelle et le SQL (3/3)

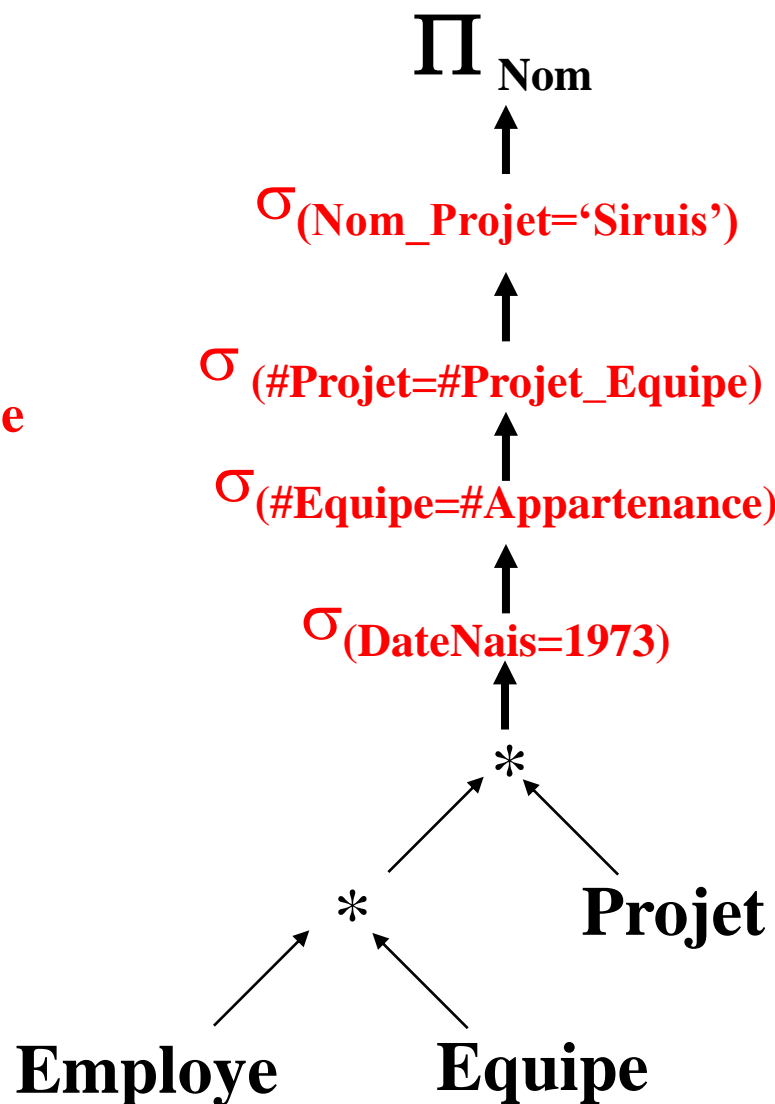
SELECT Nom

```
FROM Employe, Equipe, Projet
WHERE Nom_Projet = 'Sirius'
AND #Projet = #Projet_Equipe
AND #Equipe = #Appartenance
AND DaeNais=1973
```



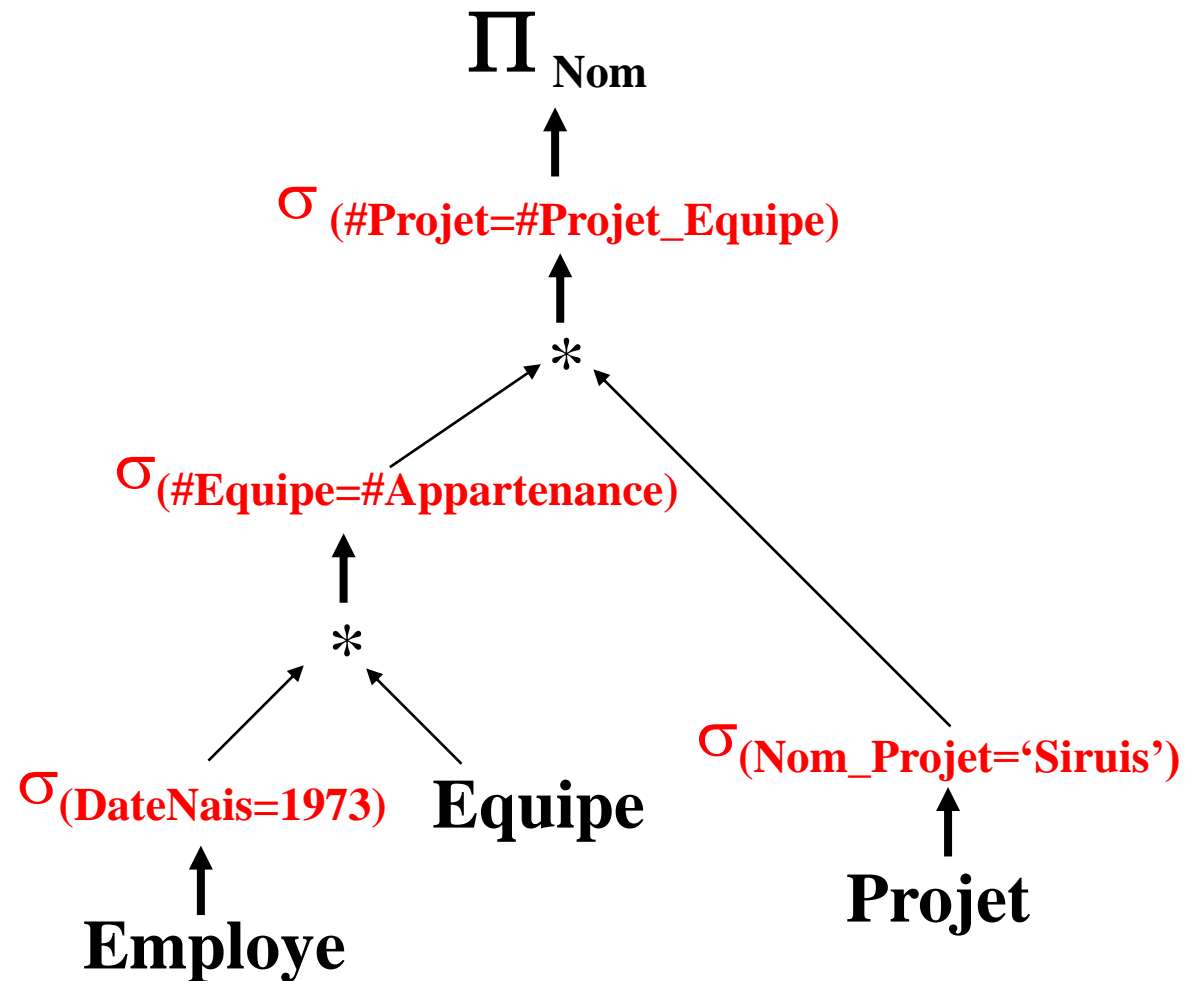
Exemple d'optimisation de requêtes (1/6)

Division des
conjonctions de
sélections

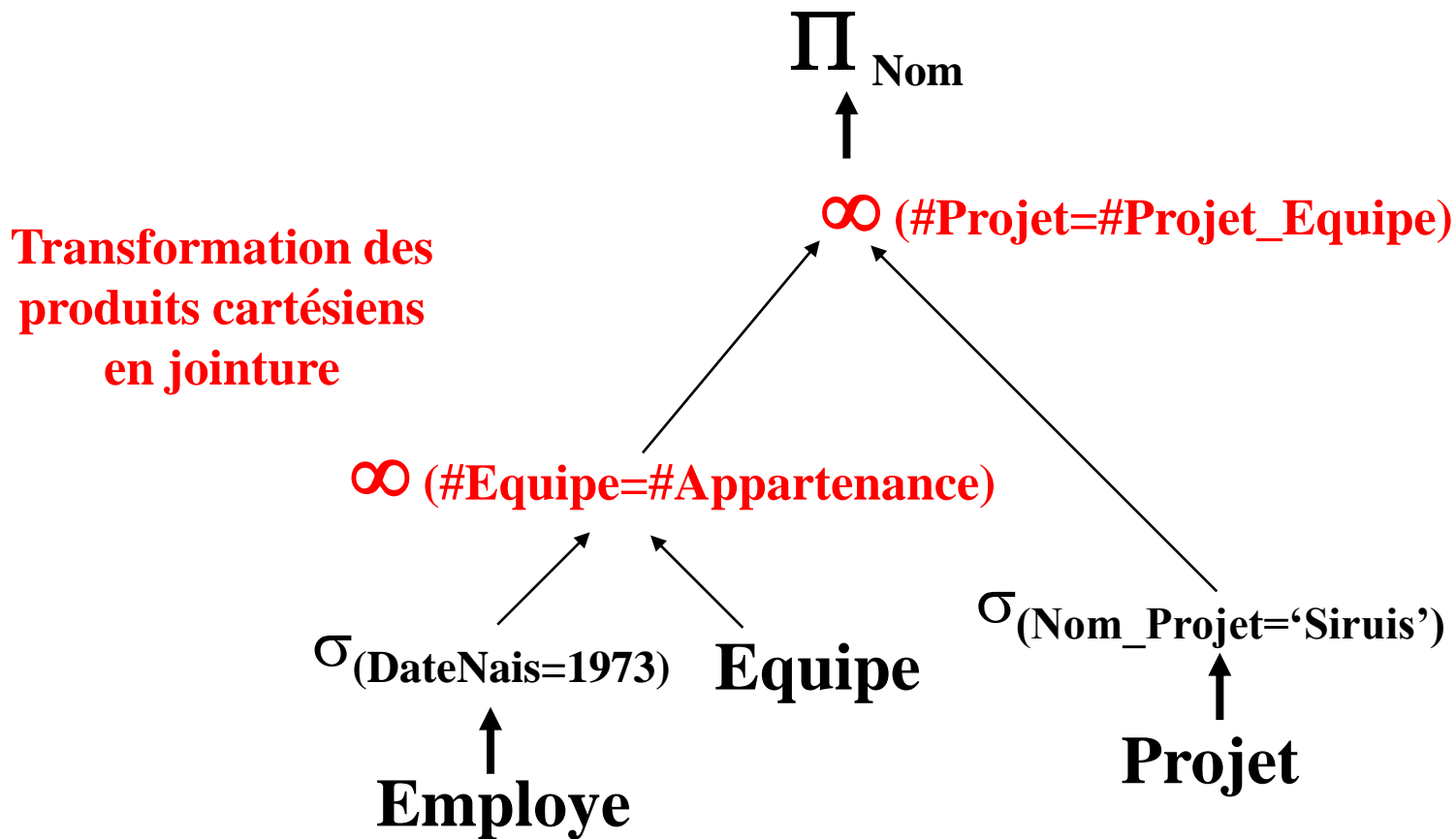


Exemple d'optimisation de requêtes (2/6)

Ré-ordonnancement
des sélections et
application des
sélections les plus
sélectives en premier

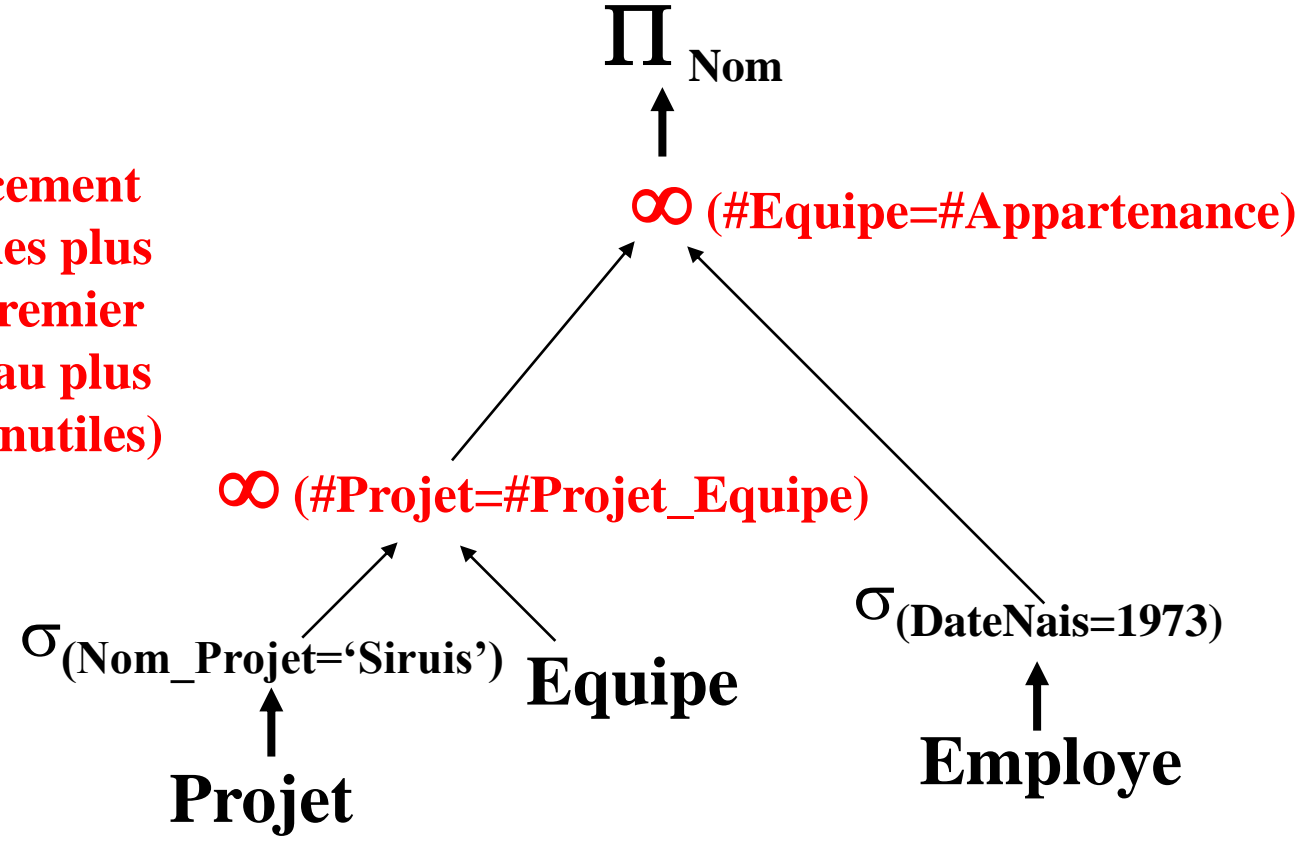


Exemple d'optimisation de requêtes (3/6)

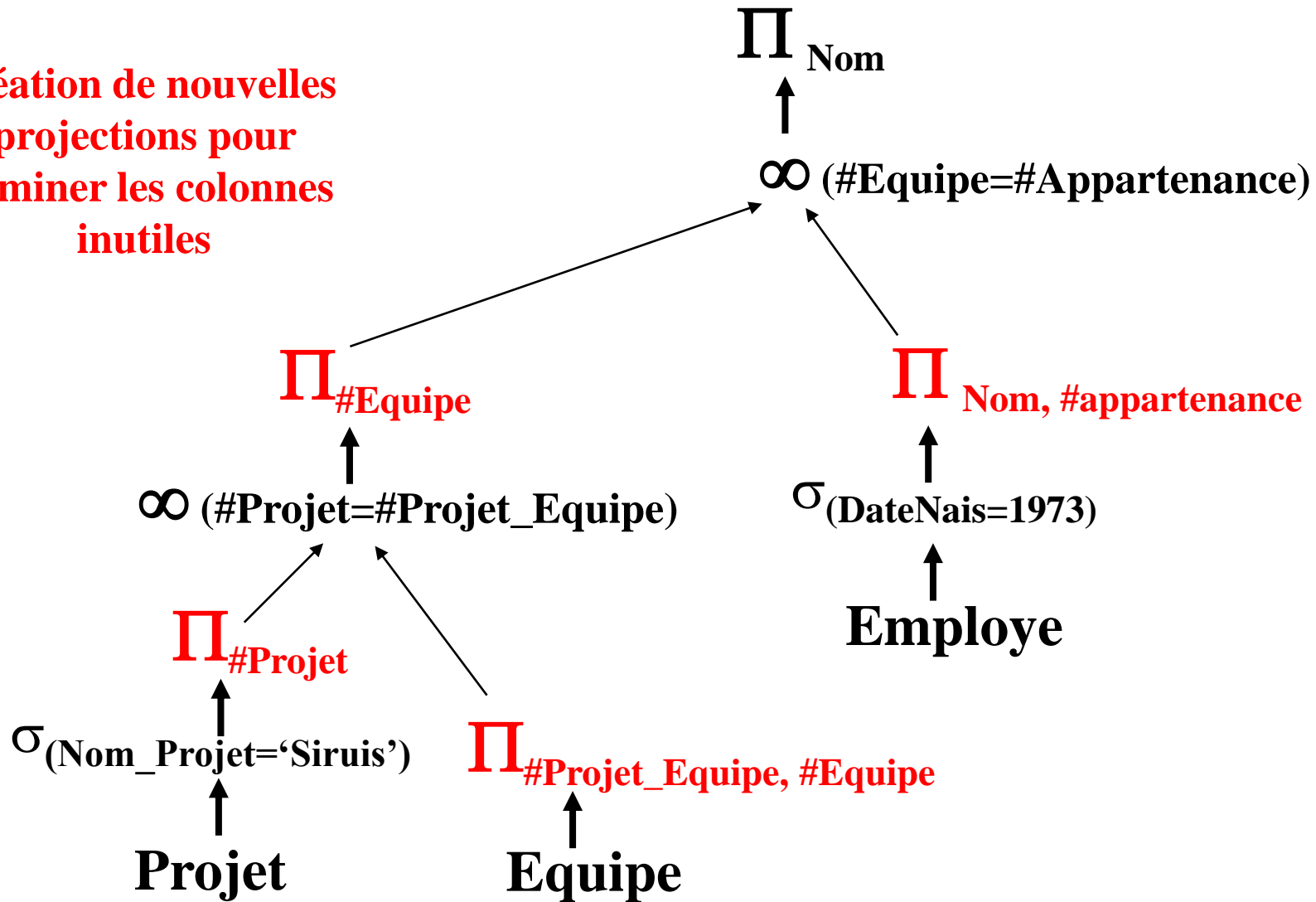


Exemple d'optimisation de requêtes (4/6)

**Ré-ordonnancement
des jointures (les plus
sélectives en premier
pour éliminer au plus
tôt les nuplets inutiles)**



**création de nouvelles
projections pour
éliminer les colonnes
inutiles**



Exemple d'optimisation de requêtes (6/6)

```

SELECT Nom
FROM Employe, Equipe, Projet
WHERE Nom_Projet = 'Sirius'
AND #Projet = #Projet_Equipe
AND #Equipe = #Appartenance
AND DaeNais=1973

```

Plan d'exécution de la requête :

$$\begin{aligned}
 & \Pi_{\text{Nom}} \left[\Pi_{\text{Nom}, \#appartenance} \left[\sigma_{(\text{DateNais}=1973)} (\text{Employe}) \right] \right. \\
 & \qquad \qquad \qquad \infty (\#Equipe=\#Appartenance) \\
 & \qquad \qquad \qquad \left. \Pi_{\#Equipe} \left(\Pi_{\#Projet} \left[\sigma_{(\text{Nom_Projet}='Sirius')} (\text{Projet}) \right] \right. \right. \\
 & \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \infty (\#Projet=\#Projet_Equipe) \\
 & \qquad \qquad \qquad \left. \left. \Pi_{\#Projet_Equipe, \#Equipe} (\text{Equipe}) \right) \right]
 \end{aligned}$$

Union, intersection et différence

Opérateurs ensemblistes :

SELECT Nom, Prenom FROM Etudiant

INTERSECT

SELECT Nom, Prenom FROM Enseignant

SELECT Nom, Prenom FROM Etudiant

UNION [ALL]

SELECT Nom, Prenom FROM Enseignant

ORDER BY Nom, Prenom

SELECT Nom, Prenom FROM Etudiant

EXCEPT

SELECT Nom, Prenom FROM Enseignant



Pas d'union sur une même relation!

Exemple d'union

SELECT nom, prenom
FROM Etudiant

nom character varying (25)	prenom character varying (25)
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal
DEBECE	Gill
DEBECE	Aude

SELECT nom, prenom
FROM Enseignant

nom character varying (25)	prenom character varying (25)
MANOUVRIER	Maude
BELHAJJAME	Khalid
NEGRE	Elsa
MURAT	Cecile
DEBECE	Aude

SELECT nom, prenom
FROM Etudiant
UNION
SELECT nom, prenom
FROM Enseignant

nom character varying (25)	prenom character varying (25)
DEBECE	Gill
MURAT	Cecile
GAMOTTE	Albert
BELHAJJAME	Khalid
ODENT	Jamal
MANOUVRIER	Maude
DEBECE	Aude
NEGRE	Elsa
HIBULAIRE	Pat

Exemple de différence

SELECT nom, prenom
FROM Etudiant

nom character varying (25)	prenom character varying (25)
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal
DEBECE	Gill
DEBECE	Aude

SELECT nom, prenom
FROM Enseignant

nom character varying (25)	prenom character varying (25)
MANOUVRIER	Maude
BELHAJJAME	Khalid
NEGRE	Elsa
MURAT	Cecile
DEBECE	Aude

SELECT nom, prenom
FROM Etudiant
EXCEPT
SELECT nom, prenom
FROM Enseignant

nom character varying (25)	prenom character varying (25)
DEBECE	Gill
GAMOTTE	Albert
HIBULAIRE	Pat
ODENT	Jamal

Attention à l'écriture de vos requêtes (1/2)

Relation TypeEval

typeid	intitule
1	Examen
2	TP
3	Partiel
4	CC

Relation ReleveNotes

releveid	etudiantid	typeid	note
1	4	1	20
2	4	3	20
3	3	1	15
4	2	1	8
5	1	2	18
6	3	2	18

Attention à l'écriture de vos requêtes (2/2)

Query SQL ●

```
1 EXPLAIN ANALYZE (SELECT Note FROM ReleveNotes NATURAL JOIN TypeEval
2 WHERE Intitule = 'Examen' OR Intitule = 'TP' );
```

Planning Time: 0.384 ms

Execution Time: 0.087 ms

Query SQL ●

```
1 EXPLAIN ANALYZE (SELECT Note FROM ReleveNotes NATURAL JOIN TypeEval
2 WHERE Intitule = 'Examen')
3 UNION
4 (SELECT Note FROM ReleveNotes NATURAL JOIN TypeEval
5 WHERE Intitule = 'TP' );
```

Planning Time: 0.538 ms

Execution Time: 0.307 ms

Results

Query #1

note
20
15
8
18
18

Affichage du plan d'exécution sous PostgreSQL (1/3)

Algorithmes de jointure

postgres on postgres@PostgreSQL 9.6

```

1  EXPLAIN ANALYZE SELECT * FROM enfant NATURAL JOIN inscription NATURAL JOIN atelier
2
3

```

Data Output Explain Messages History

QUERY PLAN
1 Hash Join (cost=2.14..3.25 rows=1 width=334) (actual time=0.077..0.07...
2 Hash Cond: (((enfant.nom)::text = (atelier.nom)::text) AND (inscription.a...
3 -> Hash Join (cost=1.07..2.14 rows=3 width=108) (actual time=0.030..0.0...
4 Hash Cond: (enfant.enfantid = inscription.enfantid)
5 -> Seq Scan on enfant (cost=0.00..1.03 rows=3 width=104) (actual time...
6 -> Hash (cost=1.03..1.03 rows=3 width=8) (actual time=0.011..0.011 ro...
7 Buckets: 1024 Batches: 1 Memory Usage: 9kB
8 -> Seq Scan on inscription (cost=0.00..1.03 rows=3 width=8) (actual tim...
9 -> Hash (cost=1.03..1.03 rows=3 width=288) (actual time=0.029..0.029 ...
10 Buckets: 1024 Batches: 1 Memory Usage: 9kB
11 -> Seq Scan on atelier (cost=0.00..1.03 rows=3 width=288) (actual time...
12 Planning time: 0.424 ms
13 Execution time: 0.154 ms

Affichage du plan d'exécution sous PostgreSQL (2/3)

Algorithmes de jointure

postgres on postgres@PostgreSQL 9.6

```

1  EXPLAIN ANALYZE SELECT * FROM enfant ,inscription ,atelier
2  WHERE enfant.enfantid=inscription.enfantid and atelier.atelierid=inscription.atelierid

```

Data Output Explain Messages History

QUERY PLAN
text
1 Hash Join (cost=2.14..3.25 rows=3 width=400) (actual time=0.058..0.06...
2 Hash Cond: (inscription.atelierid = atelier.atelierid)
3 -> Hash Join (cost=1.07..2.14 rows=3 width=112) (actual time=0.030..0....
4 Hash Cond: (enfant.enfantid = inscription.enfantid)
5 -> Seq Scan on enfant (cost=0.00..1.03 rows=3 width=104) (actual time...
6 -> Hash (cost=1.03..1.03 rows=3 width=8) (actual time=0.011..0.011 ro...
7 Buckets: 1024 Batches: 1 Memory Usage: 9kB
8 -> Seq Scan on inscription (cost=0.00..1.03 rows=3 width=8) (actual tim...
9 -> Hash (cost=1.03..1.03 rows=3 width=288) (actual time=0.021..0.021 ...
10 Buckets: 1024 Batches: 1 Memory Usage: 9kB
11 -> Seq Scan on atelier (cost=0.00..1.03 rows=3 width=288) (actual time...
12 Planning time: 0.300 ms
13 Execution time: 0.130 ms

Affichage du plan d'exécution sous PostgreSQL (3/3)

Algorithmes de jointure

postgres on postgres@PostgreSQL 9.6

```

1  EXPLAIN ANALYZE SELECT * FROM enfant inner join inscription on enfant.enfantid=inscription.enfantid
2  inner join atelier on atelier.atelierid=inscription.atelierid

```

Data Output Explain Messages History

QUERY PLAN	
text	
1	Hash Join (cost=2.14..3.25 rows=3 width=400) (actual time=0.061..0.06...
2	Hash Cond: (inscription.atelierid = atelier.atelierid)
3	-> Hash Join (cost=1.07..2.14 rows=3 width=112) (actual time=0.028..0....
4	Hash Cond: (enfant.enfantid = inscription.enfantid)
5	-> Seq Scan on enfant (cost=0.00..1.03 rows=3 width=104) (actual time...
6	-> Hash (cost=1.03..1.03 rows=3 width=8) (actual time=0.009..0.009 ro...
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on inscription (cost=0.00..1.03 rows=3 width=8) (actual tim...
9	-> Hash (cost=1.03..1.03 rows=3 width=288) (actual time=0.023..0.023 ...
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on atelier (cost=0.00..1.03 rows=3 width=288) (actual time...
12	Planning time: 0.343 ms
13	Execution time: 0.144 ms

Activer Windows

Exemples de requêtes « équivalentes »

```
select titre
from Film f, Rôle r, Artiste a
where a.nom = 'Stewart' and a.prénom='James'
and f.id_film = r.id_film
and r.id_acteur = a.idArtiste
and f.annee = 1958
```

```
select titre
from Film f, Rôle r
where f.id_film = r.id_film
and f.annee = 1958
and exists (select 'x'
            from Artiste a
            where nom='Stewart'
            and prénom='James'
            and r.id_acteur = a.id_acteur)
```

```
select titre
from Film f, Rôle r
where f.id_film = r.id_film
and f.annee = 1958
and r.id_acteur in (select id_acteur
                   from Artiste
                   where nom='Stewart'
                   and prénom='James')
```

```
select titre from Film
where annee = 1958
and id_film in
    (select id_film from Rôle
     where id_acteur in
         (select id_acteur
          from Artiste
          where nom='Stewart'
          and prénom='James'))
```

Pour aider l'optimiseur

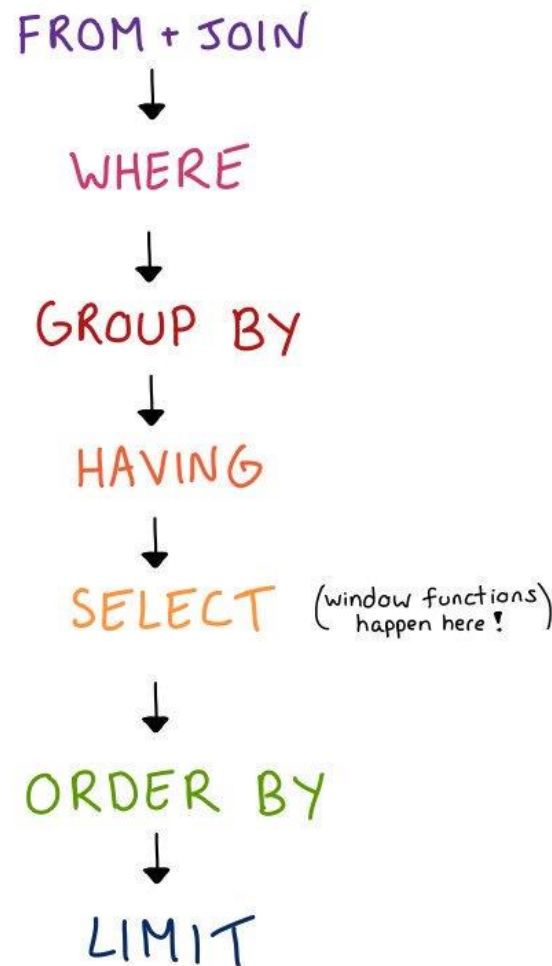
- Influence du choix de la syntaxe des requêtes SQL sur les possibilités d'optimisation laissées au SGBD
- **Préférer les requêtes SQL dites "à plat", i.e. ayant un seul bloc SELECT-FROM-WHERE, et laisser le SGBD l'optimiser.**
- Possibilité de traduire une requête avec imbrication en une requête équivalente sans imbrication (un seul bloc)
 1. Quand l'équivalence existe.
 2. Si le SGBD est capable de déceler ce type d'équivalence.
- Pour les requêtes SQL à plusieurs blocs : Etudier le plan d'exécution pour vérifier que les index sont correctement utilisés.

Hors programme en L3 : mais attention à ne pas faire d'union, ni de jointure inutiles !

JULIA EVANS
@bork

SQL queries run
in this order

Ordre d'exécution des opérateurs d'une requête



Division en SQL (1/3)

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID, ISBN, DateEmprunt, EtudiantID)

Etudiant(EtudiantID, Nom, Prenom)

« Quels livres ont été empruntés par tous les étudiants? »

$$\{ \text{t.Titre / Livre}(t) \wedge \neg [\exists u \text{ Etudiant}(u) \wedge \neg (\exists v \text{ Emprunt}(v) \wedge (v.\text{Etudiant_ID}=u.\text{Etudiant_ID}) \wedge (v.\text{ISBN}=t.\text{ISBN}))] \}$$

Il n'y a pas de mot-clé "quel que soit" en SQL2

```
SELECT t.Titre FROM Livre t WHERE NOT EXISTS
  ( SELECT * FROM Etudiant u WHERE NOT EXISTS
    ( SELECT * FROM Emprunt v
      WHERE u.EtudiantID=v.EtudiantID AND
            v.ISBN=t.ISBN
    )
  )
);
```

Division en SQL (2/3)

*Il n'y a pas de mot-clé
"quel que soit" en SQL2*

Division :

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID, ISBN, DateEmprunt, EtudiantID)

Etudiant(EtudiantID, Nom, Prenom)

« **Quels livres ont été empruntés par tous les étudiants?** »

Le résultat de la requête va contenir les nuplets t de *Livre* tels que :

Quel que soit un étudiant u (i.e. un nuplet u dans Etudiant)

Il existe un nuplet v dans Emprunt indiquant que le livre t a été emprunté par u

Dit sans quel que soit :

Le résultat de la requête va contenir les nuplets t de *Livre* tels que :

Il n'est pas possible de trouver un étudiant u (i.e. un nuplet u dans Etudiant)

Pour lequel il n'existe pas de nuplet v indiquant t a été emprunté par u

```

SELECT t.Titre FROM Livre t WHERE NOT EXISTS
  (SELECT * FROM Etudiant u WHERE NOT EXISTS
    (SELECT * FROM Emprunt v
      WHERE u.EtudiantID=v.EtudiantID AND
        v.ISBN=t.ISBN
    )
  )
);

```

Division en SQL (3/4)

Relation *Match* :

MID	DateMatch	Heure	Lieu
1	"21/01/17"	17h	Boulogne
2	"22/01/17"	15h	Asnières

Relation *Convocation* :

MID	JID
1	2
2	2

Relation *Joueur* :

JID	Nom	Prenom	AnneeNaissance
1	Comptuting	Claude	2005
2	Debecce	Gilles	2008

Noms et prénoms des joueurs convoqués à tous les matchs :

```

SELECT Nom, Prenom FROM Joueur j WHERE NOT EXISTS
  ( SELECT * FROM Match m WHERE NOT EXISTS
    ( SELECT * FROM Convocation c
      WHERE j.JID=c.JID AND c.MID=m.MID
    )
  )
);

```

Division en SQL (4/4)

La relation *Enseignement* :

L.	enseignement_id ...	departement_id ...	intitule (varchar)	description (varchar)
1	①	①	Bases de Données	Niveau Licence : Modélisation E/A et UML, Modèle relationnel, Algèbre Relation...
2	②	①	Mise à Niveau Informatique	Pour les étudiants de GMI entrant directement en IUP2: Architecture, Algorith...
3	③	①	Mise à Niveau Bases de Données	Pour les étudiants de DESS ID ou DEA.127 - Programme Licence et Maîtrise en ...
4	①	⑤	Anglais	

La relation
Inscription :

Ligne	etudiant_id (int4)	enseignement_id (int4)	departement_id (int4)	date_inscription (date)
1	①	①	①	2004-02-25
2	①	②	①	2004-07-22
3	3	2	1	2004-07-22
4	5	2	1	2004-07-22
5	4	2	1	2004-07-22
6	①	③	①	2004-07-22
7	①	①	⑤	2004-07-22
8	2	1	5	2004-07-22

```

SELECT t.etudiant_id FROM Inscription t WHERE NOT EXISTS
( SELECT * FROM Enseignement u WHERE NOT EXISTS
( SELECT * FROM Inscription v
WHERE u.enseignement_id=v. enseignement_id
AND u.departement_id=v. departement_id
AND t.etudiant_id=v.etudiant_id) );
    
```

Ligne	etudiant_id (int4)
1	1

Insertion, suppression et mise à jour

■ Insertion

```
INSERT INTO table(col1, col2, ... coln)
VALUES (val1, val2, ... valn) [RETURNING att] ;
```

```
INSERT INTO table(col1, col2, ... coln)
SELECT ...
```

■ Suppression

```
DELETE FROM table
WHERE prédicat [RETURNING att] ;
```

■ Mise à jour

```
UPDATE table
SET col1 = exp1, col2 = exp2
WHERE prédicat [RETURNING att] ;
```

■ Transactions : COMMIT, ROLLBACK [TO], SAVE POINT

Exemples d'insertion, de suppression et de mise à jour

```
INSERT INTO Personne (Nom, Prenom, dateNaissance, sexe)  
  VALUES ('Onette', 'Camille', '1973-05-04', 'M')  
  RETURNING personneId; -- pour récupérer la valeur  
                           de la clé artificielle
```

```
UPDATE Produits SET prix = prix * 1.10  
  WHERE prix <= 99.99  
  RETURNING nom, prix AS nouveau-prix;
```

```
DELETE FROM Produits ;
```

```
DELETE FROM Produits WHERE prix < 0.99;
```

```
DELETE FROM Produits WHERE prix < 0.99 RETURNING nom ;
```

Mettre les insertions, suppressions et mises à jour au sien d'une transaction

Exemple :

-- 2 requêtes de mises à jour indépendantes

```
UPDATE compte SET solde = solde - 100 where compteid = x;
```

```
UPDATE compte SET solde = solde + 100 where compteid = y;
```

-- une transaction composée de 2 requêtes de mises à jour

```
BEGIN;
```

```
    UPDATE compte SET solde = solde - 100 where compteid = x;
```

```
    UPDATE compte SET solde = solde + 100 where compteid = y;
```

```
COMMIT;
```

Clause WITH

- Pour créer des *Common Table Expressions* ou CTE ~ tables temporaires qui n'existent que pour une requête
- Ordre auxiliaire dans une clause WITH : un SELECT, INSERT, UPDATE, ou DELETE
- Clause WITH elle-même attachée à un ordre primaire SELECT, INSERT, UPDATE, ou DELETE

```
WITH query_name
(column_name1, ...) AS
  (SELECT ...)
```

```
SELECT ...
```

```
WITH query_name1 AS (
  SELECT ...
)
, query_name2 AS (
  SELECT ...
  FROM query_name1
  ...
)
SELECT ...
```


Clause WITH + SELECT

```
WITH ventes_regionales AS (  
    SELECT region, SUM(montant) AS ventes_totales  
    FROM commandes  
    GROUP BY region  
) , meilleures_regions AS (  
    SELECT region  
    FROM ventes_regionales  
    WHERE ventes_totales > (SELECT SUM(ventes_totales)/10  
                            FROM ventes_regionales)  
)
```

```
SELECT region,  
        produit,  
        SUM(quantite) AS unites_produit,  
        SUM(montant) AS ventes_produit  
FROM commandes  
WHERE region IN (SELECT region FROM meilleures_regions)  
GROUP BY region, produit;
```

Clause WITH + INSERT ou UPDATE

Exemples :

```
WITH lignes_deplacees AS (  
    DELETE FROM produits -- suppressions des nuplets  
    WHERE  
        "date" >= '2010-10-01' AND  
        "date" < '2010-11-01'  
    RETURNING * -- en retournant les nuplets supprimés  
)  
INSERT INTO log_produits  
    SELECT * FROM lignes_deplacees;  
-- clause WITH attachée à l'INSERT  
-- la requête déplace les nuplets de produits vers log_produits  
  
WITH t AS (  
    UPDATE produits SET prix = prix * 1.05  
    RETURNING *  
)  
SELECT * FROM t; -- affiche les données mises à jour
```

Clause **WITH** pour créer des variables

- Possibilité de représenter des variables en SQL par des relations ou des valeurs dans des relations
- **Création de nouvelle variable avec la clause **WITH** : attention variable immuable !!**

Listing 1: Creating and updating variables.

```
x = 1; x += 1; print(x) # x is mutable
```



```
WITH x(x) AS (          -- x is immutable
  VALUES (1)
), x2(x) AS (          -- create new variable
  SELECT x + 1 FROM x
) SELECT x FROM x2; -- print result
```



x
2

Clause WITH pour définir des fonctions

- Possibilité d'intégrer des fonctions locales

Listing 2: Calling functions.

```
def kelvin_to_celsius(temperature):
    return [k - 273.15 for k in temperature]
def kelvin_to_fahrenheit(temperature):
    return [k * 9 / 5 - 459.67 for k in temperature]
temperature = [300, 170]
print(kelvin_to_fahrenheit(temperature))
```



```
WITH params(temperature) AS (
    VALUES (300::float), (170)
), kelvin_to_celsius(temperature) AS ( -- inline
    SELECT temperature - 273.15 FROM params
), kelvin_to_fahrenheit(temperature) AS ( -- inline
    SELECT temperature * 9 / 5 - 459.67 FROM params
) SELECT * FROM kelvin_to_fahrenheit;
```



temperature

80.329999999999998

-153.670000000000002

■ Simulation d'une structure conditionnelle

```

from random import random

def kelvin_to_celsius(temperature):
    return [k - 273.15 for k in temperature]

def kelvin_to_fahrenheit(temperature):
    return [k * 9 / 5 - 459.67 for k in temperature]

temperatures = [300, 170]

if random() > 0.5:
    A = kelvin_to_celsius(temperatures)
else:
    A = kelvin_to_fahrenheit(temperatures)
    print(A)

```

* PostgreSQL fournit une fonction `random()` qui renvoie un entier entre 0 et 1 : `SELECT random();`

```

WITH condition AS ( -- random condition
    VALUES (random() > 0.5)
)

```

`SELECT * FROM condition` renvoie true ou false

```

WITH params(temperature) AS (
    VALUES (300::float), (170)
), kelvin_to_celsius(temperature) AS ( -- inline
    SELECT temperature - 273.15 FROM params
), kelvin_to_fahrenheit(temperature) AS ( -- inline
    SELECT temperature * 9 / 5 - 459.67 FROM params
), -- branching on a random value starts now
condition AS ( -- random condition
    VALUES (random() > 0.5)
), A(A) AS ( -- only one relation is selected
    SELECT * FROM kelvin_to_celsius WHERE
    (SELECT * FROM condition)
    UNION ALL
    SELECT * FROM kelvin_to_fahrenheit WHERE
    NOT (SELECT * FROM condition)
) SELECT * FROM A

```

Clause WITH RECURSIVE

- Pour faire des itérations

```
WITH RECURSIVE R AS (
    requeteDeBase
    UNION ALL
    requeteFaisantReferenceàR
)
SELECT ... FROM R
```

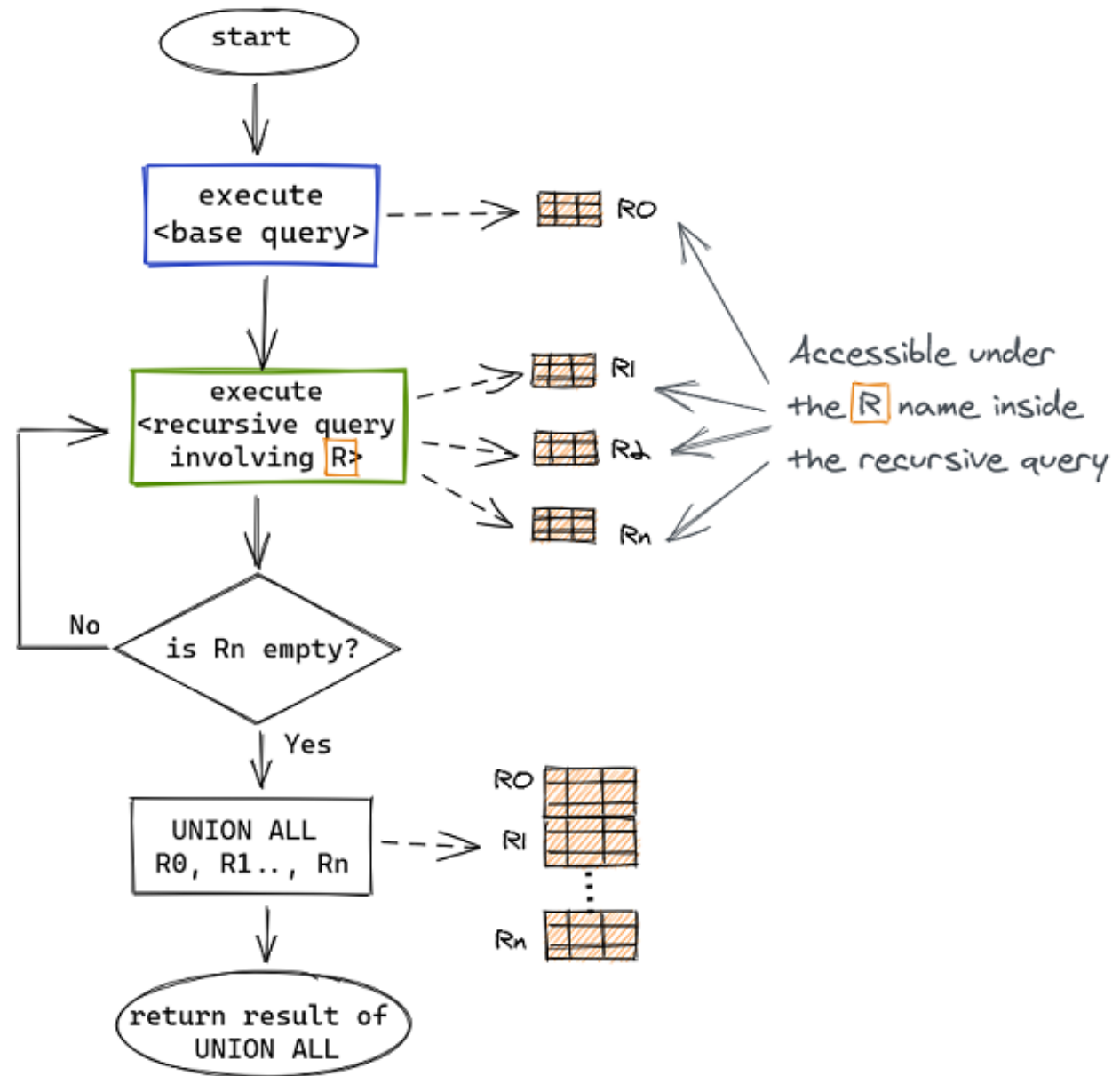


La requête faisant référence à R doit finir par ne retourner aucun nuplet pour arrêter la boucle !

Exemple : requête, qui calcule la somme des nombres de 1 à 100

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
    UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

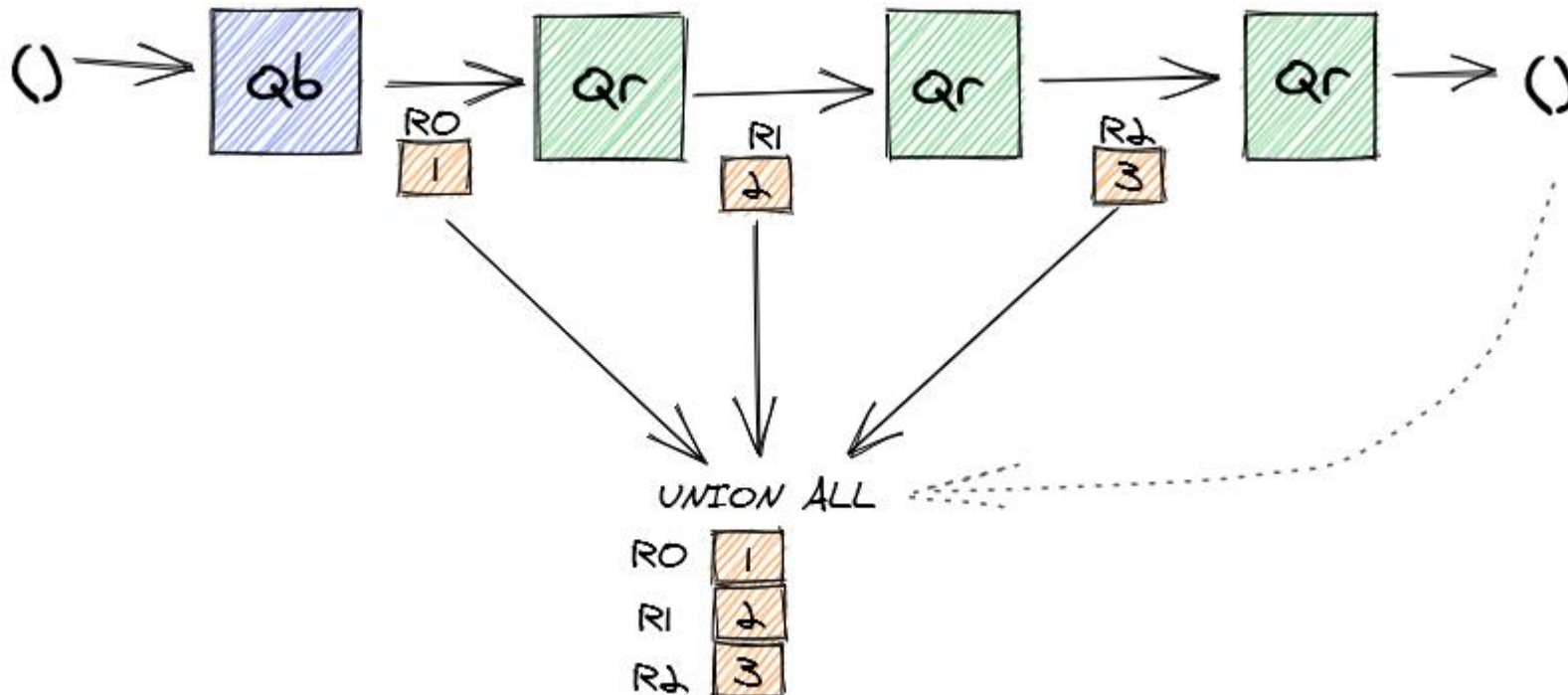
Fonctionnement de la clause WITH RECURSIVE



Exemple simple de clause WITH RECURSIVE

```

WITH countUp AS (
    SELECT 1 as n
    UNION ALL
    SELECT n+1 FROM countUp WHERE n < 3)
SELECT * FROM countUp
    
```



Exemple de clause WITH RECURSIVE

Relation ParentEnfant :

nomparent	prenomparent	nomenfant	prenomenfant
Onette	Marion	Kacontact	Jessy
Teste	Otto	Kacontact	Jessy
Onette	Camille	Onette	Marion

Requête qui affiche les ancêtres de *Jessy Kacontact* :

```
WITH RECURSIVE ancetre AS
( SELECT nomParent, prenomParent FROM ParentEnfant
  WHERE nomEnfant = 'Kacontact' AND prenomEnfant = 'Jessy'

  UNION ALL

  SELECT DISTINCT e.nomParent, e.prenomParent
    FROM ParentEnfant e, ancetre a
    WHERE e.nomEnfant = a.nomParent and e.prenomEnfant = a.prenomParent
)

SELECT * FROM ancetre;
```

nomparent	prenomparent
Onette	Marion
Teste	Otto
Onette	Camille

Utilisation de WITH RECURSIVE pour itérer

Listing 4: Loop without recursive reference in a subquery.

```
# Taylor series for approximating e^x
from math import factorial
x = [1.5, 4.0]
e_pow_x = [1.0] * len(x) # make list of ones
for i in range(1, 20):
    for j in range(len(x)):
        e_pow_x[j] += x[j] ** i / factorial(i)
print(e_pow_x) # e^x = 1 + x^1/1! + x^2/2! + ...
```



```
WITH RECURSIVE exp(x, i, e_pow_x) AS (
    VALUES (1.5::float, 1, 1.0::float),
            (4.0, 1, 1.0)
    UNION ALL
    SELECT x, i + 1, e_pow_x + POWER(x, i) / !!i
    FROM exp WHERE i < 20 -- no recursive reference
) SELECT e_pow_x FROM exp WHERE i = 20;
```



e_pow_x
4.4816890703380645
54.59814947621461

x	i	e_pow_x
1.5	1	1
4	1	1
1.5	2	2.5
4	2	5
1.5	3	3.625
4	3	13
1.5	4	4.1875
4	4	23.666666666666664
1.5	5	4.3984375
4	5	34.333333333333333
1.5	6	4.46171875
4	6	42.866666666666666
1.5	7	4.4775390625
4	7	48.555555555555555
1.5	8	4.480929129464286
4	8	51.8063492063492
1.5	9	4.48156476702009
4	9	53.43174603174603
1.5	10	4.481670706612723
4	10	54.15414462081129

Boucle avec référence récursive

Listing 5: Loop with recursive references in subqueries.

```
# Newton's method: compute x so that
# f(x) = x^2 + cos(x) + 2 * sin(x) is minimized
from math import sin, cos
x = 0
while True:
    fD = 2 * x - sin(x) + 2 * cos(x) # f'
    if abs(fD) < 1e-6:
        break
    fD2 = 2 - cos(x) - 2 * sin(x)    # f''
    x -= fD / fD2                    # update x
print(x)                             # -0.975012
```



```
WITH RECURSIVE x(x, i) AS (
    VALUES (0::float, 0::int)
UNION ALL
    SELECT x, i + 1 FROM (
        WITH x(x, i) AS ( -- recursive reference of x
            SELECT * FROM x -- workaround for PostgreSQL
        ), fD(fD) AS ( -- f'
            SELECT 2 * x - SIN(x) + 2 * COS(x) FROM x
        ), fD2(fD2) AS ( -- f''
            SELECT 2 - COS(x) - 2 * SIN(x) FROM x
        ) SELECT x - fD / fD2, fD, i FROM x, fD, fD2
    ) AS x_new(x, fD, i) WHERE ABS(fD) > 1e-6
) SELECT x FROM x WHERE i = (SELECT MAX(i) FROM x);
```



x	i
0	0
-2	1
-1.073616165220054	2
-0.9777881527674838	3
-0.9750147005864241	4
-0.9750122793245395	5

Activer Windows
Accédez aux paramètres pou

Création de tables

```
CREATE TABLE table (col1 type1 [NOT NULL] ,  
                    col2 type2 [NOT NULL] ...  
                    )
```

Contraintes :

CONSTRAINT *nom_contrainte*

PRIMARY KEY (liste attributs clé primaire)

| **NOT NULL** *immédiatement après la déclaration de l'attribut*

| **CHECK** (condition) *après la déclaration de l'attribut*

/ **UNIQUE** *après la déclaration de l'attribut*

| **FOREIGN KEY** (clé étrangère)

REFERENCES nom_table (liste-colonne)

```
CREATE TABLE table
```

```
AS SELECT ...
```

Chap. VI - SQL Exemples de création de table (1/3)

```
CREATE TABLE Enseignant
```

```
(
```

```
  Enseignant_ID          integer,
```

```
  Departement_ID        integer NOT NULL,
```

```
  Nom                    varchar(25) NOT NULL,
```

```
  Prenom                 varchar(25) NOT NULL,
```

```
  Grade                  varchar(25)
```

```
  CONSTRAINT CK_Enseignant_Grade
```

```
  CHECK (Grade IN ('Vacataire', 'Moniteur', 'ATER', 'MCF', 'PROF'))),
```

```
  Telephone              varchar(10) DEFAULT NULL,
```

```
  Fax                    varchar(10) DEFAULT NULL,
```

```
  Email                  varchar(100) DEFAULT NULL,
```

```
  CONSTRAINT PK_Enseignant PRIMARY KEY (Enseignant_ID),
```

```
  CONSTRAINT "FK_Enseignant_Departement_ID"
```

```
  FOREIGN KEY (Departement_ID)
```

```
  REFERENCES Departement (Departement_ID)
```

```
  ON UPDATE RESTRICT ON DELETE RESTRICT
```

```
);
```

*Contrainte
de domaine*

*Définition de la
clé primaire*

*Définition d'une clé
étrangère*

CREATE TABLE Reservation

```
(
Reservation_ID    integer,
Batiment          varchar(1) NOT NULL,
Numero_Salle     varchar(10) NOT NULL,
Enseignement_ID  integer NOT NULL,
Departement_ID   integer NOT NULL,
Enseignant_ID    integer NOT NULL,
Date_Resa        date NOT NULL DEFAULT CURRENT_DATE,
Heure_Debut      time NOT NULL DEFAULT CURRENT_TIME,
Heure_Fin        time NOT NULL DEFAULT '23:00:00',
Nombre_Heures    integer NOT NULL,
CONSTRAINT PK_Reservation PRIMARY KEY (Reservation_ID),
CONSTRAINT "FK_Reservation_Salle" FOREIGN KEY (Batiment,Numero_Salle) REFERENCES
    Salle (Batiment,Numero_Salle) ON UPDATE RESTRICT ON DELETE RESTRICT,
CONSTRAINT "FK_Reservation_Enseignement" FOREIGN KEY (Enseignement_ID,Departement_ID)
    REFERENCES Enseignement (Enseignement_ID,Departement_ID) ON UPDATE RESTRICT ON
    DELETE RESTRICT,
CONSTRAINT "FK_Reservation_Enseignant" FOREIGN KEY (Enseignant_ID) REFERENCES
    Enseignant (Enseignant_ID) ON UPDATE RESTRICT ON DELETE RESTRICT,
CONSTRAINT CK_Reservation_Nombre_Heures CHECK (Nombre_Heures >=1),
CONSTRAINT CK_Reservation_HeureDebFin
    CHECK (Heure_Debut < Heure_Fin)
);
```

Exemples de création de table (2/3)

Exemples de création de table (3/3)

SERIAL et MATCH SIMPLE/PARTIAL/FULL:

```
CREATE TABLE inscription(
  inscription_id SERIAL NOT NULL,
  etudiant_id integer,
  enseignement_id integer,
  master_id integer,
  CONSTRAINT pk_inscription PRIMARY KEY (inscription_id ),
  CONSTRAINT "FK_Inscription_Enseignement"
  FOREIGN KEY (enseignement_id, master_id)
  REFERENCES enseignement (enseignement_id, master_id)
  MATCH SIMPLE
  ON UPDATE RESTRICT ON DELETE RESTRICT,
  CONSTRAINT "FK_Inscription_Etudiant" FOREIGN KEY (etudiant_id)
  REFERENCES etudiant (etudiant_id) MATCH SIMPLE
  ON UPDATE RESTRICT ON DELETE RESTRICT,
  CONSTRAINT "UN_Inscription" UNIQUE (etudiant_id ,
  enseignement_id , master_id )
)
```

MATCH SIMPLE / PARTIAL / FULL

Dans tous les cas : si toutes les colonnes de FK sont renseignées, la contrainte s'applique

INSERT INTO INSCRIPTION VALUES (6, 2, 3, 4) KO

si la clé (*enseignement_id*, *master_id*)=(3, 4) n'est pas présente dans la table *enseignement*

- **MATCH SIMPLE (par défaut) :**

si une colonne au moins possède un marqueur NULL, la contrainte ne s'applique pas

INSERT INTO INSCRIPTION VALUES (3, 2, NULL, NULL) OK

INSERT INTO INSCRIPTION VALUES (4, 2, 3, NULL) OK

même si pas d'*enseignement_id* 3 dans la table *enseignement*

INSERT INTO INSCRIPTION VALUES (5, 2, NULL, 1) OK

même si pas de *master-id* 1 dans la table *master*

- **MATCH PARTIAL : La contrainte s'applique pour toutes les colonnes renseignées**

INSERT INTO INSCRIPTION VALUES (4, 2, 3, NULL) KO

car pas d'*enseignement_id* 3 dans la table *enseignement*

- **MATCH FULL : la contrainte s'applique toujours sauf si toutes les colonnes sont NULL**

INSERT INTO INSCRIPTION VALUES (3, 2, NULL, NULL) OK

INSERT INTO INSCRIPTION VALUES (4, 2, 1, NULL) KO

même si *enseignement_id* 1 existe dans la table *enseignement*

Autre exemple sur : <https://sqlpro.developpez.com/cours/sqlaz/ddl/?page=partie2#L7.3.1>

UNDER

Héritage de table (norme SQL99) :

CREATE TABLE **Personne**

(**PID Integer,**

Nom **varchar(25) NOT NULL,**

Prenom **varchar(25) NOT NULL,**

CONSTRAINT PK_Personne PRIMARY KEY (PID),

)

Tables héritées de Personne :

CREATE TABLE **Etudiant UNDER** **Personne**

(**NumCarte varchar(25) NOT NULL**)

CREATE TABLE **Enseignant UNDER** **Personne**

(**Grade** **varchar(25)**

CONSTRAINT CK_Enseignant_Grade

CHECK (Grade IN ('Vacataire', 'Moniteur','ATER', 'MCF', 'PROF')))

)

GENERATED

Colonne calculée (norme SQL2003) :

```
CREATE TABLE Etudiant
```

```
(  
Etudiant_ID    integer,  
Ville          varchar(25),  
Nom            varchar(25) NOT NULL,  
Prenom        varchar(25) NOT NULL,  
Date_Naissance date NOT NULL,  
Rue           varchar(50) DEFAULT NULL,  
CP            varchar(9) DEFAULT NULL,
```

```
Adresse GENERATED ALWAYS AS (rue || ' ' || CP || ' ' || Ville)
```

```
Telephone     varchar(10) DEFAULT NULL,  
Fax           varchar(10) DEFAULT NULL,  
Email        varchar(100) DEFAULT NULL,
```

```
CONSTRAINT PK_Etudiant PRIMARY KEY (Etudiant_ID)
```

```
);
```

Assertion

```
CREATE ASSERTION <nom contrainte>  
[ {BEFORE COMMIT |  
  AFTER {INSERT | DELETE | UPDATE[OF (Attributs)]} ON  
  <Relation>} ...]  
CHECK <Condition>  
[FOR [EACH ROW OF] <Relation> ]
```

```
CREATE ASSERTION CA_Place_Universite  
BEFORE INSERT ON Etudiant  
CHECK( (SELECT SUM(Capacite) FROM Salle)  
  > (SELECT COUNT(*) FROM Etudiant)  
  )
```

Déclencheur

```
CREATE [OR REPLACE] TRIGGER nom {BEFORE | AFTER}  
événement_déclencheur ON nom_table  
[FOR EACH ROW]  
[WHEN (condition) ]  
bloc PL/SQL sous Oracle  
| inst_de_suppr | inst_de_modif | instr_d_ajout | ERROR en SQL2  
  
événement_déclencheur = INSERT, UPDATE, DELETE
```

- Déclencheur de *niveau instruction* : pas de clause **FOR EACH ROW**
- Déclencheur de *niveau ligne* : variables liens *:new* et *:old*
 - **INSERT** : valeurs à insérer dans *:new.nom_colonne*
 - **UPDATE** : valeur originale dans *:old.nom_colonne*, nouvelle valeur dans *:new.nom_colonne*
 - **DELETE** : valeur en cours de suppression *:old.nom_colonne*

Exemple de déclencheur sous Oracle

```
CREATE OR REPLACE TRIGGER Enseignant_Actif
BEFORE DELETE ON Enseignant
FOR EACH ROW
declare
    counter number;
begin
    SELECT count(*) INTO counter
    FROM Enseignements
    WHERE Enseignant_ID = :old.Enseignant_ID;
    if counter > 0 then
        raise_application_error (-20800, 'Enseignant actif ne
        pouvant être supprimé');
    end if;
end;
```

Exemple de déclencheur avec clause WHEN

```
CREATE OR REPLACE TRIGGER UPD_salaire_personnel
BEFORE UPDATE salaire ON Personnel
FOR EACH ROW
WHEN (:old.salaire > :new.salaire)
declare
    salaire_diminution EXCEPTION;
begin
    raise salaire_diminution ;
    when salaire_diminution then
        raise_application_error(-20001, 'Le salaire ne peut pas
diminuer ')
end;
```

Définir des fonctions

Sous PostgreSQL :

```
CREATE OR REPLACE FUNCTION GetSalleCapaciteSuperieurA(int)  
RETURNS SETOF Salle  
AS '  
    SELECT * FROM Salle WHERE Capacite > $1;  
    '  
LANGUAGE SQL;  
  
SELECT * FROM GetSalleCapaciteSuperieurA(300) ;
```

Corps d'un déclencheur sous PostgreSQL

```
-- Fonction contenant le corps du trigger
CREATE OR REPLACE FUNCTION FunctionTriggerReservation ()
    RETURNS trigger AS
' DECLARE
    resa Reservation.Reservation_ID%TYPE;
BEGIN
    SELECT INTO resa Reservation_ID
    FROM Reservation
    WHERE ...
    IF FOUND THEN RAISE EXCEPTION 'Réservation impossible,
    salle occupée à la date et aux horaires demandés';
ELSE RETURN NEW;
    END IF;
END; '
LANGUAGE 'plpgsql';
```


Déclencheur sous PostgreSQL

-- code SQL de creation du trigger

```
CREATE TRIGGER InsertionReservation  
BEFORE INSERT ON Reservation  
FOR EACH ROW  
EXECUTE PROCEDURE  
    FunctionTriggerReservation ();
```

Exemple de déclencheur (1/4)

Sous PostgreSQL :

```
CREATE TABLE Personne
(
  personneId serial,
  nom varchar(25) NOT NULL,
  prenom varchar(25) NOT NULL,
  dateNaissance date NOT NULL,
  sexe varchar(1) NOT NULL,
  CONSTRAINT PK_Personne PRIMARY KEY (personneId),
  CONSTRAINT UNE_Personne UNIQUE (nom,prenom,dateNaissance),
  CONSTRAINT CK_Sexe CHECK (sexe IN ('M','F'))
);

CREATE TABLE Parente
(
  parentId int NOT NULL,
  enfantId int NOT NULL,
  CONSTRAINT PK_Parente PRIMARY KEY (parentId,enfantId),
  CONSTRAINT FK_Parente_1 FOREIGN KEY (parentId) REFERENCES Personne
(personneId),
  CONSTRAINT FK_Parente_2 FOREIGN KEY (enfantId) REFERENCES Personne
(personneId),
  CONSTRAINT CK_Parente CHECK (parentId!=enfantId)
);
```

Exemple de déclencheur (2/4)

Sous PostgreSQL :

```
-- Fonction contenant le corps du trigger
CREATE OR REPLACE Fonction FonctionVerifDate() returns trigger as
'Begin
  IF (SELECT  Personne.dateNaissance FROM Personne WHERE
PersonneID=NEW.parentID) >
    (SELECT  Personne.dateNaissance FROM Personne WHERE
PersonneID=NEW.enfantID)
  THEN
    RAISE exception 'Un parent ne peut pas avoir une date de
naissance > a celle de son enfant! ' ;
  ELSE
    return NEW;
END IF;
END ;'
LANGUAGE 'plpgsql';

-- code SQL de creation du trigger
CREATE TRIGGER  VerifDate BEFORE INSERT ON Parente
FOR EACH ROW
EXECUTE procedure FonctionVerifDate();
```

Exemple de déclencheur (3/4)

Sous PostgreSQL :

Relation *Etudiant*

etudiantid integer	nom character varying (25)	prenom character varying (25)	date_naissance date
1	Gamotte	Albert	2001-01-21
2	Computing	Claude	1999-03-15
3	Zarella	Maude	2000-09-19
4	Deblouze	Agathe	2001-03-02

Relation *Binome*

binomeid integer	idmembre1 integer	idmembre2 integer
2	1	3
3	2	[null]
4	4	[null]

Relation *ReleveNotes*

releveid integer	etudiantid integer	typeid integer	note double precision
1	4	1	20
3	4	3	20
4	3	1	15
5	2	1	8

Relation *TypeEval*

typeid integer	intitule character varying (25)
1	Examen
2	TP
3	Partiel
4	CC

```
INSERT INTO ReleveNotes (EtudiantID,TypeID,Note) VALUES (1,2,18);
```

Exemple de déclencheur (4/4)

Sous PostgreSQL :

```
CREATE OR REPLACE FUNCTION FunctionTriggerReleveNote() RETURNS trigger AS
' DECLARE
    eId Etudiant.EtudiantID%TYPE;

BEGIN
    SELECT INTO eId IDMembre2 FROM Binome WHERE (IDMembre1=NEW.EtudiantID);
    IF eid IS NOT NULL THEN
        INSERT INTO ReleveNotes(EtudiantID,TypeID,Note) VALUES (eId,2,NEW.Note);
    END IF;
    RETURN NEW;
END;'
LANGUAGE 'plpgsql';

CREATE TRIGGER InsertionNoteTPNote
AFTER INSERT ON ReleveNotes
FOR EACH ROW
WHEN (NEW.TypeID=2)
EXECUTE PROCEDURE FunctionTriggerReleveNote();
```

Autres exemples de déclencheurs

Sous PostgreSQL :

- **Exercice 1 de l'examen de Janvier 2021 (pages 46-47)**
- **Exercice 1 de l'examen de Janvier 2020 (pages 54-56)**
- **Exercice 1 de l'examen de Mai 2019 (pages 64-66)**
- **Exercice 1 de l'examen de Mai 2018 (pages 75-77)**

Modification du schéma

ALTER TABLE table

ADD ADD COLUMN nomAttribut Type

| RENAME COLUMN nomAttribut TO nouveauNom

| COLUMN nomAttribut SET NOT NULL

| ADD PRIMARY KEY AttributsPK

| ADD/DROP CONSTRAINT nom_contrainte ...

DROP TABLE table

CREATE VIEW vue (col1, col2)

AS SELECT ...

DROP VIEW vue

CREATE [UNIQUE] INDEX nom_index ON table (col1,col ...)

Exemple de création d'une base de données à partir d'une base existante

id	numsaizon	idcandidat	nomcandidat	nomepreuve	numsemaine	datedebut	datefin	nomgateau	tablierbleu	elimine	classement	nbtalriers bleus	nbmeilleur classement
1	10	1	Aya	Revisite	1	07/10/2021	30/12/2021	Tarte avocat(e)	false	false	1	0	1
2	10	2	Maud	Revisite	1	07/10/2021	30/12/2021	Tarte cafe	false	false	2	1	3
3	10	3	Mohamed	Revisite	1	07/10/2021	30/12/2021	ONE, TWO, THREE	false	false	3	0	0
4	10	1	Aya	Revisite	2	07/10/2021	30/12/2021	Oeuf de caille en chocolat blanc	false	false	2	0	1
5	10	2	Maud	Revisite	2	07/10/2021	30/12/2021	Oeufs a la coque	false	false	1	1	3

```
CREATE TABLE MeilleurPatissier
( ID serial PRIMARY KEY,
  NumSaison integer,
  IDCandidat Integer NOT NULL,
  NomCandidat varchar(25) NOT NULL,
  NomEpreuve varchar(25) NOT NULL,
  NumSemaine integer NOT NULL,
  DateDebut date NOT NULL,
  DateFin date NOT NULL,
  NomGateau varchar(100) NOT NULL,
  TablierBleu bool DEFAULT False,
  Elimine bool DEFAULT False,
  Classement integer,
  NbTabliersBleus integer DEFAULT 0,
  NbMeilleursClassements integer DEFAULT 0,
  CONSTRAINT CK_NomEpreuve CHECK (NomEpreuve
  IN ('Revisite', 'Technique', 'Creative', 'Carte
  blanche')),
  CONSTRAINT UN_CandidatSemaineEpreuve UNIQUE
  (IDCandidat, NumSaison, NumSemaine, NomEpreuve)
);
```


CREATE TABLE AS SELECT et ALTER TABLE (1/2)

```
CREATE TABLE Saison AS  
SELECT DISTINCT NumSaison, DateDebut, DateFin  
FROM MeilleurPatissier  
ORDER BY NumSaison;
```

```
ALTER TABLE Saison ADD PRIMARY KEY (NumSaison);
```

```
ALTER TABLE Saison ADD CONSTRAINT UN_Saison UNIQUE  
(DateDebut, DateFin);
```

```
ALTER TABLE Saison ADD CONSTRAINT CK_Saison CHECK (DateDebut  
< DateFin);
```

CREATE TABLE AS SELECT et ALTER TABLE (2/2)

```
CREATE TABLE Candidat AS
```

```
SELECT DISTINCT NumSaison, IDCandidat, NomCandidat,  
NbTabliersBleus, NbMeilleursClassements FROM MeilleurPatissier  
ORDER BY NumSaison, IDCandidat;
```

```
ALTER TABLE Candidat ADD PRIMARY KEY (NumSaison, IDCandidat);
```

```
ALTER TABLE Candidat ADD CONSTRAINT UN_Candidat UNIQUE  
(NumSaison, NomCandidat);
```

```
ALTER TABLE Candidat ADD CONSTRAINT FK_Candidat FOREIGN KEY  
(NumSaison) REFERENCES Saison (NumSaison);
```

```
ALTER TABLE Candidat ALTER COLUMN NbTabliersBleus SET DEFAULT 0;
```

```
ALTER TABLE Candidat ALTER COLUMN NbMeilleursClassements SET  
DEFAULT 0;
```

Vue en SQL

Pour le SGBD : c'est un requête

```
CREATE VIEW vue (col1, col2)  
AS SELECT ...
```

Exemple :

```
CREATE VIEW Info_Non_Confidentielle_Etudiant  
AS SELECT Nom, Prenom, Email FROM Etudiant;
```

Pour l'utilisateur : s'utilise dans un FROM

```
SELECT * FROM Info_Non_Confidentielle_Etudiant  
WHERE NOM= 'Gamotte';
```

Vue matérialisée

Comme une vue, mais le résultat persiste sous la forme d'une table :

```
CREATE MATERIALIZED VIEW NomVue (col1, col2)  
AS SELECT ...
```

- Accès aux données d'une vue matérialisée souvent bien plus rapide que l'accès aux tables sous-jacentes directement ou par l'intermédiaire d'une vue
- Mais données pas toujours fraîches
- Possibilité de rafraîchir les données :

```
REFRESH MATERIALIZED VIEW NomVue;
```

Chap VII – Passage au relationnel

Pour créer une base de données :

Étape N° 1 : Concevoir la base de données

- = Réfléchir à ce que va contenir la base de données et comment structurer les données
- = Modélisation de la base de données

Démarche :

- Établir la liste des données devant être stockées dans la base
- Définir la structure des données

⇒ **Modèle conceptuel de données**

(Modèle Entité/Association ou UML - hors programme – toujours donné)

Étape N° 2 : Définir le modèle relationnel = passage au relationnel

- = le **schéma** des relations de la base de données

Modélisation

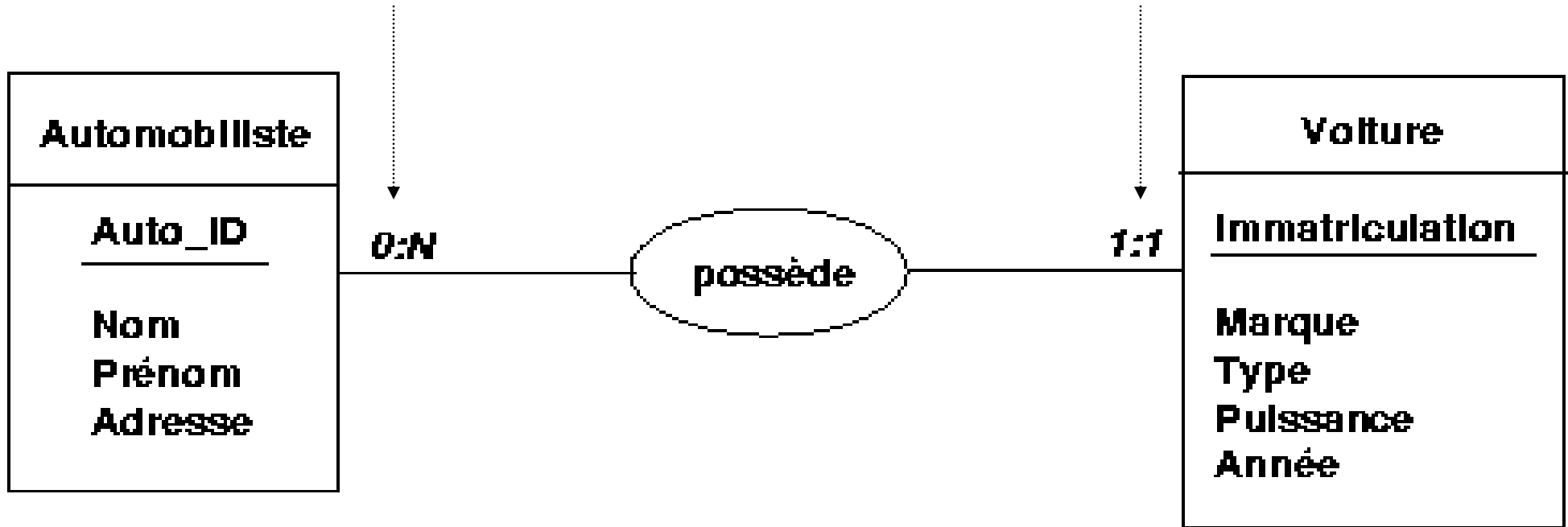
Méthodologie à suivre pour modéliser un problème

- Déterminer les **entités/classes** et **attributs** :
 - entité/instance de classe = objet décrit par de l'information
 - objet caractérisé uniquement par un identifiant = attribut(s)
 - attribut multi-valué ou avec une association 1:N = entité ou instance
 - attacher les attributs aux ensemble d'entités/classes qu'ils décrivent le plus directement
 - éviter au maximum les identificateurs composites
- Identifier les **généralisations-spécialisations/héritage**
- Définir les **associations**
 - éliminer les associations redondantes
 - éviter les associations n-aires
 - calculer les **cardinalités** de chaque association

Modélisation Entité/Association (Format Merise)

*Un automobiliste possède
entre zéro et N voitures*

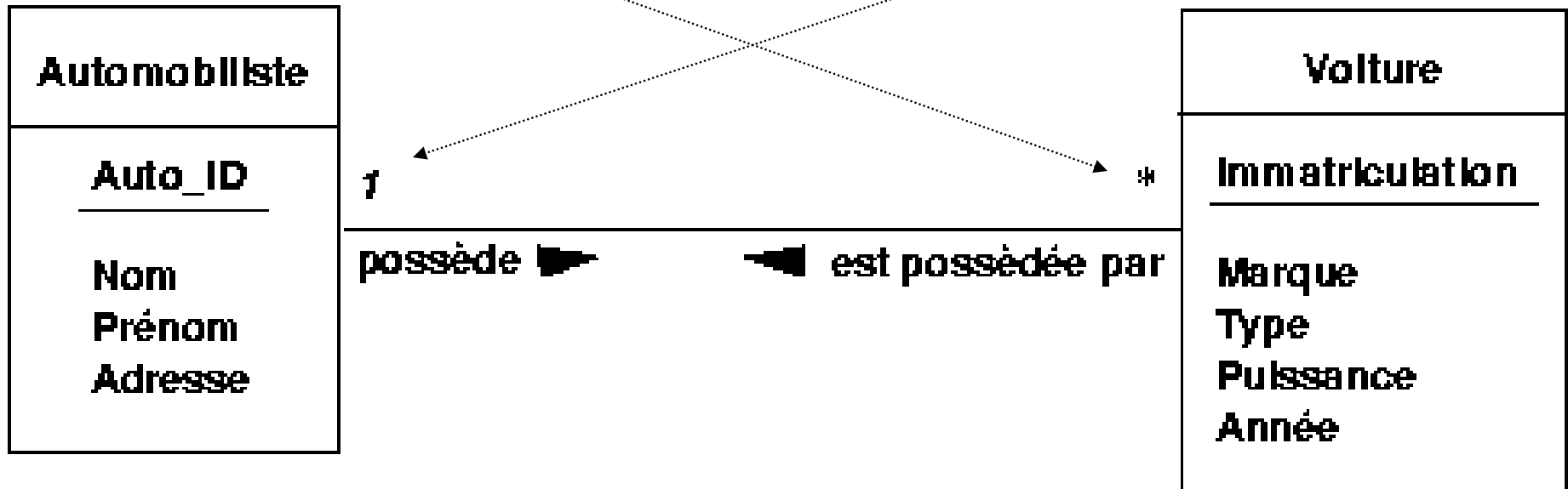
*Une voiture a un et un
seul propriétaire*



Modélisation UML

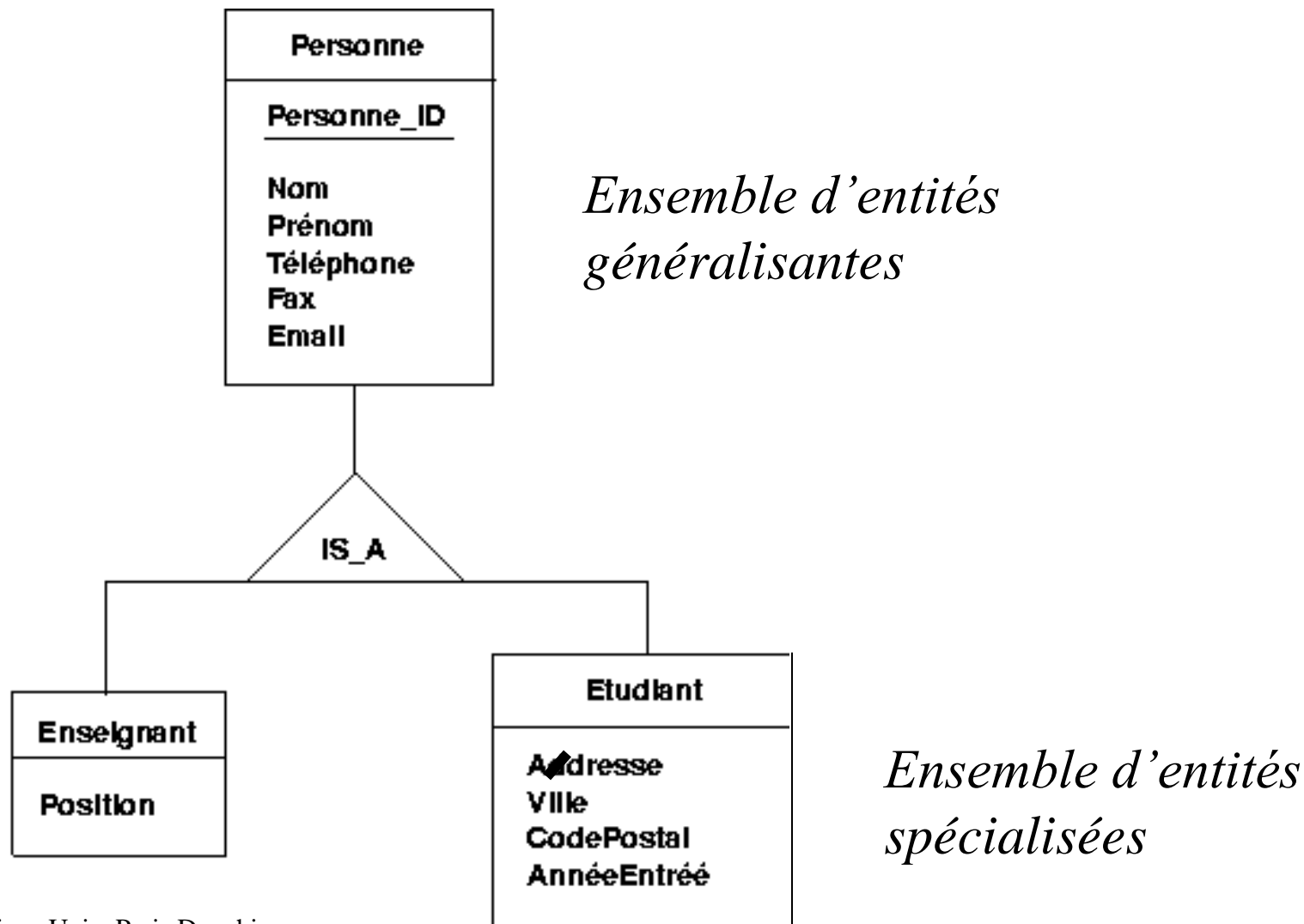
*Un automobiliste possède
entre zéro et N voitures*

*Une voiture a un et un
seul propriétaire*

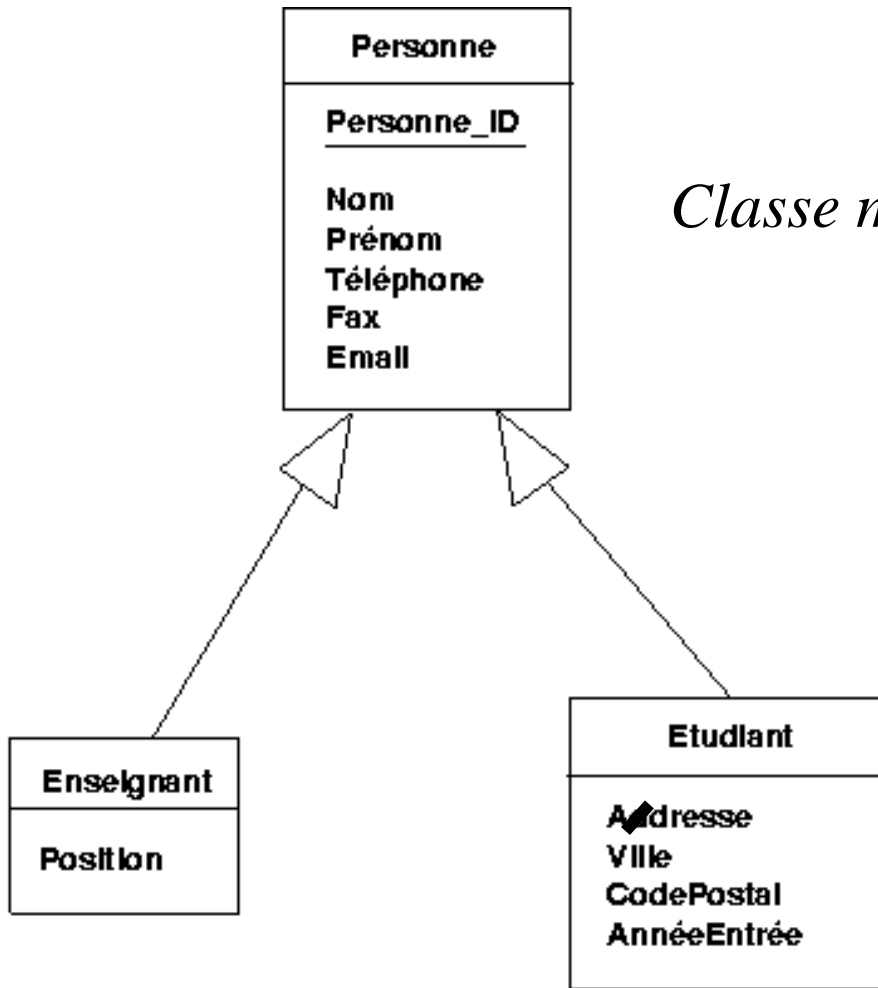


Attention : petite liberté prise avec UML, les attributs soulignés ici ne correspondent pas à des attributs dérivés mais aux identificateurs (pour ne pas les oublier lors du passage au relationnel!!)

Généralisation/Spécialisation (E/A - Merise)



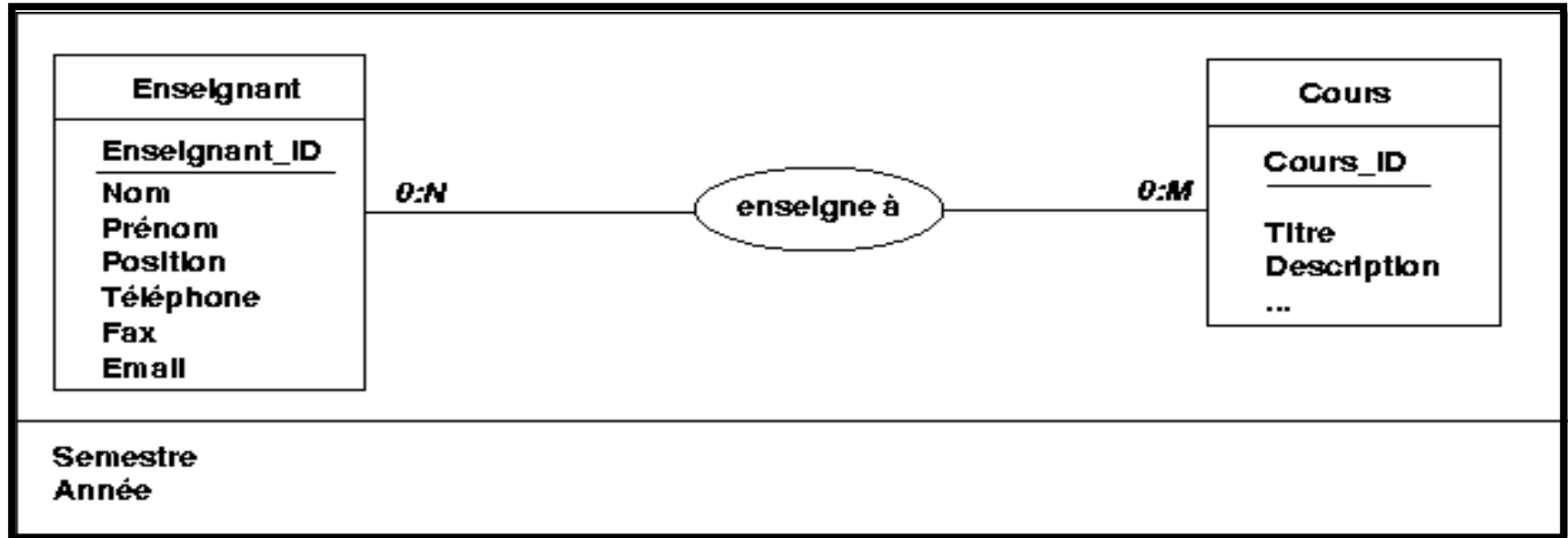
Héritage (UML)



Classe mère / Sur-classe

Classes dérivées ou filles / sous-classes

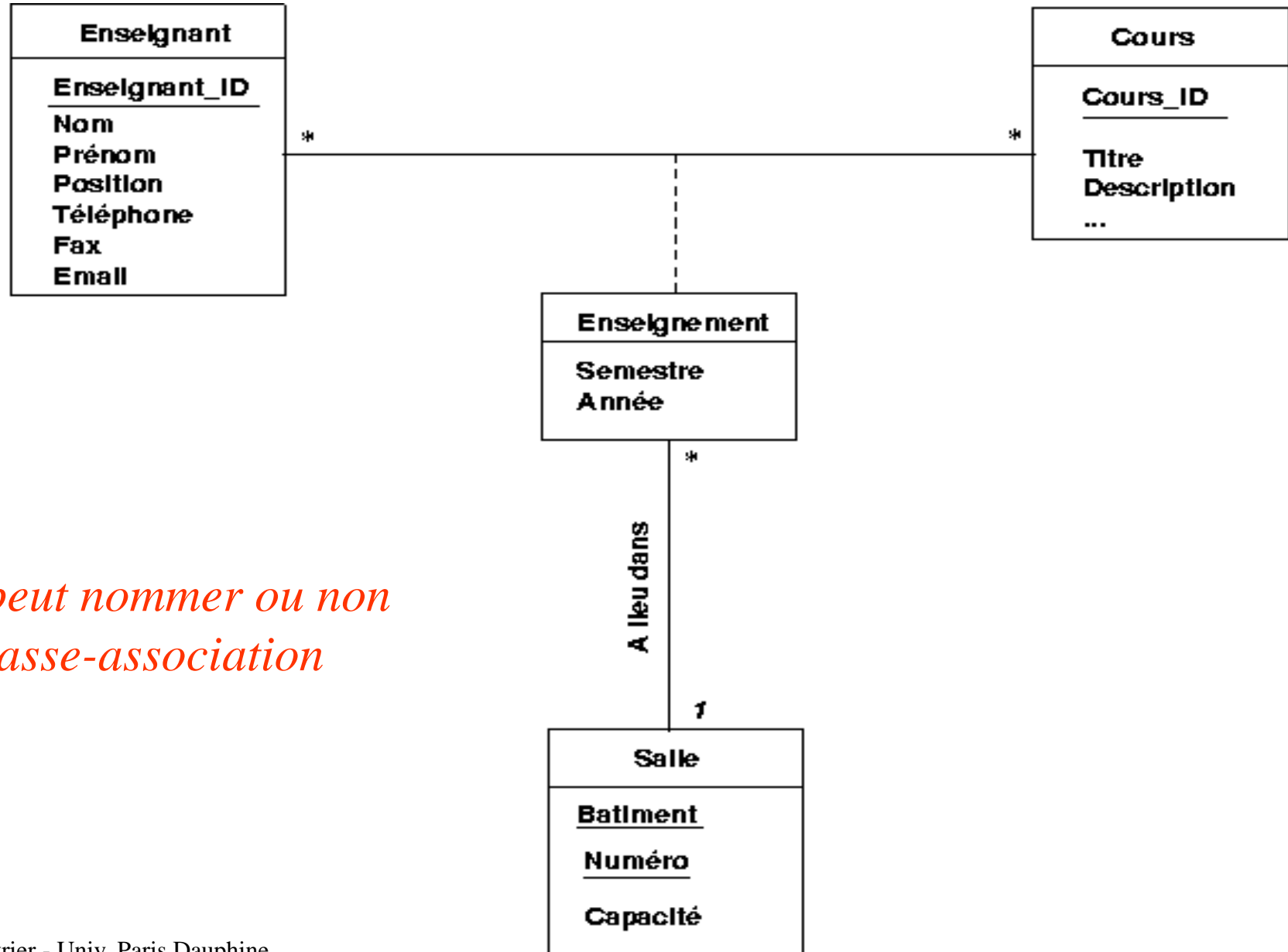
Agrégat (E/A - Merise)



On peut nommer ou non l'agrégat

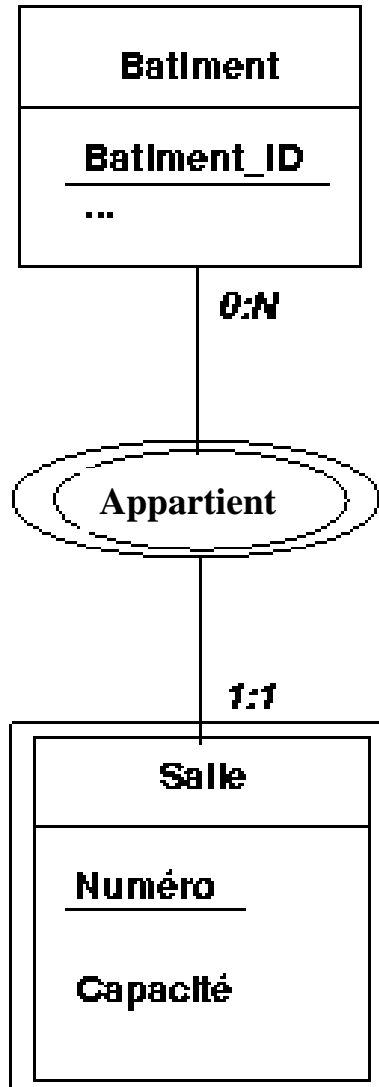


Classe-Association (UML)



On peut nommer ou non la classe-association

Entité Faible (E/A - Merise)

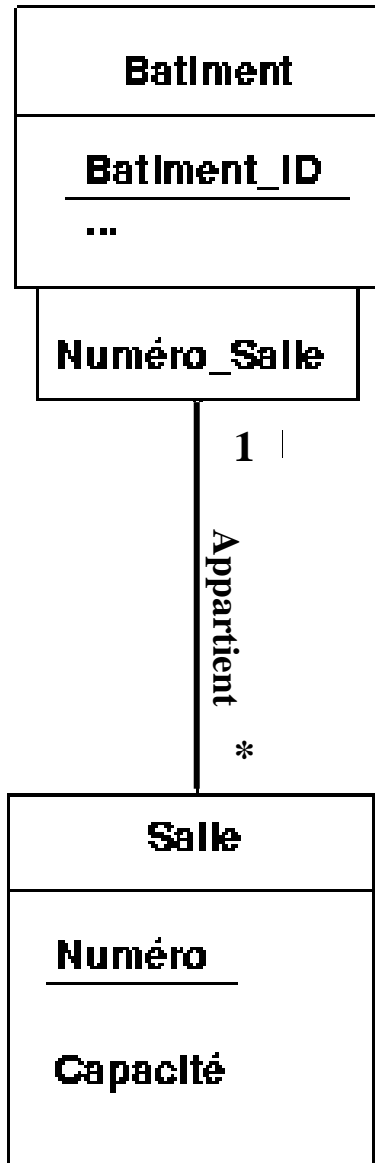


Chaque salle a un numéro unique dans un bâtiment donné

Ex. Salle 1 du bâtiment A et Salle 1 du bâtiment C

Pour distinguer une salle d'une autre, il faut connaître le bâtiment auquel elle est rattachée

Association qualifiée (UML)

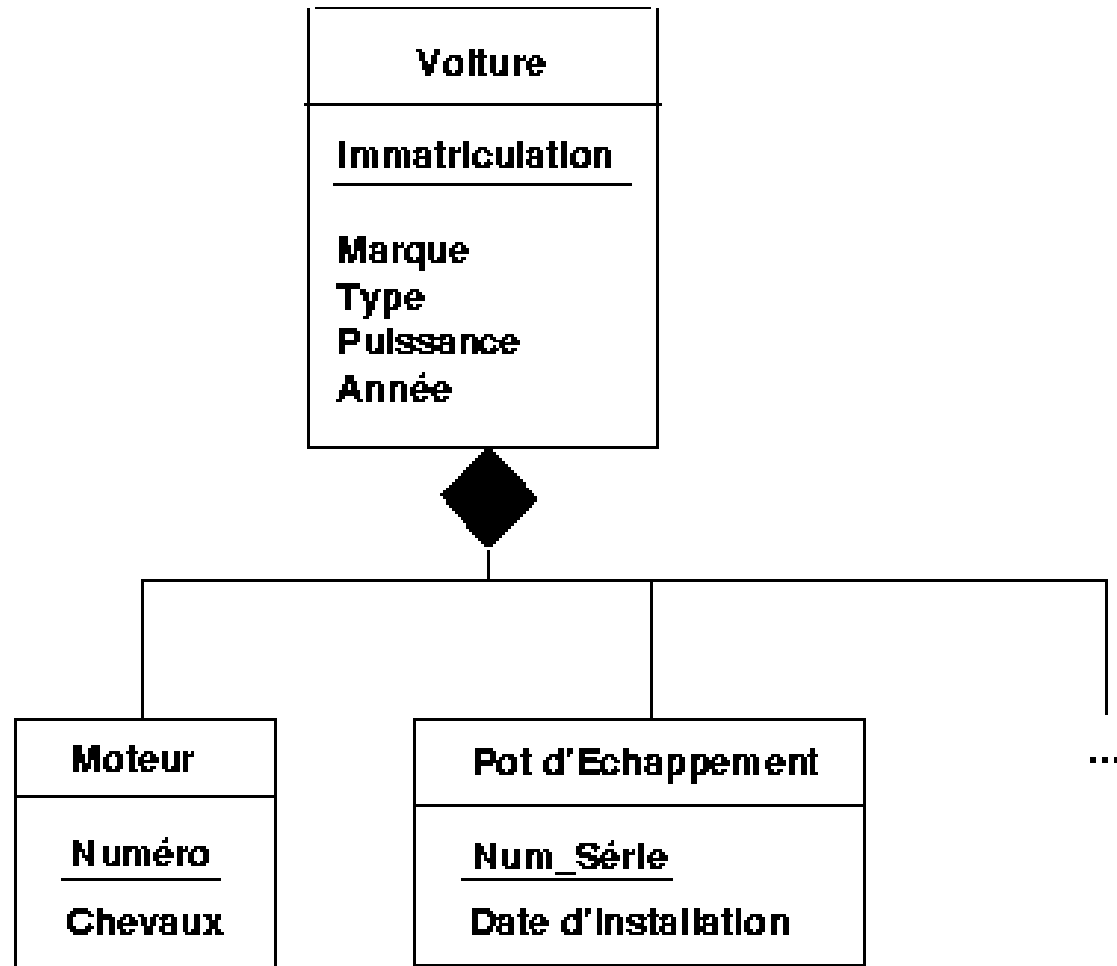


Chaque salle a un numéro unique dans un bâtiment donné

Ex. Salle 1 du bâtiment A et Salle 1 du bâtiment C

Pour distinguer une salle d'une autre, il faut connaître le bâtiment auquel elle est rattachée

Composition (UML)



Dépendances fonctionnelles

Un attribut (ou un ensemble d'attributs) Y **dépend fonctionnellement** d'un attribut (ou ensemble d'attributs) X si :

étant donné une valeur de X , il lui correspond une valeur unique de Y (\forall l'instant considéré)

$X \rightarrow Y$: Y **dépend fonctionnellement de X**
ou X **détermine Y**

Déclaration des dépendances **au niveau du schéma conceptuel**

Exemple de dépendances fonctionnelles

Volture
<u>Immatriculation</u>
Marque
Type
Puissance
Année

identificateur

Tous les autres attributs

Immatriculation → Marque, Type, Puissance, Année

~~Marque, Type, Puissance, Année → Immatriculation~~

Type → Marque

Ex. Le type "C3" sera toujours associé, dans la base de données, à la marque "Citroën".

Enseignant
<u>Enseignant_ID</u>
Nom
Prénom
Position
Téléphone
Fax
Email

EnseignantID → Nom, Prénom, Position ...

Nom, Prénom, Position, ... → Enseignant_ID

Si un numéro de téléphone est associé à un seul enseignant :

Telephone → Enseignant_ID

Passage au relationnel (1/2)

2 étapes :

- 1. Transformations des ensembles d'entités/classes :** à chaque ensemble d'entités/classe correspond une relation avec les mêmes attributs.

Attention aux ensembles d'entités faibles/association qualifiée et aux généralisation-spécialisations/Hiérarchie de classes.

- 2. Transformations des associations :**

- a) Suppression/Fusion de relations créées à l'étape 1**
- b) Modification de relations créées à l'étape 1**
- c) Ajout de nouvelles relations**

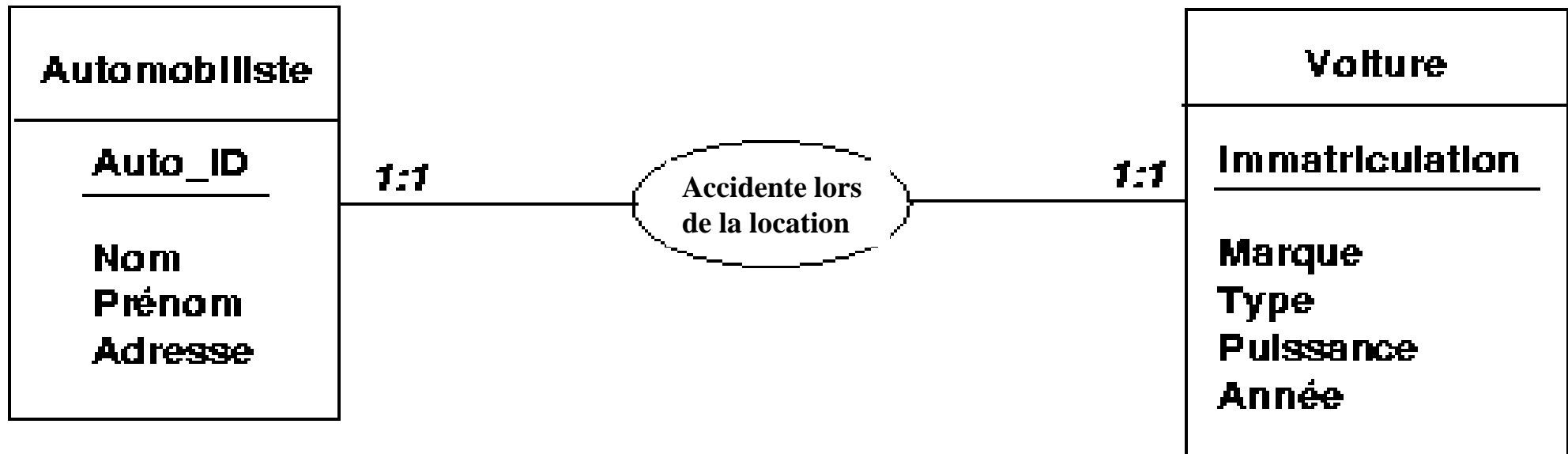
En fonction des cardinalités/multiplicités

Passage au relationnel (2/2)

Transformation des ensembles d'entités :

- **chaque ensemble d'entités/classes $E \Rightarrow$**
 - une relation R avec les mêmes attributs que E
 - l'identificateur de E devient la clé primaire de R
- **chaque ensemble d'entités faibles/association qualifiée $E \Rightarrow$**
 - une relation R qui comprend tous les attributs de E +
 - l'identificateur de l'ensemble d'entités fortes/classe associé(e)
- **généralisation-spécialisation/héritage \Rightarrow**
 - l'ensemble d'entités généralisante/classe mère $E \Rightarrow$ une relation R
 - chaque ensemble d'entités E_i spécialisé/classe fille
 \Rightarrow une relation R_i dans laquelle l'identifiant est de même domaine que l'identifiant de E et est une clé étrangère faisant référence à la clé primaire de E

Transformation des ensembles d'associations E/A : fusion



Automobiliste (Auto_ID, Nom, Prénom, Adresse)

Voiture (Immatriculation, Marque, Type, Puissance, Année)

Comment faire le lien ?

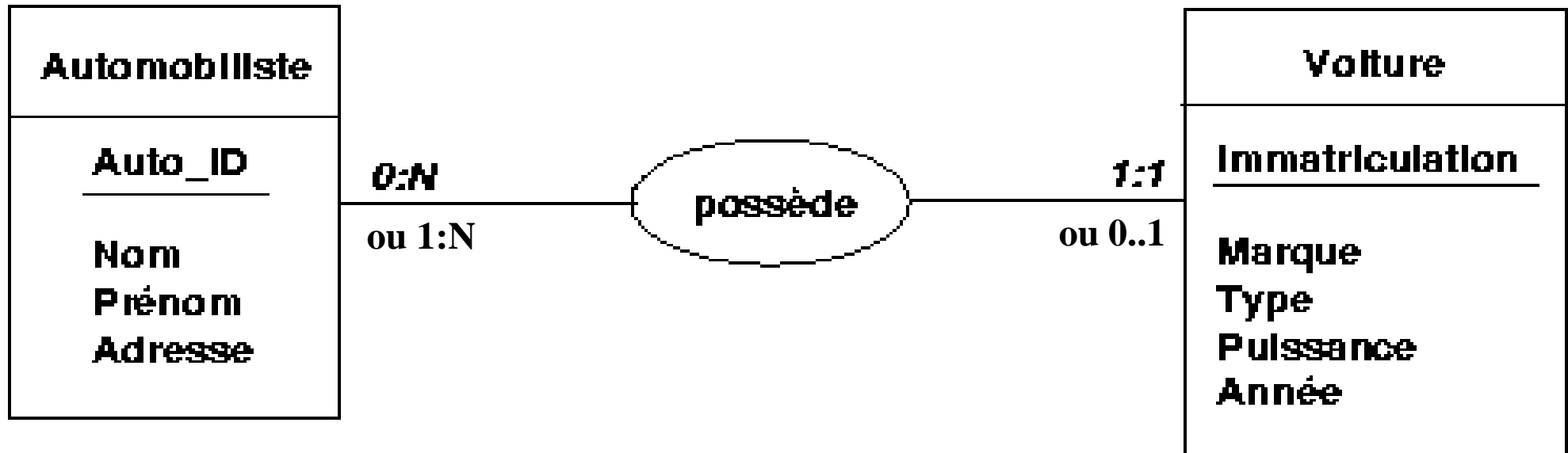
Transformation des ensembles d'associations UML : fusion



Accidente (Auto_ID, Nom, Prénom, Adresse, Immatriculation, Marque, Type, Puissance, Année)

On peut choisir l'un ou l'autre comme clé primaire

Transformation des ensembles d'associations E/A : clé étrangère

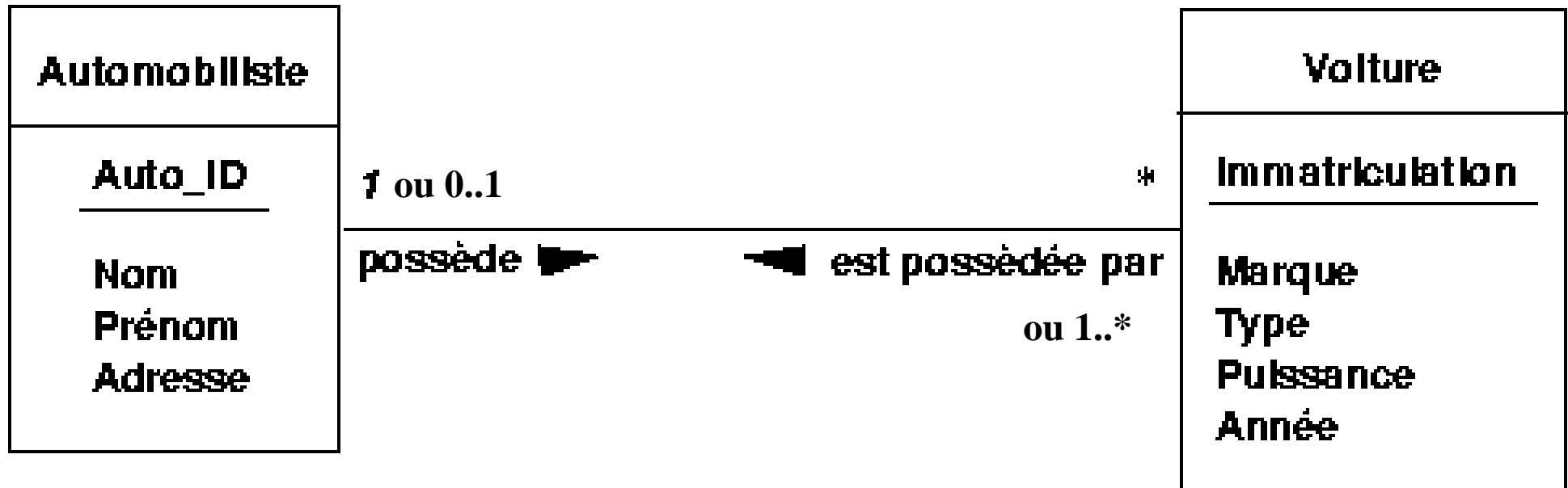


Automobiliste (Auto_ID, Nom, Prenom, Adresse)

Voiture (Immatriculation, Marque, Puissance, Type, Annee, #Auto_ID)

NB : #Auto_ID fait référence à Auto_ID de Automobiliste

Transformation des ensembles d'associations UML : clé étrangère

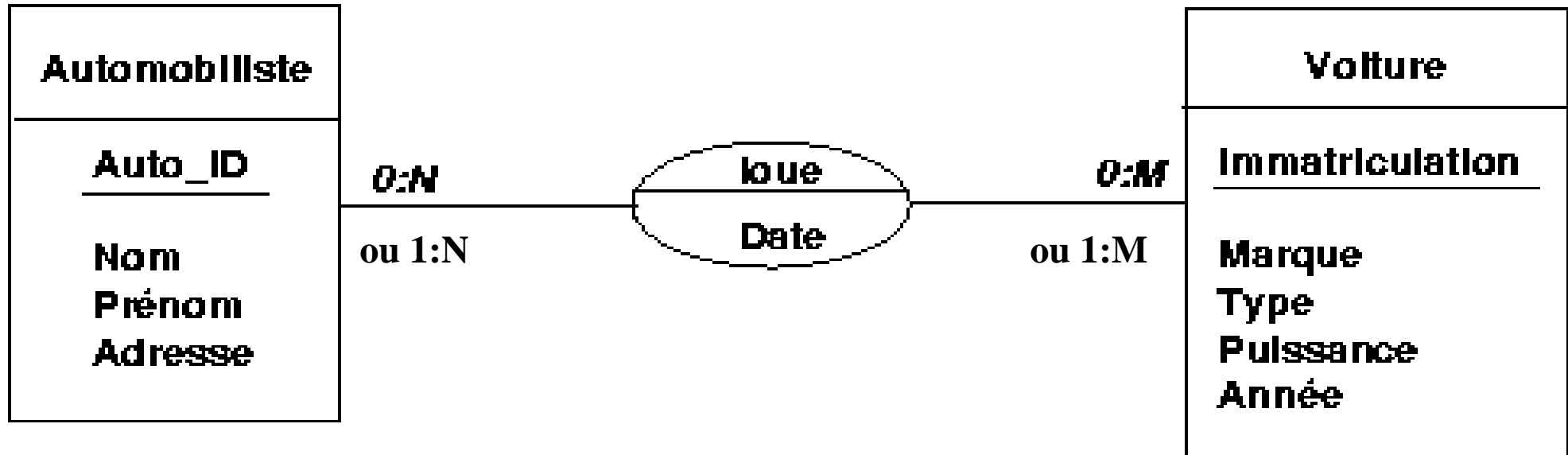


Automobiliste (Auto_ID, Nom, Prenom, Adresse)

Voiture (Immatriculation, Marque, Puissance, Type, Annee, **#Auto_ID**)

NB : #Auto_ID fait référence à Auto_ID de Automobiliste

Transformation des ensembles d'associations E/A : nouvelle relation



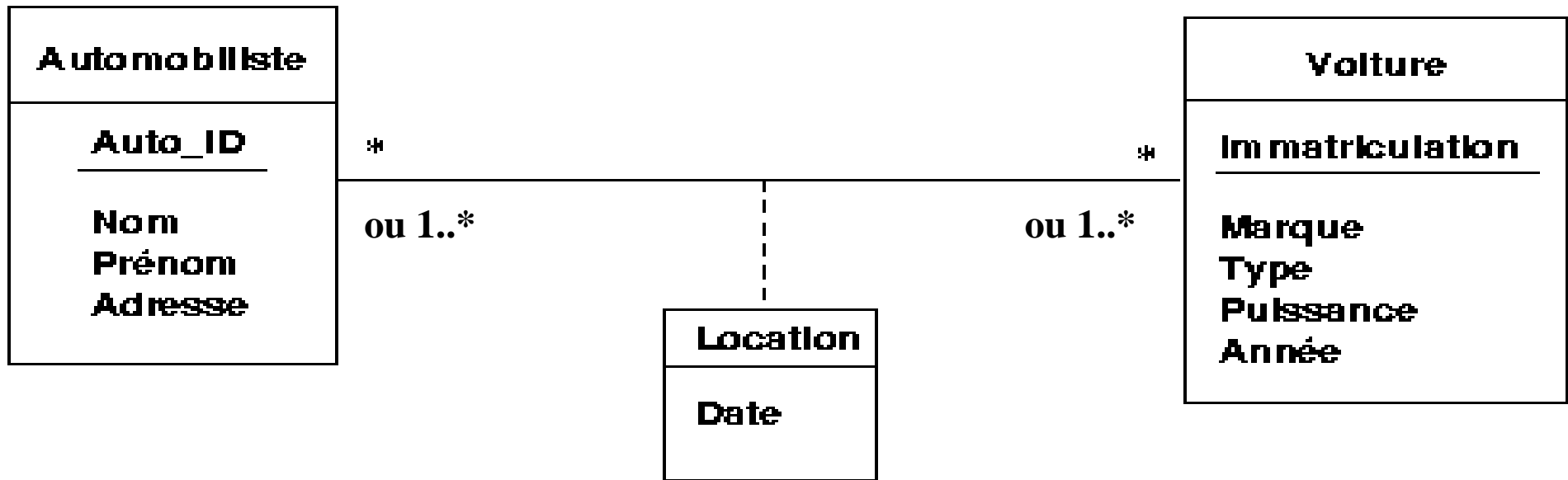
Automobiliste (Auto_ID, Nom, Prénom, Adresse)

Voiture (Immatriculation, Marque, Puissance, Type, Année)

Location (#Auto_ID, #Immatriculation, Date) ou

Location (Loc_ID, #Auto_ID, #Immatriculation, Date)

Transformation des ensembles d'associations UML : nouvelle relation



Automobiliste (Auto_ID, Nom, Prenom, Adresse)

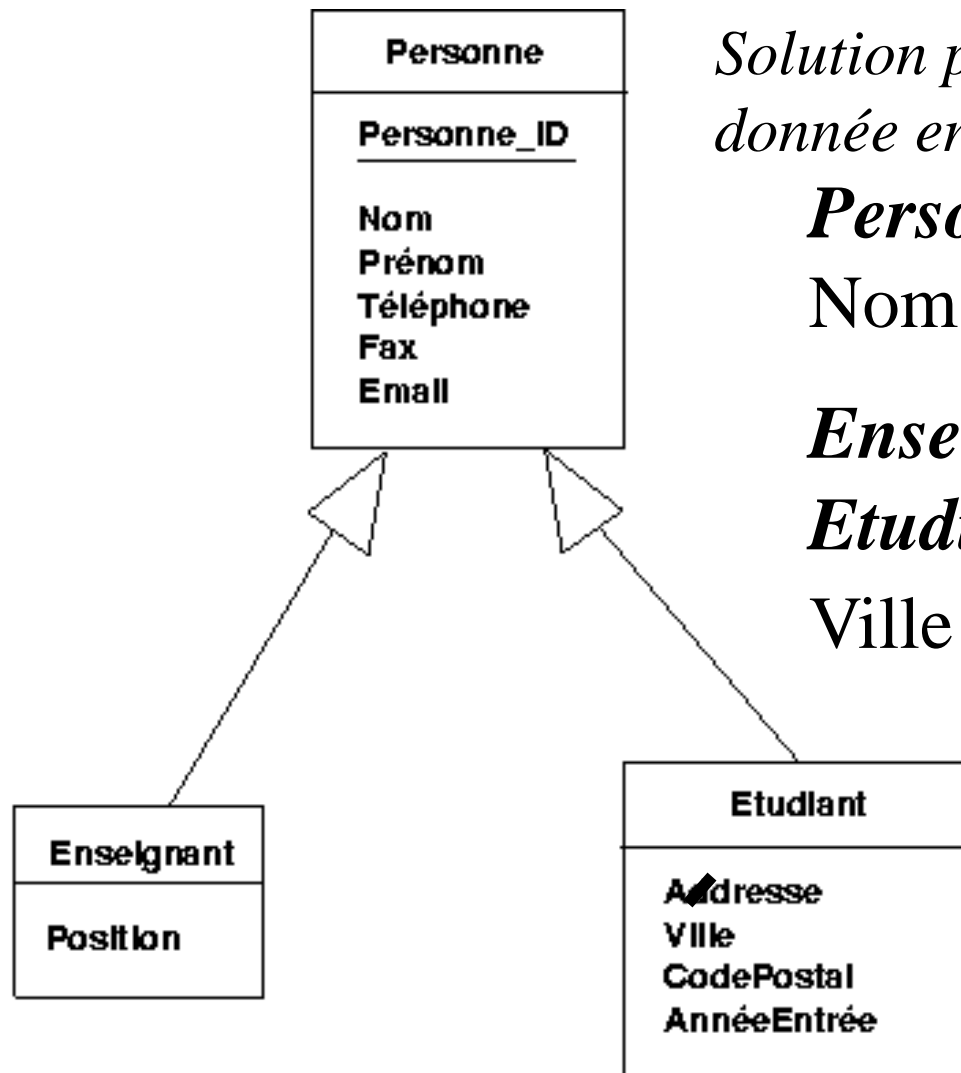
Voiture (Immatriculation, Marque, Puissance, Type, Année)

Location (#Auto_ID, #Immatriculation, Date) ou

Location (Loc_ID, #Auto_ID, #Immatriculation, Date)

Transformation des concepts

Généralisation-Spécialisation / Héritage



Solution possible (une autre sera donnée en cours) :

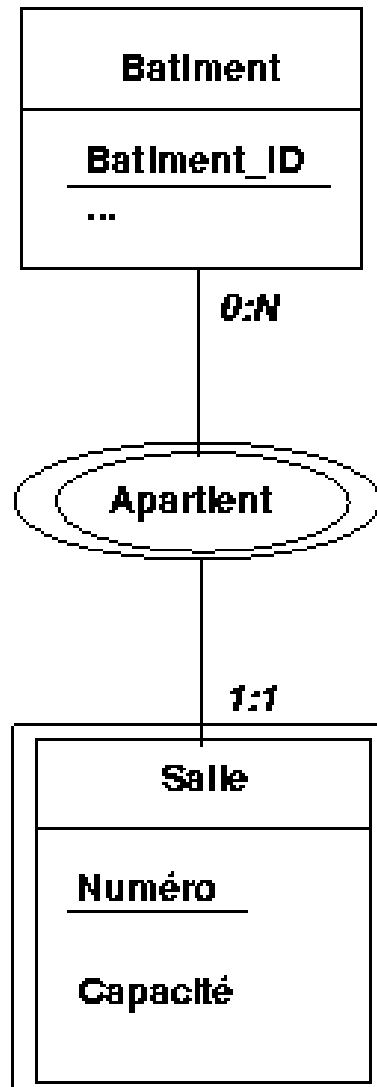
***Personne** (Personne_ID,
Nom, Prénom, Telephone ...)*

***Enseignant** (#Personne_ID, Position)*

***Etudiant** (#Personne_ID, Adresse,
Ville ...)*

NB : #*Personne_ID* dans
Enseignant et *Etudiant* font
référence à *Personne_ID* dans
Personne

Transformation des entités faibles E/A



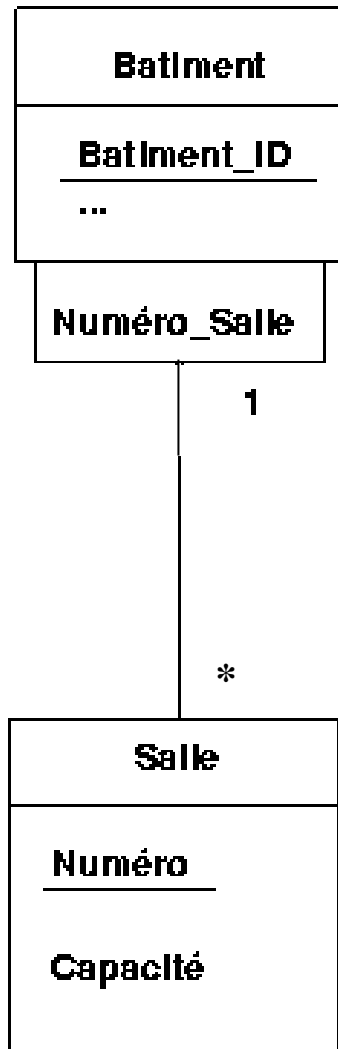
Bâtiment (Batiment_ID, ...)

Salle (Numéro, #Batiment_ID, Capacité)

NB : *Une salle est identifiée par le couple (Numéro, #Bâtiment_ID)*

#Bâtiment_ID fait référence à Bâtiment_ID de Bâtiment

Transformation des associations qualifiées UML



Bâtiment (Batiment_ID, ...)

Salle (Numero, #Batiment_ID, Capacité)

NB : *Une salle est identifiée par le couple (Numero, #Bâtiment_ID) ;*

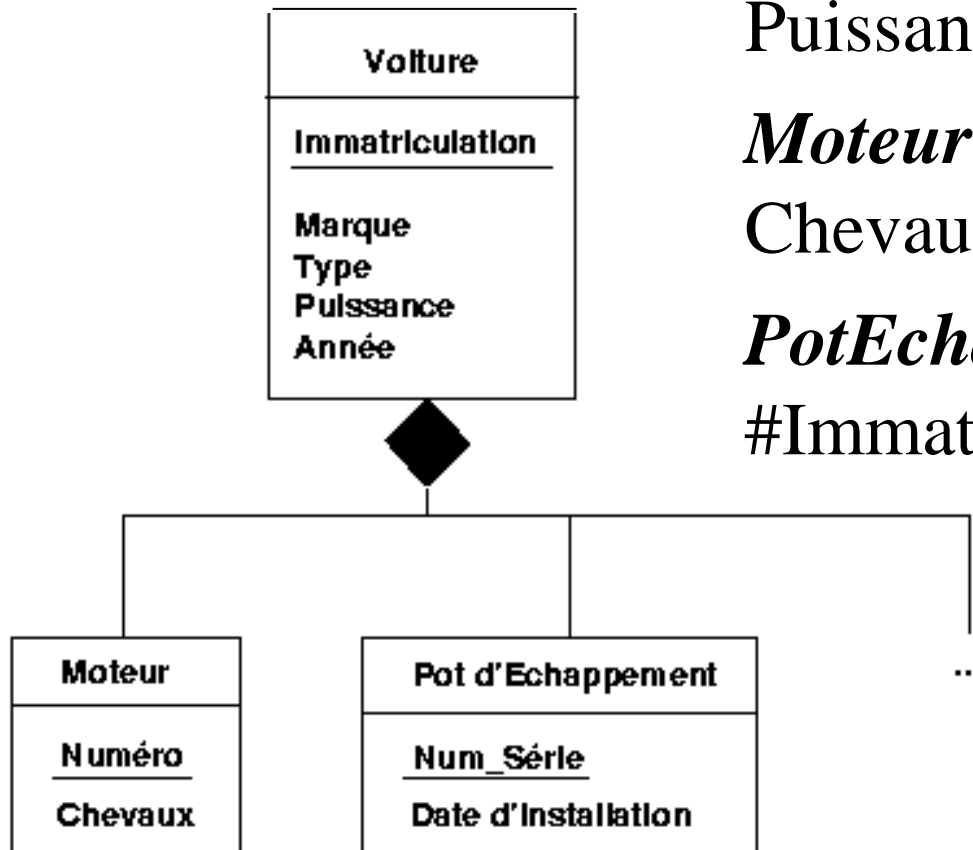
#Bâtiment_ID fait référence à Bâtiment_ID de Bâtiment

Transformation de la composition UML

Voiture (Immatriculation, Marque, Puissance, Type, Année)

Moteur (Numéro, #Immatriculation, Chevaux)

PotEchappement (Num_Série, #Immatriculation, DateInstallation)



Dépendances fonctionnelles

Ne pas oublier de définir les DF :

- ***Accidente*** (Auto_ID, Nom, Prénom, Adresse, Immatriculation, Marque, Type, Puissance, Année)
Auto_ID → Nom, Prénom, Adresse
Immatriculation → Marque, Type, Puissance, Année
Type → Marque
Auto_ID → Immatriculation et Immatriculation → Auto_ID
Choisir l'un ou l'autre comme clé primaire
- ***Voiture*** (Immatriculation, Marque, Puissance, Type, Année, Auto_ID)
Immatriculation → Auto_ID ~~Auto_ID → Immatriculation~~
+ *les Dépendances fonctionnelles de Voiture*
- ***Location*** (Auto_ID, Immatriculation, Date)
Pas de dépendance non triviale

Chap VIII - Dépendances fonctionnelles

Soit X et Y deux ensembles d'attributs d'un schéma R :

- **Dépendance fonctionnelle (DF) définie sur un schéma de relation**

$$X \rightarrow Y$$

$\Leftrightarrow \forall r$ **une instance de R** et $\forall t_1$ et t_2 2 nuplets de r ,

$$t_1.X = t_2.X \Rightarrow t_1.Y = t_2.Y$$

\Leftrightarrow "Si deux nuplets de r ont mêmes valeurs pour les attributs de X alors ils ont même valeur pour les attributs de Y . »

- Si $X \rightarrow Y$ n'est pas vérifiée par le schéma R alors $\exists r$ **une instance de R** et $\exists t_1$ et t_2 2 nuplets de r tels que $t_1.X = t_2.X$ et $t_1.Y \neq t_2.Y$
(les 2 nuplets ont la même valeur pour X mais pas pour Y)

Types de dépendances fonctionnelles

Une dépendance $X \rightarrow Y$ est :

- **triviale** : si $Y \subseteq X$
- **non triviale** : $X \cap Y \neq \emptyset$ et $Y \not\subseteq X$
- **complètement non triviale** : $X \cap Y = \emptyset$

Exemple : Soient A, B, et C trois attributs de R

$ABC \rightarrow A$ est triviale

$AC \rightarrow AB$ est non triviale

$AB \rightarrow C$ est complètement non triviale

Exercice

Exemple d'instance de relation :

A	B	C	D
a ₁	b ₁	c ₁	d ₁
a ₁	b ₁	c ₂	d ₁
a ₂	b ₂	c ₂	d ₂
a ₃	b ₂	c ₂	d ₂

Quelles DF pourrait vérifier l'instance ci-dessus ?

Attention: une dépendance fonctionnelle se définit sur un schéma et est donc vérifiée pour toutes les instances associées à un schéma.

Règles

Axiomes de Armstrong :

- **Réflexivité** : si $Y \subseteq X$ alors $X \rightarrow Y$
- **Augmentation** : Si $X \rightarrow Y$ alors $\forall Z \ XZ \rightarrow YZ$
- **Transitivité** :
Si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$

On déduit :

- **Union** : $\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$
- **Pseudo-transitivité** :
 $\{X \rightarrow Y, WY \rightarrow Z\} \models \{XW \rightarrow Z\}$
- **Décomposition** :
Si $X \rightarrow Y$ et $Z \subseteq Y$ alors $X \rightarrow Z$

Fermeture

- **Fermeture d'une famille de dépendances fonctionnelles**

$$\mathbf{F}^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$$

- **Fermeture d'un ensemble d'attributs X par rapport à une famille de dépendances fonctionnelles F**

$$[\mathbf{X}]^+ = \{A \mid F \models X \rightarrow A\}$$

- **Lemme**

La dépendance fonctionnelle $X \rightarrow Y$ peut être déduite des axiomes d'Amstrong si $Y \subseteq [X]^+$

Fermeture d'ensemble d'attributs

Algorithme de calcul de $[X]^+$:

1. $X_0 = X$
2. $X_{i+1} = X_i \cup \{Z / Y \rightarrow Z \text{ avec } Y \subseteq X_i\}$
3. On répète l'étape 2 tant que $X_{i+1} \neq X_i$

A la fin $[X]^+ = X_i$

Exemple : $F = \{ AB \rightarrow C ; BC \rightarrow AD ; D \rightarrow E ; CG \rightarrow B \}$

Calculer $[AB]^+$

Equivalence et Couverture

- Deux familles de dépendances fonctionnelles F et G sont **équivalentes** si $F^+ = G^+$
- Si $F^+ \subset G^+$ alors G est **une couverture** de F
- **Une famille de dépendances fonctionnelles F est minimale si :**
 1. En partie droite de toute dépendance de F , il n'y a qu'un seul attribut
 2. Il n'y a pas de dépendance fonctionnelle $X \rightarrow A$ dans F telle que $(F \setminus \{X \rightarrow A\})$ soit équivalente à F
 3. Il n'y a pas de dépendance fonctionnelle $X \rightarrow A$ et $Z \subset X$ tels que $(F \setminus \{X \rightarrow A\}) \cup \{Z \rightarrow A\}$ soit équivalente à F

Attention: pas d'unicité des couvertures minimales

Famille minimale équivalente

Algorithme pour trouver une famille minimale équivalente à F :

1. Mettre toutes les DF de F sous forme canonique (un seul attribut à droite) – *en appliquant la décomposition*
2. Partir de $F_0 = F$ avec les DF sous forme canonique – équivalent à F et vérifiant la propriété 1 d'une famille minimale

Calculer $F_{i+1} = F_i - \{X_i \rightarrow A / X_i \rightarrow A \in F_i\}$
 avec F_{i+1} équivalente à F_i
 (*on peut retrouver $X_i \rightarrow A$ à partir des DF de F_{i+1} en appliquant la transitivité*)

On s'arrête quand $F_{i+1} = F_i$ - **on obtient $F' = F_i$ équivalente à F et vérifiant la propriété 1 et 2 d'une famille minimale**
3. Partir de $F'_0 = F'$

Calculer $F'_{i+1} = F'_i - \{X_j \rightarrow A / X_j \rightarrow A \in F'_i\} \cup \{Y_j \rightarrow A / Y_j \subseteq X_j\}$
 avec F'_{i+1} équivalente à F'_i
 (*on diminue la partie gauche des DF de F' quand cela est possible*)

On s'arrête quand $F'_{i+1} = F'_i$ (refaire éventuellement l'étape 2)

On obtient $F'' = F'_i$ **équivalente à F et minimale**

Exercices sur les familles équivalentes

Les 2 familles sont-elles équivalentes ?

$$F = \{ A \rightarrow BC ; B \rightarrow C \}$$

$$G = \{ A \rightarrow B ; B \rightarrow C \}$$

Trouver une famille minimale équivalente à :

1. $F = \{ A \rightarrow B ; A \rightarrow C ; B \rightarrow A ; B \rightarrow C ; C \rightarrow A ; C \rightarrow B \}$
2. $F = \{ A \rightarrow BC ; B \rightarrow C ; A \rightarrow B ; AB \rightarrow C \}$
3. $F = \{ AB \rightarrow C ; A \rightarrow B \}$

Clé

Soit $R(A_1, A_2, \dots, A_n)$ et F une famille de DF associée à R

- Un sous-ensemble K de $\{A_1, A_2, \dots, A_n\}$ est une **clé minimale** de R si et seulement si :
 - ① La dépendance fonctionnelle $K \rightarrow A_1, A_2, \dots, A_n \in F^+$ ou $[K]^+ = \{A_1, A_2, \dots, A_n\}$
 - ② $\forall X \subset K$, on a pas $X \rightarrow A_1, A_2, \dots, A_n$
- Si K n'est pas un ensemble minimal alors K est une **surclé**
- Les dépendances fonctionnelles permettent de déduire les clés minimales des relations

Clé minimale

Soit F une famille miniamle de DF associée à R :

- Tout attribut n 'apparaissant qu'à droite des DF de F n 'appartient à aucune clé minimale de R
- Tout attribut n 'apparaissant qu'à gauche des DF de F appartient à toutes les clés minimales de R
- Tout attribut n 'apparaissant dans aucune DF de F appartient à toutes les clés minimales de R
- On ne peut rien déduire des attributs apparaissant à droite de certaines DF et à gauche d'autres

Exercices sur les clés minimales

Trouver les clés minimales des relations suivantes :

1. $R(A,B,C)$ et $F = \{ A \rightarrow B ; A \rightarrow C \}$
2. $R(A,B,C,D)$ et $F = \{ A \rightarrow B ; A \rightarrow C \}$
3. $R(A,B,C,D)$ et $F = \{ A \rightarrow B ; B \rightarrow A ; B \rightarrow C \}$
4. $R(A,B,C,D,E,G,H)$ et $F = \{ AB \rightarrow C ; C \rightarrow DE ; D \rightarrow AG ; G \rightarrow E \}$
 - a) Calculer $[ABH]^+$, $[BDH]^+$, $[BCH]^+$ et $[BGH]^+$
 - b) En déduire les clés minimales de R

Chap IX - Décomposition de schéma

Exemple :

R(Fournisseur, Adresse, Produit, Prix)

F = {Fournisseur → Adresse ; Fournisseur, Produit → Prix}

- Que signifie chaque DF ?
- Quelle(s) est(sont) la(les) clé(s) minimale(s) de *R* ?
- Cette relation peut-elle contenir de la redondance d'information ?
- Peut-on insérer un fournisseur dont on ne connaît que l'adresse et pas encore le catalogue de produit ?
- Peut-on modifier l'adresse d'un fournisseur ?
- Peut-on supprimer l'unique produit vendu par un fournisseur sans perdre son adresse ?

Décomposition de schéma

- **Décomposition sans perte d'information (*Lossless jointure*)**

La décomposition de R en R_1, R_2, \dots, R_n est sans perte d'information si et seulement si $\forall r$, instance de R :

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \dots \bowtie \Pi_{R_n}(r)$$

- **Décomposition sans perte de dépendances**

La décomposition de R , munie d'une famille de dépendances F , en R_1, R_2, \dots, R_n est sans perte de dépendance si et seulement si :

$$F^+ = (F_{R_1}^+ \cup F_{R_2}^+ \cup \dots \cup F_{R_n}^+)^+ \text{ avec } F_{R_i} = \{X \rightarrow Y, XY \subseteq R_i$$

et $X \rightarrow Y \in F^+\}$

Rappel jointure naturelle

Jointure naturelle :

```
select * from enseignant natural join departement;
```

```
select * from enseignant , departement where
enseignant.departement_id=departement.departement_id;
```

```
select * from enseignant inner join departement on
enseignant.departement_id=departement.departement_id;
```

Query #1 Execution time: 1ms

departement_id	enseignant_id	nom	prenom	grade	telephone	fax	email	nom_departement
1	1	MANOUVRIER	Maude	MCF	4185	4091	maude.manouvrier@dauphine.fr	MIDO
1	2	BELHAJJAME	Khalid	MCF			khalid.belhajjame[at]dauphine.fr	MIDO
1	3	NEGRE	Elsa	MCF			elsa.negre[at]dauphine.fr	MIDO

Query #2 Execution time: 1ms

enseignant_id	departement_id	nom	prenom	grade	telephone	fax	email	departement_id	nom_departement
1	1	MANOUVRIER	Maude	MCF	4185	4091	maude.manouvrier@dauphine.fr	1	MIDO
2	1	BELHAJJAME	Khalid	MCF			khalid.belhajjame[at]dauphine.fr	1	MIDO
3	1	NEGRE	Elsa	MCF			elsa.negre[at]dauphine.fr	1	MIDO

Décomposition \neg SPI

Exemple : $R(A,B,C)$ décomposée en $R_1(A,B)$ et $R_2(B,C)$

Une instance r de R

$r_1 = \Pi_{R_1}(r)$

$r_2 = \Pi_{R_2}(r)$

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

A	B
a_1	b_1
a_2	b_1

B	C
b_1	c_1
b_1	c_2

$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_2

Décomposition \neg SPI car :

$\exists r$, instance de R : $r \neq \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

Décomposition SPI

- $r \subseteq \prod_{i=1}^n R_i (r) ?$
 - $r \supseteq \prod_{i=1}^n R_i (r) ?$
- } $\forall \mathbf{r}$

Soit $t (a_1, a_2, \dots, a_n)$, un nuplet générique issu de $\prod_{i=1}^n R_i (r)$

t est construit à partir de $t_1 \in R_1, t_2 \in R_2, \dots, t_n \in R_n$

avec $t_1 \in R_1$

$t_2 \in R_2$

Est - ce - que $t \in r ?$

...

$t_n \in R_n$

Exemples de décomposition SPI (1/2)

Exemple :

$R(\text{Fournisseur}, \text{Adresse}, \text{Produit}, \text{Prix})$

$F = \{\text{Fournisseur} \rightarrow \text{Adresse} ; \text{Fournisseur}, \text{Produit} \rightarrow \text{Prix}\}$

R est décomposée en :

$R_1(\text{Fournisseur}, \text{Adresse})$ et $R_2(\text{Fournisseur}, \text{Produit}, \text{Prix})$

La décomposition est-elle SPI ? A-t-on $\forall r, \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \subseteq r$?

$\forall r$, instance de R, soit $t = (f, a, po, pi)$ un **nuplet générique de $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$** :

t est construit en faisant la jointure de 2 nuplets : $t_1 \in \Pi_{R_1}(r)$ et $t_2 \in \Pi_{R_2}(r)$

Donc il y a, dans r ,
au moins 2 nuplets
tels que :

<i>Fournisseur</i>	<i>Adresse</i>	<i>Produit</i>	<i>Prix</i>
f	a	x_{13}	x_{14}
f	x_{22}	po	pi

Exemples de décomposition SPI (2/2)

Exemple : $F = \{\text{Fournisseur} \rightarrow \text{Adresse} ; \text{Fournisseur}, \text{Produit} \rightarrow \text{Prix}\}$

<i>Fournisseur</i>	<i>Adresse</i>	<i>Produit</i>	<i>Prix</i>
<i>f</i>	<i>a</i>	<i>x₁₃</i>	<i>x₁₄</i>
<i>f</i>	<i>x₂₂</i> <i>a</i>	<i>po</i>	<i>pi</i>

Donc $t \in r$, donc $\forall \mathbf{r}, \Pi_{R_1}(\mathbf{r}) \cap \Pi_{R_2}(\mathbf{r}) \subseteq \mathbf{r}$

Et comme on a toujours $\forall \mathbf{r}, \mathbf{r} \subseteq \Pi_{R_1}(\mathbf{r}) \cap \Pi_{R_2}(\mathbf{r})$

Par conséquent : $\forall \mathbf{r}, \mathbf{r} = \Pi_{R_1}(\mathbf{r}) \cap \Pi_{R_2}(\mathbf{r})$

La décomposition est donc SPI

Exercice : $R(A,B,C,D,E)$ et $F = \{A \rightarrow C; B \rightarrow C; C \rightarrow D; DE \rightarrow C; CE \rightarrow A\}$

R est décomposée en $R_1(A,D)$, $R_2(A,B)$, $R_3(B,E)$, $R_4(C,D,E)$ et $R_5(A,E)$

La décomposition est-elle SPI ?

Montrer qu'une décomposition est non SPI (1/2)

Attention : ne pas réussir à montrer que $\forall r, t \in r$ ne permet pas de démontrer que c'est \neg SPI.

Exemple : $R(A, B, C)$ et $F = \{A \rightarrow B\}$, décomposée en $R_1(A, B)$ et $R_2(B, C)$

$\forall r$, instance de R , soit $t = (a, b, c)$ un nuplet générique de $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$:

A	B	C
a	b	x_{13}
x_{21}	b	c

$A \rightarrow B$ ne permet pas de déduire les inconnues

Il faut un contre-exemple, i.e. trouver une instance r telle que

$$r \neq \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \dots \bowtie \Pi_{R_n}(r)$$

En utilisant le tableau précédent en remplaçant les variables et les inconnues par des valeurs

Montrer qu'une décomposition est non SPI (1/2)

Soit r une instance de R :

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

La variable a prend la valeur a_1 , la variable b prend la valeur b_1 et la variable c prend la valeur c_1 . L'inconnue x_{13} prend la valeur a_2 et x_{21} prend la valeur c_2

r_1 :

A	B
a_1	b_1
a_2	b_1

r_2 :

B	C
b_1	c_1
b_1	c_2

$r_1 \bowtie r_2$:

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2
a_1	b_1	c_2
a_2	b_1	c_1

Décomposition \neg SPI car :

$\exists r$, instance de R : $r \neq \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

Décomposition SPD

- Association d'une famille F_i à chaque sous-relation R_i
- Calcul des F_i à partir de F^+ :

$$\forall X \rightarrow Y \in F^+ \text{ et } XY \subseteq R_i \Rightarrow X \rightarrow Y \in F_i$$

- Perte de dépendances \Rightarrow

$$\exists X \rightarrow Y \in F \text{ tq } X \rightarrow Y \notin [X]_{\cup F_i}^+$$

$$[X]_{\cup F_i}^+ \text{ calculé itérativement par } X \cup [(X \cap R_i)^+ \cap R_j]$$

Exemples de décompositions SPD

Exemple 1 : $R(\text{Rue}, CP, \text{Ville})$ et $F = \{CP \rightarrow V ; R, V \rightarrow CP\}$

R est décomposée en $R_1(CP, V)$ et $R_2(CP, R)$

$F_1 = \{CP \rightarrow V\}$ et $F_2 = \emptyset$

La dépendance $R, V \rightarrow CP$ est perdue.

La décomposition est donc \neg SPD.

Exemple 2 : $R(\text{Nom}, \text{Bureau}, \text{Telephone})$

et $F = \{N \rightarrow T ; T \rightarrow B ; B \rightarrow N\}$

R est décomposée en $R_1(N, T)$ et $R_2(T, B)$

$F_1 = \{N \rightarrow T ; T \rightarrow N\}$ et $F_2 = \{T \rightarrow B ; B \rightarrow T\}$

$\in F$

$\in F^+$

La dépendance $B \rightarrow N$ n'est pas perdue car $B \rightarrow T ; T \rightarrow N \Rightarrow B \rightarrow N$

La décomposition est donc SPD car $F^+ = (F^+_{R_1} \cup F^+_{R_2})^+$

Exercice : décomposition SPD

Exercice :

$R(A,B,C,D)$ et $F = \{A \rightarrow B ; B \rightarrow C ; C \rightarrow D ; D \rightarrow A\}$

R est décomposée en $R_1(A,B)$, $R_2(B,C)$ et $R_3(C,D)$

- Calculer F_1 , F_2 et F_3
- La décomposition est-elle SPD ?

Chap X - Formes normales

Forme normale de Boyce-Codd (BCNF)

- Un schéma de relation est **BCNF** si et seulement si :

DF élémentaires telles que une clé détermine un attribut

- Un schéma de relation est BCNF ssi :

$\forall X \rightarrow A$, DF non triviale associée à R :

X est une (sur)clé **et** A n'est pas un attribut de clé

- La forme normale BCNF évite la redondance d'information
- Toute relation a 2 attributs est BCNF

BCNF : Exercices

Exemple 1 :

$R(\text{Cru}, \text{Pays}, \text{Région}, \text{Qualité})$

avec $F = \{ \text{Cru}, \text{Pays} \rightarrow \text{Région} ; \text{Cru}, \text{Pays} \rightarrow \text{Qualité} ; \text{Région} \rightarrow \text{Pays} \}$

- Quelles sont les clés minimales de R ?
- R est-elle BCNF ?

Exemple 2 : $R(A, B, C)$ avec $F = \{A \rightarrow B ; B \rightarrow A ; B \rightarrow C \}$

- Quelles sont les clés minimales de R ?
- R est-elle BCNF ?

Algorithme de décomposition BCNF

- **Entrée** : R —BCNF et F une famille **minimale** de DF associée à R
- **Sortie** : R_1, R_2, \dots, R_n BCNF issues de la décomposition de R

1. On prend $X \rightarrow A$ une DF de F et on crée :

$R_1(X,A)$ associée à $F_1 = \{X \rightarrow A, \dots\}$ BCNF

Mettre dans F_1 les DF qui portent sur X et A

$R_2(R - \{A\})$

2. Si R_2 est BCNF , l'algo est terminé sinon on décompose R_2 en reprenant l'étape 1

- La décomposition de schéma BCNF est toujours sans perte d'information mais pas toujours sans perte de dépendances

Exercices d'application de l'algorithme de décomposition BCNF (1/2)

Exemple 1 : $R(A, B, C, D)$ avec $F = \{D \rightarrow A ; B \rightarrow C\}$ $K=BD$, \neg BCNF

$R_1(B, C)$ avec $F_1 = \{B \rightarrow C\}$ $K_1=B$, BCNF

$R_2(A, B, D)$ avec $F_2 = \{D \rightarrow A\}$ $K_2=BD$, \neg BCNF

$R_{21}(A, D)$ avec $F_{21} = \{D \rightarrow A\}$ $K_{21}=D$, BCNF

$R_{22}(B, D)$ avec $F_{22} = \emptyset$ $K_{22}=BD$, BCNF

Donc R est décomposée en 3 relations BCNF :

$R_1(B, C)$, $R_{21}(A, D)$ et $R_{22}(B, D)$

La décomposition est SPI et SPD

Exercices d'application de l'algorithme de décomposition BCNF (2/2)

Exemple 2 : $R(\text{Cru}, \text{Pays}, \text{Région}, \text{Qualité})$

avec $F = \{\text{Cru}, \text{Pays} \rightarrow \text{Région} ; \text{Cru}, \text{Pays} \rightarrow \text{Qualité} ; \text{Région} \rightarrow \text{Pays}\}$

$K = (\text{Cru}, \text{Pays})$ ou $(\text{Cru}, \text{Région}) \not\vdash \text{BCNF}$

$R_1(\text{C}, \text{P}, \text{Q})$ avec $F_1 = \{\text{C}, \text{P} \rightarrow \text{Q}\}$ $K_1 = (\text{C}, \text{P}) \vdash \text{BCNF}$

$R_2(\text{C}, \text{P}, \text{R})$ avec $F_2 = \{\text{C}, \text{P} \rightarrow \text{R} ; \text{R} \rightarrow \text{P}\}$ $K_2 = (\text{C}, \text{P})$ ou (C, R) , $\not\vdash \text{BCNF}$

$R_{21}(\text{R}, \text{P})$ avec $F_{21} = \{\text{R} \rightarrow \text{P}\}$ $K_{21} = \text{R}$, $\vdash \text{BCNF}$

$R_{22}(\text{C}, \text{R})$ avec $F_{22} = \emptyset$ $K_{22} = (\text{C}, \text{R})$, $\vdash \text{BCNF}$

Donc R est décomposée en 3 relations BCNF :

$R_1(\text{C}, \text{P}, \text{Q})$, $R_{21}(\text{R}, \text{P})$ et $R_{22}(\text{C}, \text{R})$

La décomposition est SPI mais $\not\vdash \text{SPD}$

3ème forme normale

- Un schéma de relation est **3NF** si :
La famille F minimale associée ne contient ni dépendance transitive ni dépendance partielle
- $X \rightarrow A$ est **partielle** si, $\forall K$, X est une partie de clé et A ne l'est pas
- $X \rightarrow A$ est **transitive** si $\forall K$, ni X ni A ne sont des parties de clé
- Si K est clé alors $K \rightarrow X \rightarrow A$ est une **chaîne non triviale**
- Un schéma R , munie d'une famille F de DF, est 3NF ssi :

$\forall X \rightarrow A$, DF non triviale associée à R :

X est une (sur)clé **ou** A appartient à une clé

DF partielle

- **Exemple 1 :**

$R(A,B,C,D) F = \{A \rightarrow B ; B \rightarrow A ; C \rightarrow D \}$

$K = (AC) \text{ ou } (BC)$

$C \rightarrow D$ est **partielle** car C est un morceau de toutes les clés minimales possibles

- **Exemple 2 :**

$R(A,B,C,D) F = \{A \rightarrow B ; B \rightarrow A ; \}$

$K = (ACD) \text{ ou } (BCD)$

$A \rightarrow B$ n'est **pas partielle** car A n'est pas un morceau de toutes les clés possibles

$A \rightarrow B$ est de la forme : un attribut clé à droite qd $K=(BCD)$

DF transitive

- **Exemple 3 :**

$$R(A,B,C) F = \{A \rightarrow B ; B \rightarrow C\}$$

$$K = A$$

$B \rightarrow C$ est **transitive** car B et C n'appartiennent à aucune clé minimale

- **Exemple 4 :**

$$R(A,B,C,D,E) F = \{A \rightarrow B ; B \rightarrow A ; C \rightarrow D ; D \rightarrow E\}$$

$$K = (AC) \text{ ou } (BC)$$

$D \rightarrow E$ est **transitive** car D et E n'appartiennent à aucune clé minimale

$C \rightarrow D$ est partielle car C appartient à toutes les clés minimales

Algorithme de décomposition 3NF

- **Entrée** : $R \text{ --3NF}$ et F une famille **minimale** de DF associée à R
- **Sortie** : R_1, R_2, \dots, R_n 3NF ou mieux issues de la décomposition de R

1. A chaque DF $X \rightarrow A$ de F et on crée :

$R_i(X,A)$ associée à $F_i = \{X \rightarrow A ; \dots\}$ 3NF ou BCNF

Mettre dans F_i les DF qui portent sur X et A

2. Si aucune R_i ne contient de clé minimale de R , on ajoute $R_n(K)$

- La décomposition de schéma 3NF est toujours sans perte d'information et toujours sans perte de dépendances
- On peut fusionner des relations R_i à condition que la relation issue de la fusion soit toujours au moins 3NF

Exemple d'application de l'algorithme de décomposition 3NF

Soit $R(A, B, C, D, E, G)$

avec $F = \{AB \rightarrow C ; C \rightarrow A ; D \rightarrow E ; AB \rightarrow G\}$ $K=ABD$ ou BCD , \neg 3NF

$R_1(A, B, C)$ avec $F_1 = \{AB \rightarrow C ; C \rightarrow A\}$ $K_1=AB$ ou BC , 3NF car $C \rightarrow A$ est de la forme un attribut $\in K$ quand $K=(AB)$

$R_2(D, E)$ avec $F_2 = \{D \rightarrow E\}$ $K_2=D$, BCNF

$R_3(A, B, G)$ avec $F_3 = \{AB \rightarrow G\}$ $K_3=AB$, BCNF

$R_4(A, B, D)$ ou $R_4(B, C, D)$ avec $F_4 = \emptyset$, BCNF

Donc R est décomposée en 4 relations :

$R_1(A, B, C)$, $R_2(D, E)$, $R_3(A, B, G)$ et $R_4(A, B, D)$ ou $R_4(B, C, D)$

La décomposition est SPI et SPD

On pourrait fusionner $R_1(A, B, C)$ et $R_3(A, B, G)$ en $R_{13}(A, B, C, G)$ 3NF

Objectif principaux de la décomposition de schéma

- La 3NF n'élimine pas le redondance d'information
 - Objectif principaux de la décomposition de schéma :
 - Avoir des relations BCNF (ou au minimum 3NF)
 - Être sans perte d'information
 - Être sans perte de dépendance
 - Avoir le moins de relations possibles
- ⇒ On préférera sacrifier la forme normale BCNF et avoir une 3NF SPI-SPD

1ère, 2ème et 3ème formes normales

- Un schéma de relation R est **1NF** si :
 - \forall attribut de R, il contient une **valeur atomique**
- Un schéma de relation R est **2NF** si et seulement si :
 - le schéma est en 1NF
 - **et \neg (\exists une dépendance fonctionnelle partielle)**
- Un schéma de relation R est **3NF** si :
 - le schéma est 2NF
 - **et \neg (\exists une dépendance fonctionnelle transitive)**