Licence Informatique des Organisations - 3ème année

2024-2025

Bases de données relationnelles

Maude Manouvrier

Plan du Cours

I. Introduction : Définitions et vocabulaire

II. Modèle relationnel

III. Algèbre relationnelle

IV. Calcul relationnel

V. Algèbre relationnelle étendue

VI. SQL

VII. Modélisation Entité/Association et UML et passage au relationnel

VIII. Dépendances fonctionnelles

IX. Décomposition de schéma

X. 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 - InfoSup, Dunod, 5ème Edition, 2022, ISBN : 978-2100784608

T. Connoly, C. Begg et A. Strachan, *Database Systems A Pratical Approach to Desigh, Implementation and Management*, 6ème 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, 7ème édition, 2019, ISBN: 978-0073523323 - https://www.db-book.com/

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, 7eme Ed. 2000

C.J. Date, *A Guide to SQL Standard*, Addison Wesley, 1989

R.A. El Masri et S.B. Navathe, *Fundamentals of Database Systems*, Prentice Hall, 7eme Ed. 2015

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

Base de données relationnelles :

- **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)

Systèmes 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 de SGBD relationnels : MySQL, PostgreSQL (utilisé en TP), Oracle, Microsoft SQLServer, etc.

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

Classement des SGBD

421 systems in ranking, July 2024

Rank Jul 2024	Rank Jun 2024	Rank Jul 2023	DBMS	Database Model	Score Jul 2024	Score Jun 2024	Score Jul 2023
1.	1.	1.	Oracle	Relational, Multi-model	1240.37	-3.72	-15.64
2.	2.	2.	MySQL	Relational, Multi-model	1039.46	-21.89	-110.89
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	807.65	-13.91	-113.95
4.	4.	4.	PostgreSQL	Relational, Multi-model	638.91	+2.66	+21.08
5.	5.	5.	MongoDB	Document, Multi-model	429.83	+8.75	-5.67
6.	6.	6.	Redis	Key-value, Multi-model	156.77	+0.82	-7.00
7.	↑ 8.	↑ 11.	Snowflake	Relational	136.53	+6.17	+18.84
8.	↓ 7.	8.	Elasticsearch	Search engine, Multi-model	130.82	-2.01	-8.77
9.	9.	↓ 7.	IBM Db2	Relational, Multi-model	124.40	-1.50	-15.41
10.	10.	10.	SQLite	Relational	109.95	-1.46	-20.25

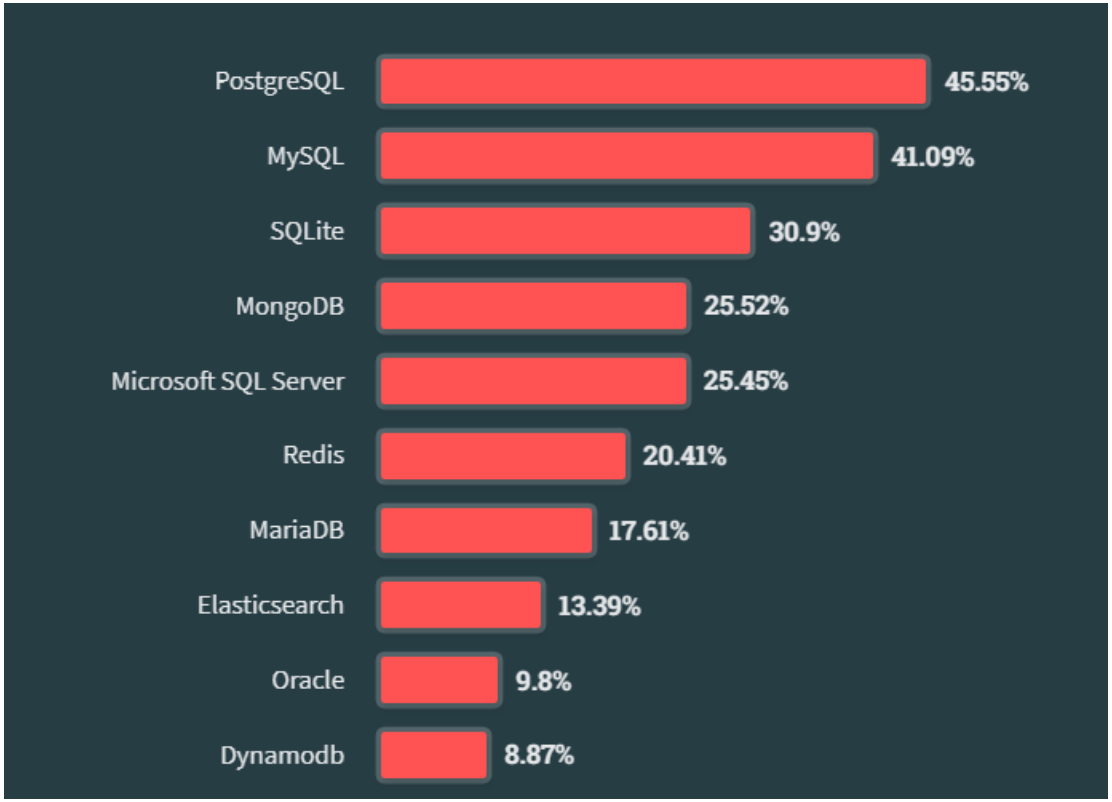
https://db-engines.com/en/ranking

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

Bases de données les plus populaires

Etude de mai 2023 *stackoverflow* basée sur les réponses de 90 000 réponses de développeurs issues de 180 pays

https://survey.stackoverflow.co/2023/#technology



Position du SQL parmi les tendances des offres d'emploi en 2023

"The Rise of SQL

It's become the second programming language everyone needs to know"

cf https://spectrum.ieee.org/the-rise-of-sql

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* >= *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 jour 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 ⇒ pas d'interrogation sans savoir où est l'information recherchée ("navigation") et sans écrire de programmes
- **1970** : papier fondateur d'Edgar 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, O_2, Orion, Objectstore, Versant, Matisse...).
- **Aujourd'hui** : **relationnel-objet, NoSQL (MongoDB, Redis, Neo4j) et NewSQL**
- **cf . L'histoire des bases de données** https://www.youtube.com/watch?v=iu8z5QtDQhY

Différents modèles de bases de données

Bases de données relationnelles (fin des années 70) – Innovation de rupture

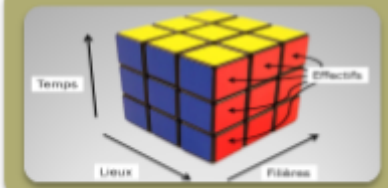
- **Outils de modélisation : E/R, ..**
- **langages de requêtes : SQL, etc.**
- **Optimisation des requêtes**
- **Indexation : Arbres B+, hachage, etc.**
- **Interfaces, outils de report**
- **Gestion des transactions, OLTP**

Bases de données avancées (années 80)

- *modèles* : relationnel étendu, objet, objet-relationnel …
- *applis* : spatial, temporel, multimédia, active, bases de connaissances, ...

Base de données et web (années 90)

- **XML et Bases de Données**
- **Web-Mining**



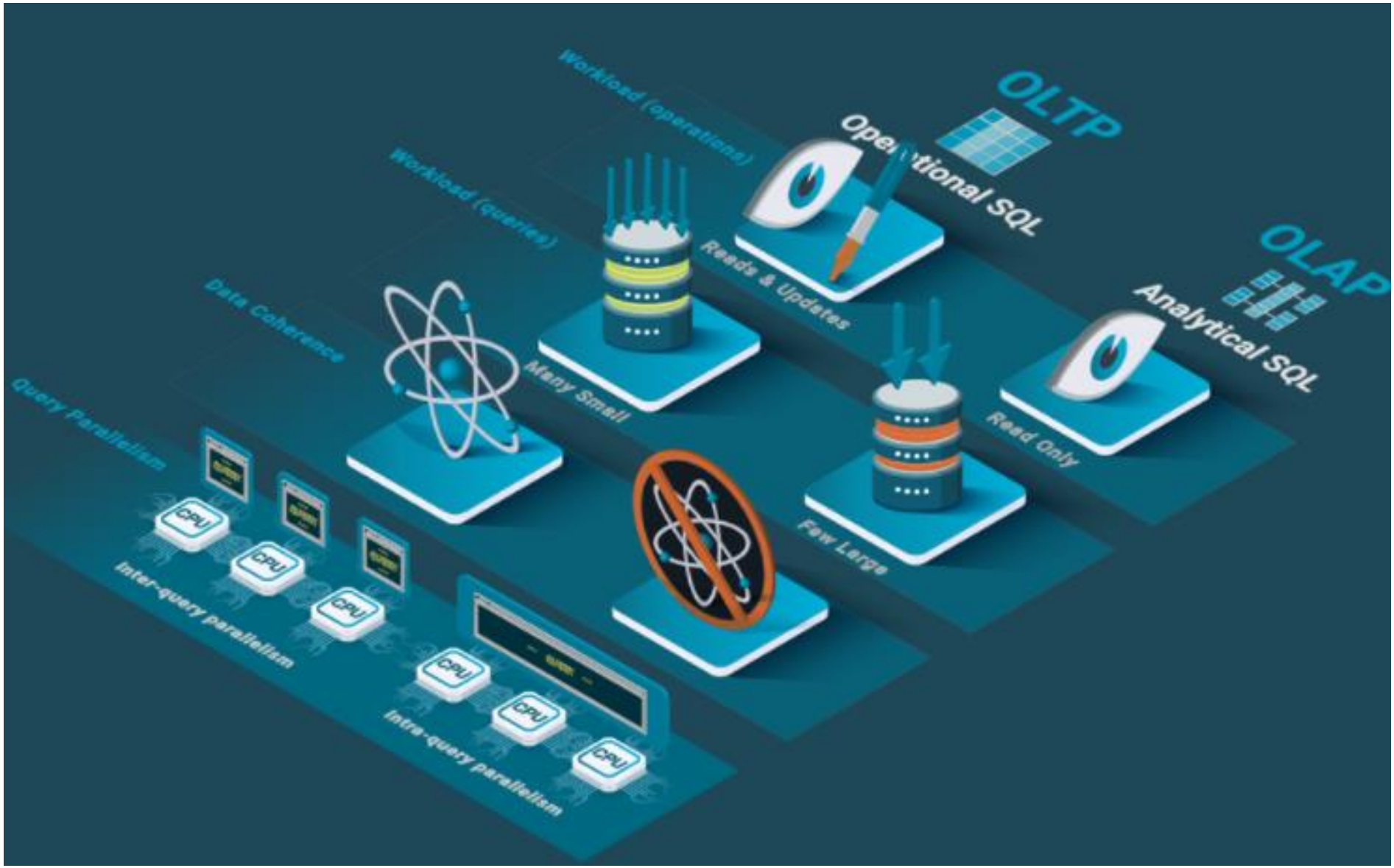
Décisionnel (années 90)

- **Entrepôts et OLAP (*OnLine Analytical Processing*)**
- **KDD et data mining**

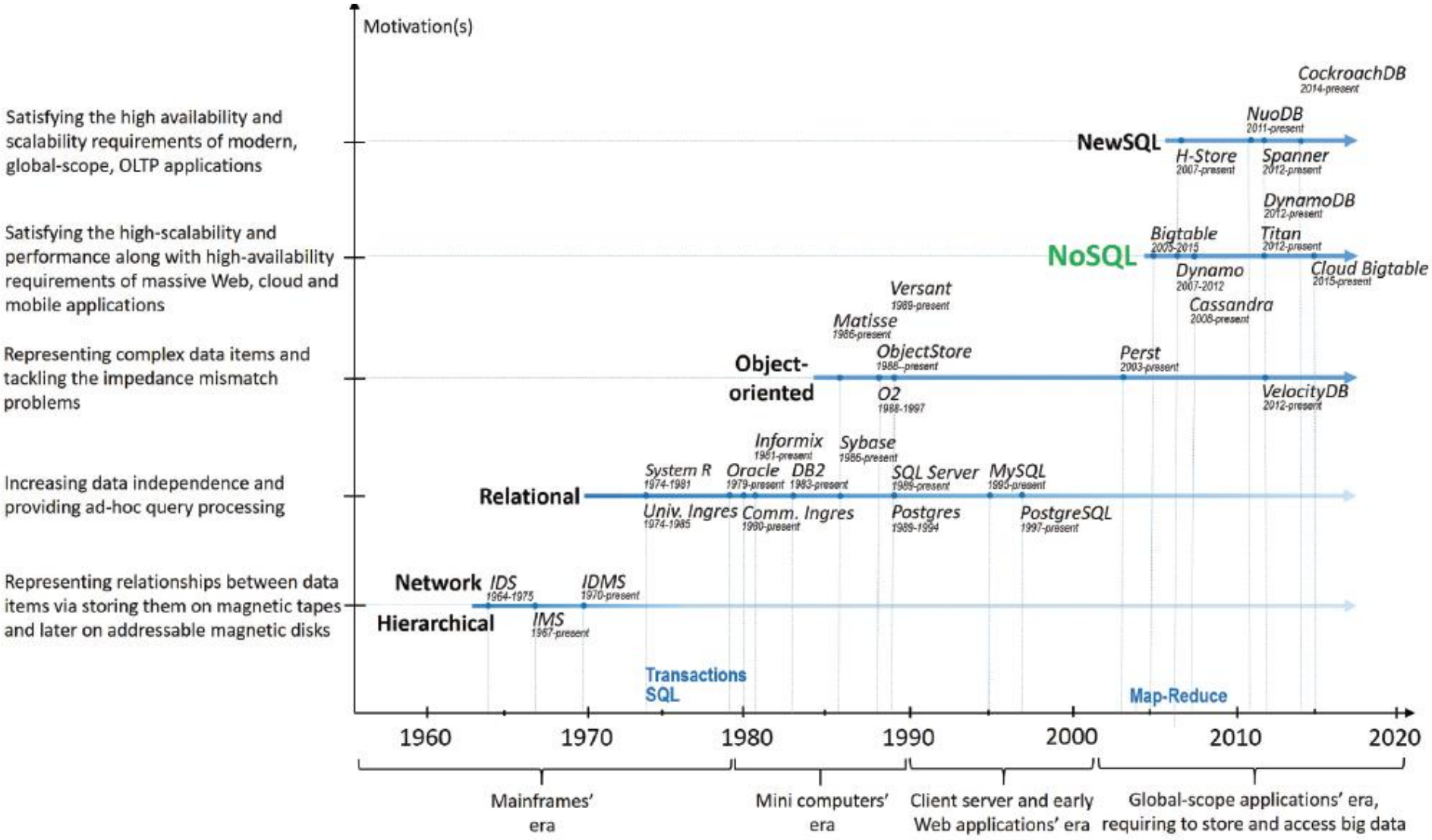
Nouvelles bases de données (fin des années 2000)

- **NoSQL**
- **In Memory**

OLAP (*On Line Analytical Processing*) vs OLTP (*On Line Transactional Processing*)



Date de naissance des différents moteurs

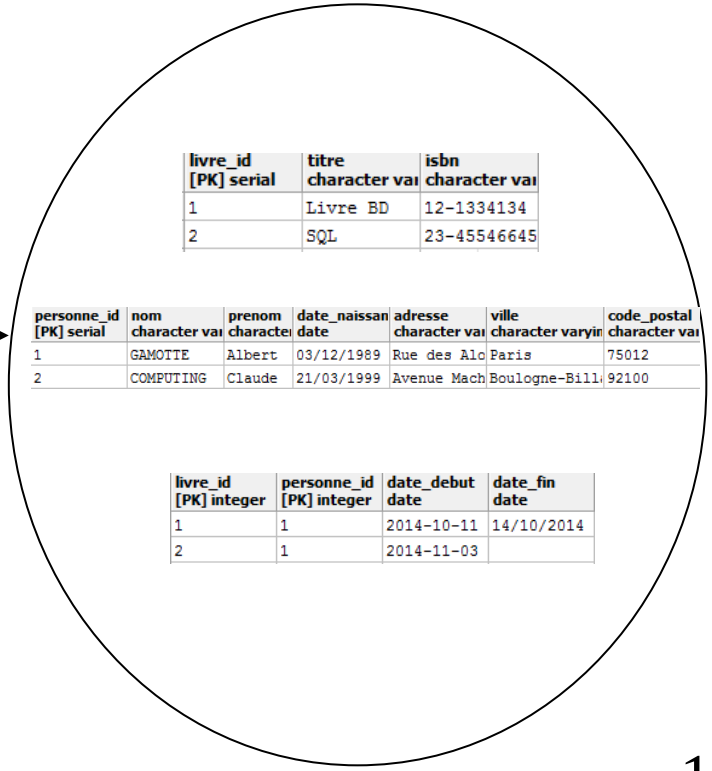


Chap II - Modèle relationnel

- **Données** : Ce que l'on stocke
- **Modèle relationnel** : Modèle permettant d'**organiser les données** en une **représentation schématique** qui autorisera son exploitation par un 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 var	isbn character var
1	Livre BD	12-1334134
2	SQL	23-45546645

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

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	

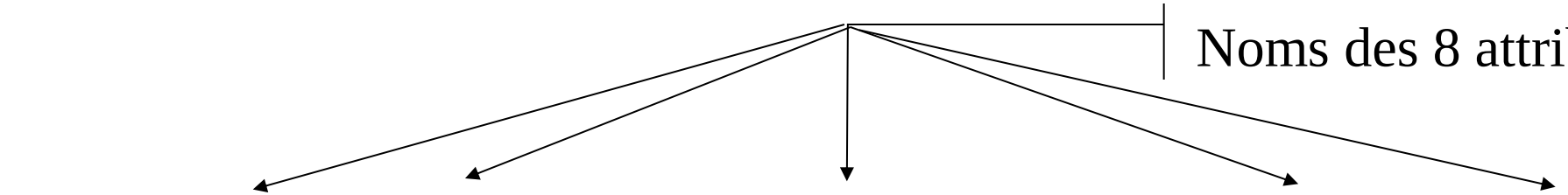
The Law of the Relational Database

If the only tool you have is a relational database,
everything looks like a table.

A Walk in Graph Databases - 2012

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 var	telephone character var	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

3 nuplets

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 primaire 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 K** 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*, *Prenom*, *Adresse*)

Clés primaires possibles :
PersonneID (clé artificielle) ou *NSS* ou (*Nom*,*Prenom*,*Adresse*) (clé métier)

Clés non minimales :
PersonneID + d'autres att. ou *NSS* + d'autres att.
ou (*Nom*,*Prenom*,*Adresse*) + d'autres att.

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

Clé primaire : *Immatriculation* (clé métier)

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 vai	isbn character vai
1	Livre BD	12-1334134
2	SQL	23-45546645

⇒ Stockage unique des informations de chaque emprunteur

personne_id [PK] serial	nom character vai	prenom characte	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

⇒ 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 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 characte	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

- 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 characte	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

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 [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 characte	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

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 vai	prenom characte	date_naissan date	adresse character vai	ville character varyin	code_postal character vai
1	GAMOTTE	Albert	03/12/1969	Rue des Alo	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	Manourvier	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	Manourvier	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	Manourvier	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

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)

<u>Étape N° 1</u> : 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)

<u>Étape N° 2</u> : 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 ∈ {"Professeur", "MCF", "ATER" …}*
- *Seuls Tel, Fax et Email peuvent prendre la valeur NULL*

Exemple de modèle relationnel (2/2)

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 var	telephone character var	fax character var	email character varying(100)
1	1	MANOUVRIER	Maude	MCF			manouvrier@lamsade.d
2	1	NEGRE	Elsa	MCF			negre@lamsade.dauphi
3	1	BELHADJJAME	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 : Enoncé est UNIQUE

Examen(ExamID, Date, Heure)

Exercice(ExoID, Enoncé)

Contenu_Exam(#ExamID, #ExoID, Position)

Modèle relationnel 2 : Enoncé est UNIQUE

Examen(ExamID, Date, Heure)

Exercice(ExoID, Enoncé, #ExamID, Position)

Modèle relationnel 3 : Enoncé est UNIQUE

Examen(ExamID, Date, Heure)

Exercice(ExoID, #ExamID, Enoncé)

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 varyir	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 z	heure_fin time without ti	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*)

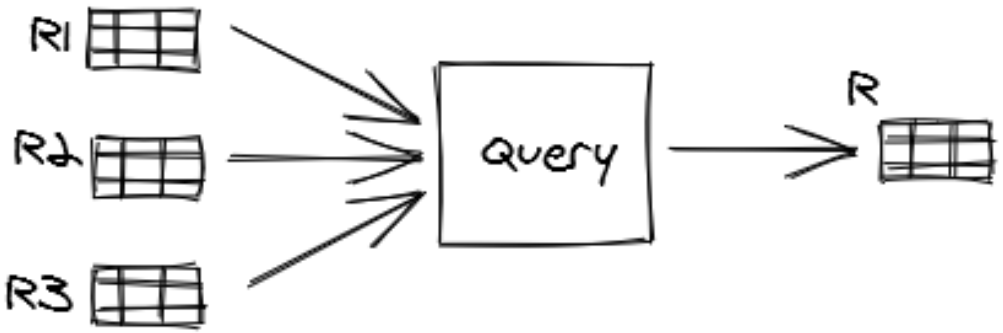
Chap III - Algèbre relationnelle

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)

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

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

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

  $\Pi_{\text{Nom,Prénom}}$(Etudiant)

  $\Pi_{\text{Nom,Prénom}}(\sigma_{(\text{Ville}=\text{`` Paris ''})}$ (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 $\Pi_{Nom,Prenom}(\sigma_{(grade='MCF')}(Enseignant))$:

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

Exemples de projection et de sélection

r :

A	B	C
a	b	c
d	a	f
c	b	d

$\pi_{B,A}(r)$:

B	A
b	a
a	d
b	c

r :

A	B	C
a	b	c
d	a	f
c	b	d

$\sigma_{B=b}(r)$:

A	B	C
a	b	c
c	b	d

Opérateurs binaires

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

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

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

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

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

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

  Département(Département_ID, Nom_Département)

  Schéma du Produit Cartésien de Enseignant × Departement :

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

Exemple d'union

$\Pi_{Nom,Prénom}$(Etudiant) :

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

$\Pi_{Nom,Prénom}$(Enseignant) :

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

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

$\Pi_{Nom,Prénom}$(Etudiant) :

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

$\Pi_{Nom,Prénom}$(Enseignant) :

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

$\Pi_{Nom,Prénom}$(Etudiant) - $\Pi_{Nom,Prénom}$(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× Département***

Autres opérateurs binaires

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

- **Renommage :**

  $\Pi_{A',B',\ldots}(r_{A \to A', B \to B', \ldots})$

  ou $\rho_{A \to A', B \to B', \ldots}(r)$

- **Intersection :**

  $r \cap s = r - (r - s)$

- **Théta-jointure :**

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

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

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

Exemple de renommage et d'intersection

$\Pi_{Last\_Name,First\_Name}(Enseignant\ _{Nom \rightarrow Last\_Name,\ Prénom \rightarrow First\_Name})$:

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

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

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

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

r :

A	B	C
a	b	c
d	a	f
c	b	d

s :

A	B	C
b	g	a
d	a	f

$r \cup s$:

A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

$r - s$:

A	B	C
a	b	c
c	b	d

$r \cap s$:

A	B	C
d	a	f

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 :

A	B	C
a	b	c
d	a	f
c	b	d

s :

B	C	A
g	a	b
a	f	d

$r \cup s$

A	B	C
a	b	c
d	a	f
c	b	d
g	a	b
a	f	d

$r - s = r$

A	B	C
a	b	c
d	a	f
c	b	d

$r \cap s$:

A	B	C

Exemple de produit cartésien et de jointure naturelle

r :

A	B
a	b
d	e

s :

C	D
c	d
f	g

$r \times s$:

A	B	C	D
a	b	c	d
a	b	f	g
d	e	c	d
d	e	f	g

r :

A	B
a	b
d	e

s :

B	C
b	f
g	h

$r \bowtie s$:

A	B	B	C
a	b	b	f

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 ∞ *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** $\infty_{Dept= Dept\_ID}$ **Département***

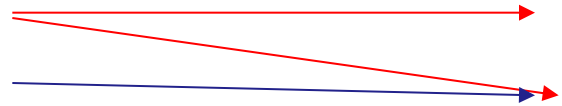
Exemple de Théta-jointure

r :

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

s :

D	E
3	1
6	2



$r \underset{B<D}{\bowtie} s$:

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

Division : définition formelle

Requête qui contient le terme « pour tous »

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

la relation r ÷ s a pour schéma R - S

un nuplet *t* appartient à **r ÷ 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 $\Pi_S(t_r) = \Pi_S(t_s)$
- $t_r(R\text{-}S) = t$ ou $\Pi_{R\text{-}S}(t_r) = t$

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

Division

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID,ISBN, DateEmprunt,EtudiantID)

Etudiant(EtudiantID,Nom, Prenom)

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

$$\Pi_{Titre,EtudiantID}(Livre \bowtie Emprunt) \div \Pi_{EtudiantID}(Etudiant)$$

Les titres des livres et les id des étudiants emprunteurs

Titre	IDEtudiant
Vives les Bases de données	1
Le SQL c'est facile!	1
Vives les Bases de données	3
Le SQL c'est facile!	2
Le SQL c'est facile!	3

Les ID de **tous** les étudiants de la base de données

IDEtudiant
1
2
3

=> Renvoie une table composée d'un attribut *Titre*, contenant les titres des livres, qui dans le résultat de la jointure, sont associés aux id de tous les étudiants

Titre
Le SQL c'est facile!

Exemple de division (1/2)

r :

A	B	C	D
a	b	c	d
a	b	e	f
g	h	c	d
i	j	k	ℓ

s :

C	D
c	d
e	f

$r \div s$:

A	B
a	b

Exemple de division (2/2)

r :

A	B	C	D
a	b	c	d
a	b	e	f
g	h	c	d
i	j	k	ℓ

s :

C	D
c	d
e	f

$r \div s$:

A	B
a	b

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 :

$Convocation \div \Pi_{MID}(Match) =$

JID
2

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

$\Pi_{Nom,Prenom,MID}(Joueur \infty Convocation) \div \Pi_{MID}(Match) =$

Nom	Prenom
Debece	Gilles

3ème 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* :

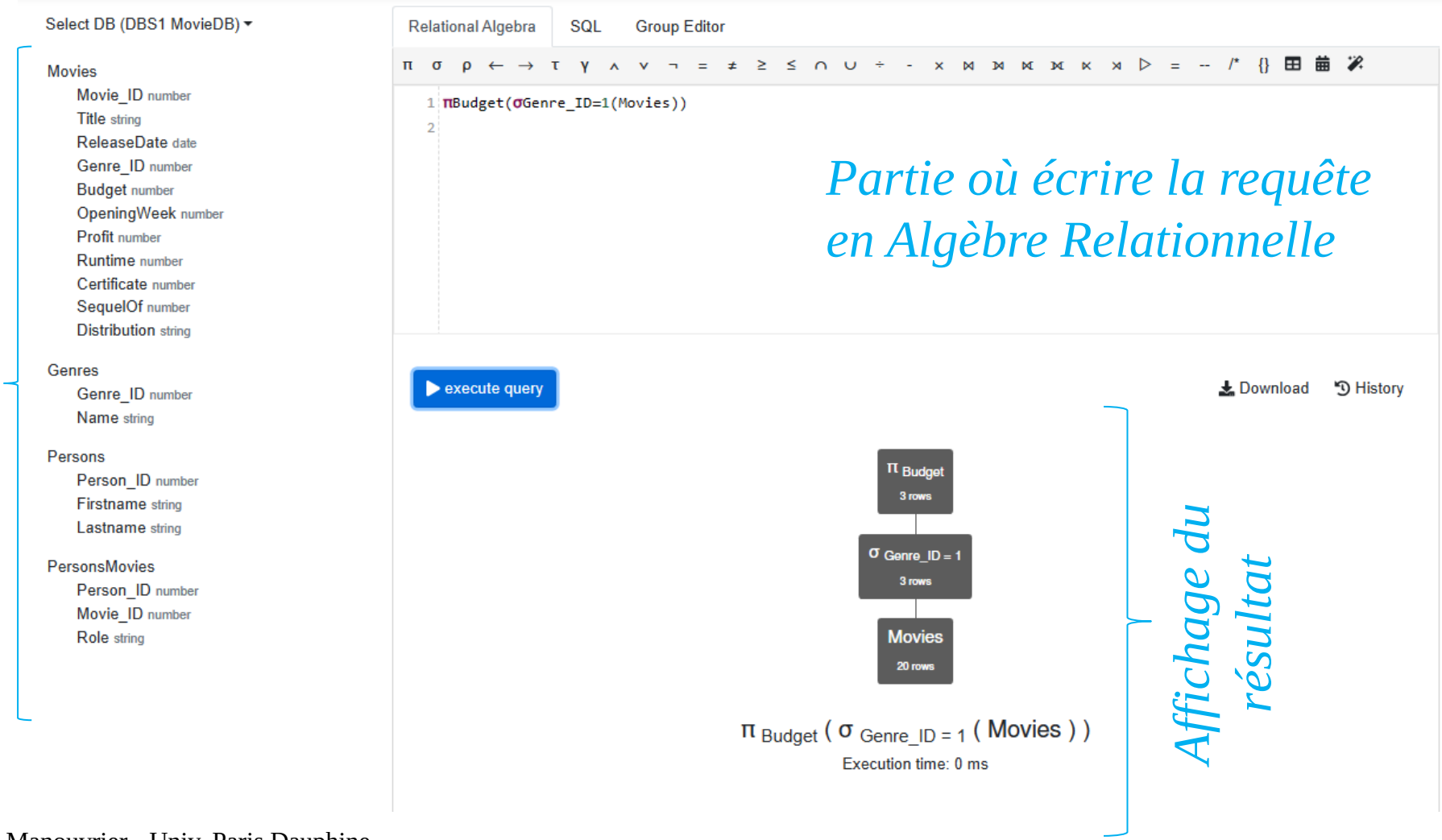
Ligne	etudiant_id (int4)	enseignement_id (int4)	departement_id (int4)	date_inscription (date)
1	1	1	1	2004-02-25
2	1	2	1	2004-07-22
3	3	2	1	2004-07-22
4	5	2	1	2004-07-22
5	4	2	1	2004-07-22
6	1	3	1	2004-07-22
7	1	1	5	2004-07-22
8	2	1	5	2004-07-22

$\Pi_{Etudiant\_ID,\ Enseignement\_ID,\ Departement\_ID}$ (*Inscription*) ÷ $\Pi_{Enseignement\_ID,\ Departement\_ID}$ (*Enseignement*) :

Ligne	etudiant_id (int4)
1	1

Pratiquer l'algèbre relationnelle en ligne

Utilisation du site en ligne :
https://dbis-uibk.github.io/relax/calc/gist/d37f667154aec34f5c4954723ae01db9/DBS1_MovieDB/0



Exercice 1

Sur la base des films :
https://dbis-uibk.github.io/relax/calc/gist/d37f667154aec34f5c4954723ae01db9/DBS1_MovieDB/0

Tester les requêtes suivantes : *vous pouvez faire un copier-coller sur le site*

1. πBudget(σGenre_ID=1(Movies))
2. σGenre_ID=1(πBudget(Movies))
3. Movies⋈Genres
4. Movies⋉Genres
5. Movies⋊Genres
6. Movies⋋Genres
7. πName(σMovie_ID=NULL(Movies⋉Genres))
8. πName(Genres) - πName(Movies⋈Genres)
9. πName(Genres▷Movies)
10. Persons∩PersonsMovies
11. πPerson_ID(Persons)∩πPerson_ID(PersonsMovies)

Certaines requêtes vont générer une erreur, essayez de comprendre pourquoi.
Essayez de traduire chaque requête en français.

Exercice 2

Sur la base des pizzas :
https://dbis-uibk.github.io/relax/calc/gist/7d1871f79a8bcb4788de/uibk_db_pizza/0

Ecrire en Algèbre, les requêtes suivantes :

1. Qui fréquente les pizzeria 'Pizza Hut' OU 'Dominos' (l'une ou l'autre ou les 2) ?
2. Qui fréquente les pizzeria 'Pizza Hut' ET 'Dominos' (les 2 pizzerias) ?
3. Qui fréquente les pizzeria 'Pizza Hut' MAIS PAS 'Dominos' ?
4. Qui fréquente au moins 2 pizzerias différentes ?
5. Qui fréquente une seule pizzeria ?
6. Quelles pizzas sont dégustées par des femmes ?
7. Quelles pizzerias servent des pizzas dégustées par des femmes et à quel prix ?
8. Quelles pizzerias servent des pizzas NON dégustées par des femmes et à quel prix ? (*NB: les pizzas 'sausage' ne sont pas dégustées par les femmes*)
9. Qui mange toutes les pizzas possibles (présentes dans la base) ? (*NB: il s'agit de 'Dan'*)
10. Quelles pizzas sont servies dans toutes les pizzeria (apparaissant dans la base de données) ? (*NB: il s'agit de la pizza 'cheese'*)

Contraintes et DF

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

$$\Pi_{Departement\_ID}(Enseignant) \subseteq \Pi_{Departement\_ID}(Departement)$$

$$\Pi_{Departement\_ID}(Enseignant) - \Pi_{Departement\_ID}(Departement) = \varnothing$$

- **Expressions des dépendances fonctionnelles :**

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

$\Leftrightarrow$ si on connait X on peut en déduire Y

$\Leftrightarrow$ $\forall r$, 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 :**

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

  - **`r(t)` indiquant que `t` est un nuplet d'une instance `r`**
  - **$t.att_1$ op $valeur_1$ et op un opérateur de comparaison**
  - **$t.att_1$ op $s.att_2$ ...**

Exemples de requêtes en calcul

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

$\Leftrightarrow$

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

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

$\Leftrightarrow$

```
{t.Nom, t.Prenom / Etudiant(t) ∧ (t.Ville='Paris') }
```

Quantificateurs

- `∃t (Q(t))` : **il existe un nuplet `t` dans la base qui vérifie le prédicat `Q(t)`**
- `∀t (Q(t))` : **pour tous les nuplets `t`, le prédicat `Q(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**

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

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

  $\{t \mid \neg\ Etudiant(t)\}$

Expression des opérateurs algébriques (1/2)

- $\sigma_{\Theta}(r) \Leftrightarrow$ `{t / r(t) ∧ Θ(t) }`

- $\Pi_{A1,A2,\ldots,An}(r) \Leftrightarrow$ `{t.A1, t.A2, … , t.An / r(t) }`
  avec `A1, A2 , An` … des attributs de R

Soient `R(A1, …,An)` et `S(B1, …Bn)` schéma-compatibles

- $r \cup s \Leftrightarrow$ `{t / ∃u,v r(u) ∧ s(v) ∧ [(t.A1=u.A1) … (t.An=u.An) ] ∨ [(t.A1=v.B1) … (t.An=v.Bn) ]}`

- $r - s \Leftrightarrow$ `{t / r(t) ∧ ¬[ ∃u s(u) ∧ (t.A1=u.B1) ∧ … ∧ (t.An=u.Bn) ] }`

Exemple d'écriture d'union et de différence

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

$\Leftrightarrow$

$$\{t.Nom,\ t.Prenom\ /\ \exists u,v\ Etudiant(u) \wedge Enseignant(v) \wedge [(t.Nom=u.Nom) \wedge (t.Prenom=u.Prenom)] \vee [(t.Nom=v.Nom) \wedge (t.Prenom=v.Prenom)]\}$$

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

$$\Leftrightarrow \{t.Nom,\ t.Prenom\ /\ Etudiant\ (t) \wedge \neg[\ \exists u\ Enseignant(u) \wedge (t.Nom = u.Nom) \wedge (t.Prenom = u.Prenom)\ ]\ \}$$

Expression des opérateurs algébriques (2/2)

Soient R(A1, …,An) et S(B1, …Bp)

- $r \times S \Leftrightarrow$ {t / ∃u,v r(u) ∧ s(v)∧ (t.A1=u.A1) ∧ …
  (t.An=u.An) ∧ (t.B1=v.B1) ∧ … ∧ (t.Bp=v.Bp) }

Soient R(A1, …,An,C1,…,Cm) et S(B1, …Bp,C1,…,Cm)

- $r \infty S \Leftrightarrow$ {t / ∃u,v r(u) ∧ s(v)∧ (t.A1=u.A1) ∧ …
  (t.An=u.An) ∧ (t.B1=v.B1) ∧ … ∧ (t.Bp=v.Bp) ∧ (t.C1=u.C1)
  … ∧ (t.Cm=u.Cm) ∧ (u.C1=v.C1) ∧ … ∧ (u.Cm=v.Cm) }

Exemples d'écriture de produit cartésien et de jointure

Enseignant × Departement

$\Leftrightarrow$ {t / ∃e,d Enseignant(e) ∧ Departement(d) ∧
(t.EnseignantID=e.EnseignantID) ∧ …
∧ (t.DepartementID=d.DepartementID) ∧ … }

Enseignant ∞ Departement

$\Leftrightarrow$ {t / ∃e,d Enseignant(e) ∧ Departement(d) ∧
(t.EnseignantID=e.EnseignantID) ∧ …
∧ (t.DepartementID=d.DepartementID) ∧ …
∧ (e.DepartementID=d.DepartementID)}

Exemple d'écriture de théta-jointure

r :

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

s :

D	E
3	1
6	2

$r \underset{B<D}{\bowtie} s$:

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

$r \infty_{B<D} s$

$\Leftrightarrow$ **{t / ∃u,v r(u) ∧ s(v) ∧ (t.A=u.A) ∧ (t.B=u.B)**

∧ (t.C=u.C) ∧ (t.D=v.D) ∧ (t.E=v.E) ∧ (u.B<v.D)}

Division (1/4)

Exemple :

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID, ISBN, DateEmprunt,EtudiantID)

Etudiant(EtudiantID, Nom, Prenom)

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

La requête retourne les valeurs de l'attribut `Titre` *des nuplets* t *de la relation* `Livre` *tq :*

{ t.titre/ Livre(t) ∧ [∀ u Etudiant (u)

Quel que soit un nuplet u *si c'est un nuplet de* `Etudiant` *(i.e. quel que soit un étudiant) alors*

⇒ (∃ v Emprunt(v) ∧

(v.Etudiant_ID=u.Etudiant_ID) ∧ (v.ISBN=t.ISBN)

)

]

}

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

Division (2/4)

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 :

Pour tous les étudiants *u* (i.e. pour tous les nuplet *u* dans Etudiant)

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

Titre	IDEtudiant
Vives les Bases de données	1
Le SQL c'est facile!	1
Vives les Bases de données	3
Le SQL c'est facile!	2
Le SQL c'est facile!	3

v_1 v_2 v_3

IDEtudiant
1
2
3

u_1 u_2 u_3

Division (3/4)

Exemple :

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID, ISBN, DateEmprunt,EtudiantID)

Etudiant(EtudiantID, Nom, Prenom)

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

La requête retourne les valeurs l'attribut **`Titre`** *des nuplets* **t** *de la relation* **`Livre`** *tq :*

{ t.Titre/ Livre(t) ∧ [∀ u ¬ (Etudiant(u)) ∨

(∃ v Emprunt(v) ∧

(v.Etudiant_ID=u.Etudiant_ID) ∧ (v.ISBN=t.ISBN)

)

]

}

Quel que soit **u**, **u** *n'est pas dans* **`Etudiant`** *ou (***u** *est dans* ***`Etudiant`*** *et) il existe* **v** *…*

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

Division (4/4)

r :

	A	B	C	D	
t	a	b	c	d	v_1
	a	b	e	f	v_2
	g	h	c	d	
	i	j	k	ℓ	

s :

	C	D
u_1	c	d
u_2	e	f

$r \div s$:

A	B
a	b

r ÷ s

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

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

Expression de la division

Soient *R(A1, …,An,C1,…,Cm)* et *S(C1,…,Cm)*

- $r \div s \Leftrightarrow$ {t.A1, …t.An / r(t) ∧
  [∀u s(u) ⇒ ∃v r(v)∧ (v.C1=u.C1) ∧ … ∧
  (v.Cm=u.Cm) ∧ (t.A1=v.A1)
  ∧ … ∧ (t.An=v.An)
  ] }

$\Leftrightarrow$ {t.A1, …t.An / r(t) ∧
[∀u ¬ (s(u)) ∨ (∃v r(v)∧ (v.C1=u.C1) ∧ … ∧
(v.Cm=u.Cm) ∧ (t.A1=v.A1)
∧ … ∧ (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 »

$$\neg \; (\; \exists e \; Emprunt(e) \; \wedge \; \neg \; (\; \exists c \; Compte(c)$$
$$\wedge \quad (c.num\_client = e.num\_client)$$
$$\wedge \quad (c.nom\_banque = e.nom\_banque)$$
$$\wedge \quad (\; c.solde \geq (e.montant \; / \; 2)$$
$$) \;)$$

Chap V - Algèbre relationnelle étendue

- **Projection généralisée :**

  ajout d'expressions arithmétiques dans une projection

Compte

nomclient	credit	debit
Gamotte	2000	560
Debece	1352	850

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

nomclient	solde
Gamotte	1440
Debece	502

Différents types de jointure

- **Jointure : Association de plusieurs relations selon des critères,** généralement une égalité entre les valeurs d'une clé étrangère et celles de la clé primaire à laquelle elle fait référence
- **Jointure interne (*inner*) :** inclut seulement les nuplets des deux tables pour lesquelles les champs joints sont égaux.
- **Jointure naturelle** : jointure interne sur les champs ayant les mêmes noms dans les deux tables
- **Jointure externe (*outer*)** gauche (*left* - resp. droite/*right*) : inclut tous les nuplets de la table de gauche (resp. droite) et uniquement les nuplets de la table de droite (resp. gauche) pour lesquelles les champs joints sont égaux.

Jointure externe

- **Jointure externe** (***outer-join***) :
  - jointure externe à gauche :]∞
  - jointure externe à droite : ∞[
  - jointure externe :]∞[

  r]∞ s ⇒ r ∞ s et conservation des attributs des nuplets de r qui ne joignent avec aucun nuplet de s (les valeurs des attributs de s sont mises à NULL)

  r ∞**[s** ⇒ r ∞ s et conservation des attributs des nuplets de s qui ne joignent avec aucun nuplet de r (les valeurs des attributs de r sont mises à NULL)

  r **]∞[** s = (**r]∞** s) ∪ (r ∞**[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

  $\mathbf{Sum}_{\text{Capacite}}(\text{Salle})$

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

  $\mathbf{Avg}_{\text{Capacite}}(\text{Salle})$

- Nombre d'étudiants à l'Université

  $\mathbf{Count}_{\text{Etudiant\_ID}}(\text{Etudiant})$

- Capacité de la plus petite/grande salle

  $\mathbf{Min}_{\text{Capacite}}(\text{Salle})$ $\mathbf{Max}_{\text{Capacite}}(\text{Salle})$

- Nombre d'enseignants par départements :

  $\mathbf{Group}_{\text{Nom\_Departement}}, \mathbf{Count}_{\text{Enseignant\_ID}}(\text{Enseignant} \infty \text{Departement})$

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 Departement_Name ,**Count** Enseignant_ID(Enseignant ∞ Departement)

Department_Name	count
Math	2
Info	1

Chapitre VI - 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 2023** : SQL/PGQ (*property graph queries*) …

Langage SQL

- **Basé sur la logique du 1er 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/**
  - **http://deptfod.cnam.fr/bd/tp/**
  - **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
```

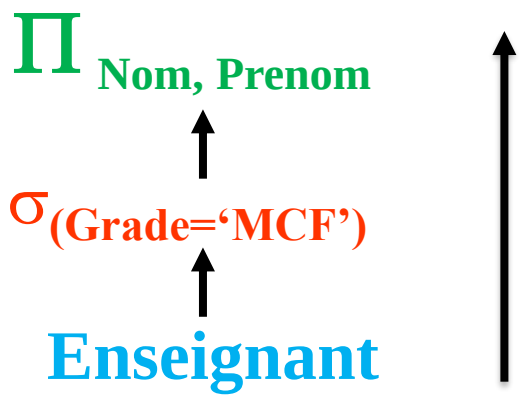
L..	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	NAIJA	Yosr
3	BAHRI	Afef
4	LIMAM	Medhi
5	MyTaylor	IsRich
6	RIGAUX	Philippe
7	CHAKHAR	Salem
8	MURAT	Cécile

```
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'
```



Plan d'exécution

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);
```

LIKE et ILIKE

ILIKE est utilisé à la place de LIKE pour faire des correspondances sans tenir compte de la casse (pas standard mais dans PostgreSQL)

```
SELECT * FROM R2;
```

pid	personne	age	sexe
1	Ines	99	F
2	Paul	101	F
3	Ahmed	75	M

```
SELECT * FROM R2 WHERE Personne ILIKE '%D';
```

pid	personne	age	sexe
3	Ahmed	75	M

```
SELECT * FROM R2 WHERE Personne LIKE '%D';
```

Query has no result

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 entre ()
- FAUX si aucun nuplet n'est retourné par la requête entre ()
- 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

COUNT

- **`COUNT(*)` compte combien il y a de nuplets**
- **`COUNT(Att)` compte combien il y a de valeurs `NON NULL` d'un attribut**
- **`COUNT(DISTINCT Att)` compte combien il y a de valeurs `NON NULL` distinctes d'un attribut**

```
SELECT * FROM etudiant ORDER BY Ville, Nom;
```

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
667	Dupond	Noémie	1987-09-18	Rue de Dôle	69007	Lyon
110	Dupont	Albert	1980-06-01	Rue de Crimée	69001	Lyon
421	Durand	Gaston	1980-11-15	Rue de la Meuse	69008	Lyon

```
SELECT COUNT(*) FROM etudiant ;
```

COUNT(*)
7

```
SELECT COUNT(ville) FROM etudiant ;
```

COUNT(ville)
6

```
SELECT COUNT(DISTINCT ville) FROM etudiant ;
```

COUNT(DISTINCT ville)
4

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 :

```
SELECT a,
 CASE WHEN a=1 THEN 'un'
      WHEN a=2 THEN 'deux'
      ELSE 'autres'
   END
FROM test;
```

```
 a
---
 1
 2
 3
```

```
 a | case
---+-------
 1 | un
 2 | deux
 3 | autres
```

Exemple de requête avec `CASE ... WHEN`

Une relation `Employees` :

employeeid	firstname	lastname	city	country	managerid
1	John	Doe	Seattle	USA	null
2	Jane	Smith	New York	USA	1
3	Bob	Johnson	Seattle	USA	1
4	Alice	Brown	Chicago	USA	2
5	Charlie	Davis	Seattle	USA	3
6	Eva	Garcia	Madrid	Spain	3
7	Sophie	Lee	Seoul	South Korea	2
8	Thomas	Miller	New York	USA	1
9	Hiroshi	Sato	Tokyo	Japan	7
10	Wei	Li	Shanghai	China	7

Requête suivie de son résultat :

```
SELECT FirstName, LastName, City,
       CASE
         WHEN Country = 'USA' THEN 'Domestic'
         ELSE 'International'
       END AS Location
FROM Employees;
```

firstname	lastname	city	location
John	Doe	Seattle	Domestic
Jane	Smith	New York	Domestic
Bob	Johnson	Seattle	Domestic
Alice	Brown	Chicago	Domestic
Charlie	Davis	Seattle	Domestic
Eva	Garcia	Madrid	International
Sophie	Lee	Seoul	International
Thomas	Miller	New York	Domestic
Hiroshi	Sato	Tokyo	International
Wei	Li	Shanghai	International

Exemple de requête plus complexe CASE … WHEN

Personne

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

NbTicketsAchetes

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

Résultat de la requête

nom character varyir	prenom character v	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 var	numero_salle character varying(10)	enseignement_id integer	master_id integer	enseignant_id integer	date_resa date	heure_debut time without	heure_fin time without	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

EVERY : fonction d'agrégation

Une relation `Employees` :

employeeid	firstname	lastname	city	country	managerid
1	John	Doe	Seattle	USA	null
2	Jane	Smith	New York	USA	1
3	Bob	Johnson	Seattle	USA	1
4	Alice	Brown	Chicago	USA	2
5	Charlie	Davis	Seattle	USA	3
6	Eva	Garcia	Madrid	Spain	3
7	Sophie	Lee	Chicago	USA	2
8	Thomas	Miller	New York	USA	1
9	Hiroshi	Sato	Tokyo	Japan	7
10	Wei	Li	Shanghai	China	7

Utiliser `EVERY` dans un `HAVING` :

```
SELECT ManagerID
FROM Employees
GROUP BY ManagerID
HAVING ManagerID IS NOT NULL
   AND EVERY(City = 'Chicago');
```

managerid
2

Ne pas utiliser `EVERY` dans un `WHERE` !

```
SELECT *
FROM Employees
WHERE EVERY (City = 'Seattle');
```

Query Error: error: aggregate functions are not allowed in WHERE

Différents types de jointure

- **Jointure : Association de plusieurs relations selon des critères,** généralement une égalité entre les valeurs d'une clé étrangère et celles de la clé primaire à laquelle elle fait référence
- **Jointure interne (*inner*) :** inclut seulement les nuplets des deux tables pour lesquelles les champs joints sont égaux.
- **Jointure naturelle** : jointure interne sur les champs ayant les mêmes noms dans les deux tables
- **Jointure externe (*outer*)** gauche (*left* - resp. droite/*right*) : inclut tous les nuplets de la table de gauche (resp. droite) et uniquement les nuplets de la table de droite (resp. gauche) pour lesquelles les champs joints sont égaux.

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
```

Différente manière d'écrire la jointure naturelle

On peut écrire une jointure naturelle de différente manière.

Par exemple, en prenant la base exemple de l'exercice 28, R1∞R3 peut s'écrire de 3 manières différentes :

```
SELECT * FROM R1, R3
WHERE R1.Enfant=R3.Enfant;
ou
SELECT *
FROM R1 INNER JOIN R3 ON R1.Enfant=R3.Enfant;
```

Dans ce cas l'attribut *Enfant* apparait 2 fois (pour R1 et pour R3) :

parent	enfant	enfant	ecole
Pierre	Berenice	Berenice	Ecole Merveilleuse
Marie	Berenice	Berenice	Ecole Merveilleuse
Pierre	Jacques	Jacques	Super College
Marie	Jacques	Jacques	Super College
Samir	Tristan	Tristan	Ecole Merveilleuse
Kim	Tristan	Tristan	Ecole Merveilleuse

ou

```
SELECT * FROM R1 NATURAL JOIN R3;
```

Dans ce cas l'attribut *Enfant* apparait 1 fois :

enfant	parent	ecole
Berenice	Pierre	Ecole Merveilleuse
Berenice	Marie	Ecole Merveilleuse
Jacques	Pierre	Super College
Jacques	Marie	Super College
Tristan	Samir	Ecole Merveilleuse
Tristan	Kim	Ecole Merveilleuse

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

Fonctions et opérateurs préfinis

Souvent spécifiques au SGBD – Exemple sous PostgreSQL :

- `ABS(num)` : valeur absolue
- `str1 || str2` : concaténation de chaînes de caractères
- `NOW()` : la date et heure courante
- `CEILING(num)` : l'entier immédiatement supérieur ou égal à num
- `FLOOR(num)` : l'entier immédiatement inférieur ou égal à num
- `CURRENT_DATE`: la date courante
- `CURRENT_TIME` : l'heure courante
- …

```
SELECT Nom, Prenom, EXTRACT(YEAR FROM NOW()) - EXTRACT(YEAR FROM Date_Naissance) As Age
FROM Etudiant
```

nom	prenom	date_naissance
HIBULAIRE	Pat	23/08/1980
ODENT	Jamal	12/05/1978
DEBECE	Gill	15/07/1979
DEBECE	Aude	15/08/1979

nom	prenom	age
HIBULAIRE	Pat	44
ODENT	Jamal	46
DEBECE	Gill	45
DEBECE	Aude	45

Requête dans un FROM

Possibilité d'utiliser une requête dans le FROM :

Obligation d'utiliser une variable nuplet associée à la sous requête du FROM, même si on ne s'en sert pas

```
SELECT Parent, COUNT(Enfant) as nbEnfants
FROM R1
GROUP BY Parent;
```

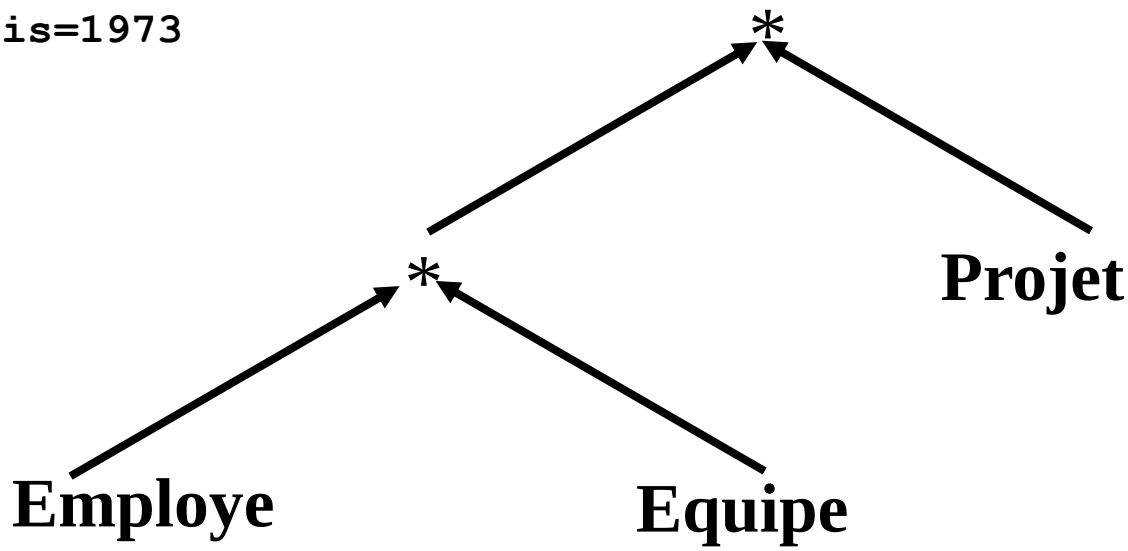
parent	nbenfants
Yaelle	1
Samir	1
Paul	1
Ines	1
Kim	1
Ahmed	2
Pierre	2
Marie	2
Raoul	1

```
SELECT AVG(nbEnfants)
FROM (SELECT Parent, COUNT(Enfant) as nbEnfants FROM R1 GROUP BY Parent) foo ;
```

avg
1.3333333333333333

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
```



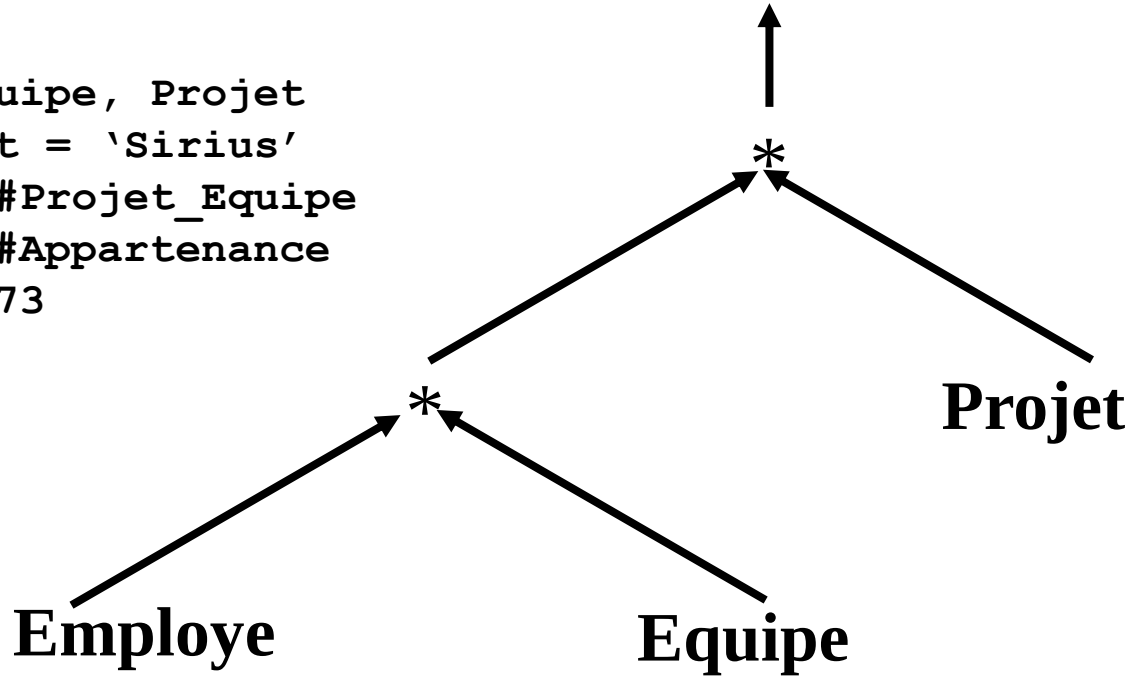
Analyse du FROM

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

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

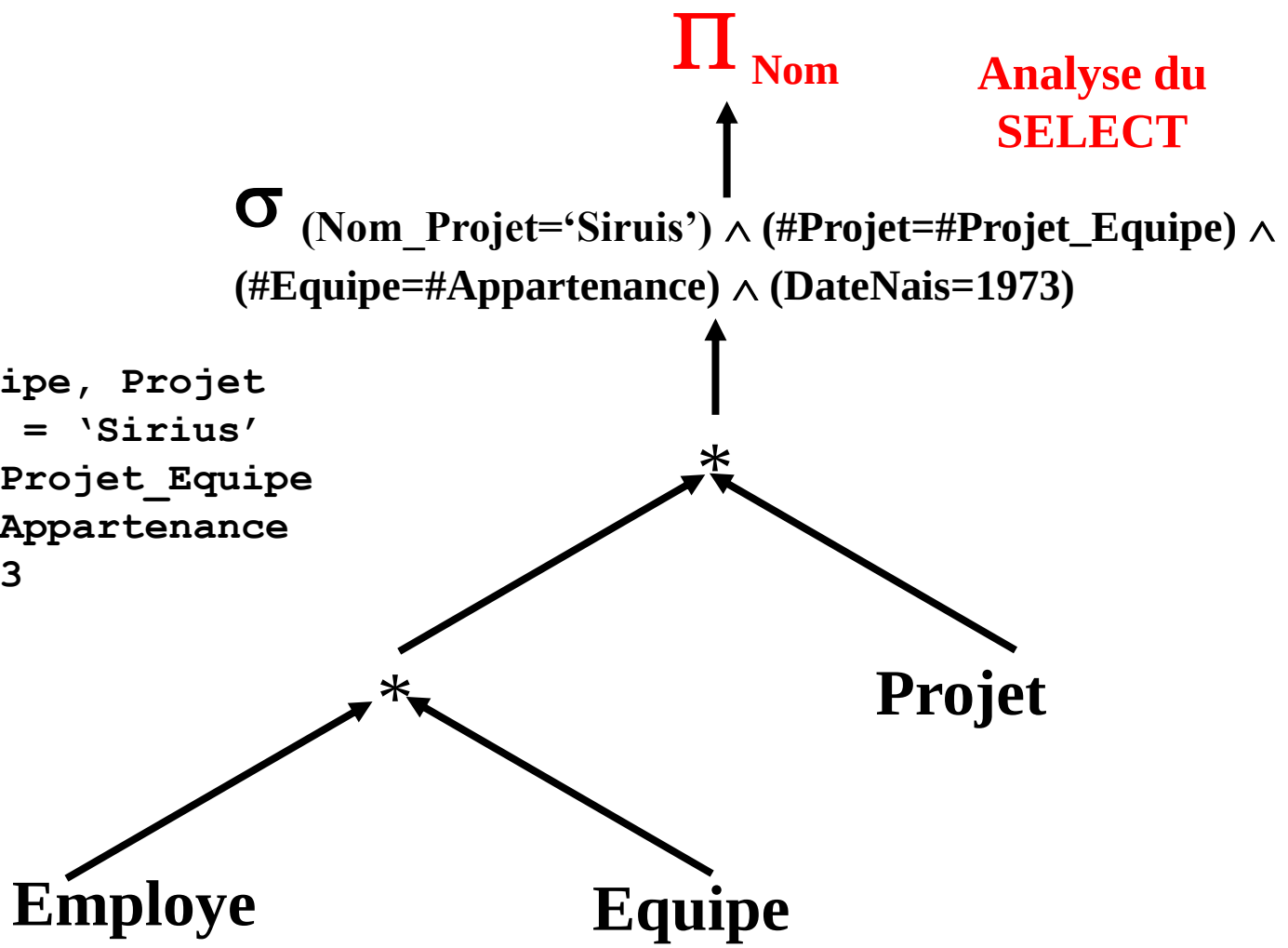
```
SELECT Nom
 FROM Employe, Equipe, Projet
 WHERE Nom_Projet = 'Sirius'
   AND #Projet = #Projet_Equipe
   AND #Equipe = #Appartenance
   AND DaeNais=1973
```

Analyse du WHERE



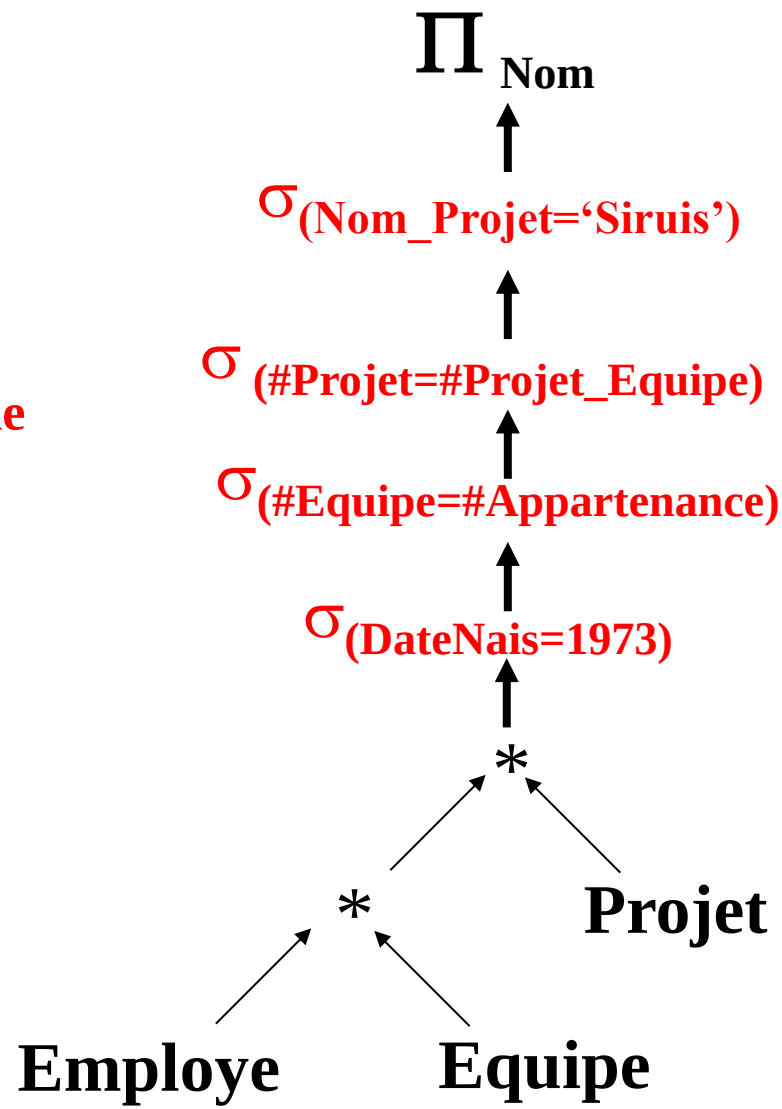
Lien entre l'algèbre relationnelle et le SQL (3/3)

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



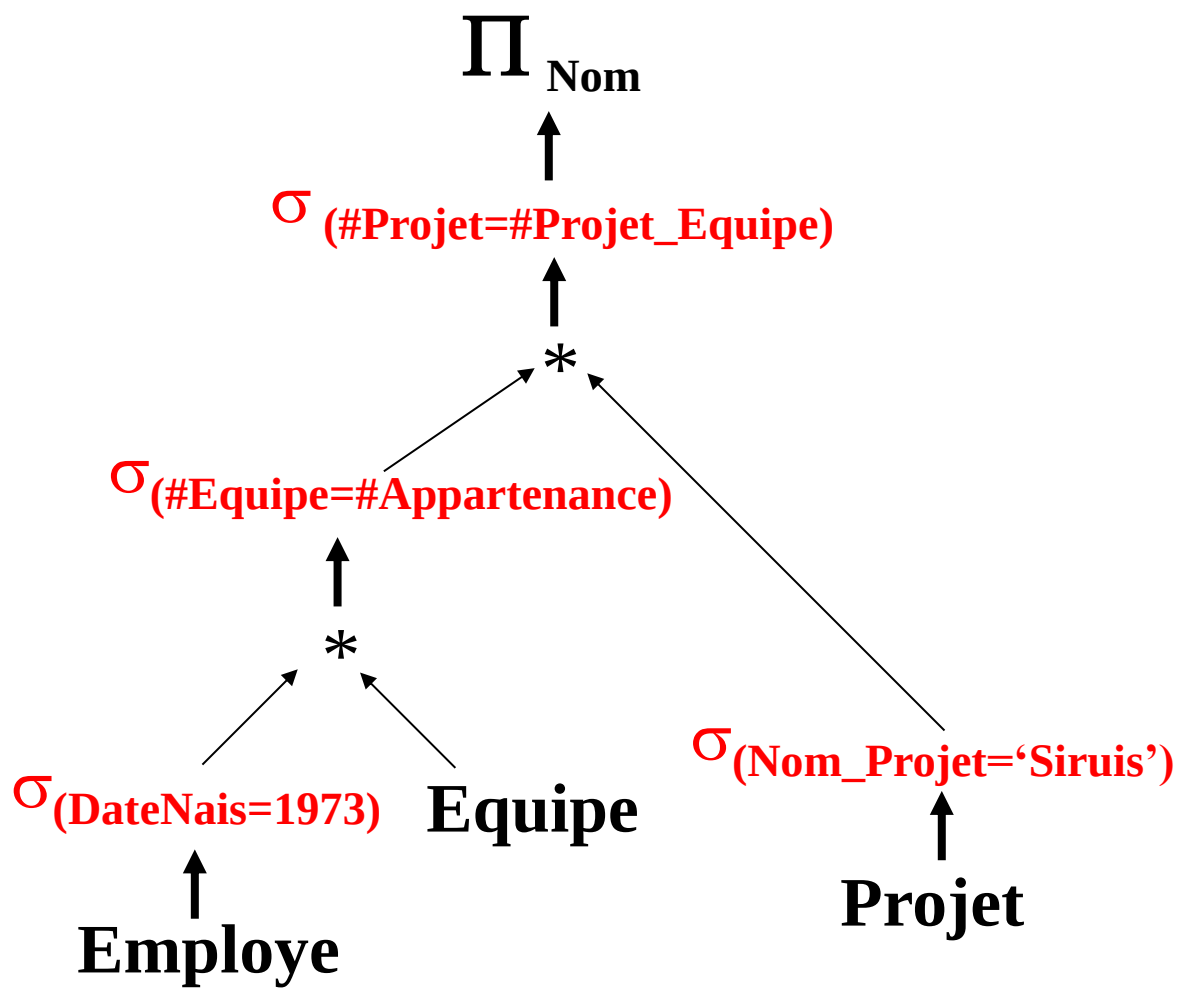
Exemple d'optimisation de requêtes (1/6)

Division des conjonctions de sélections



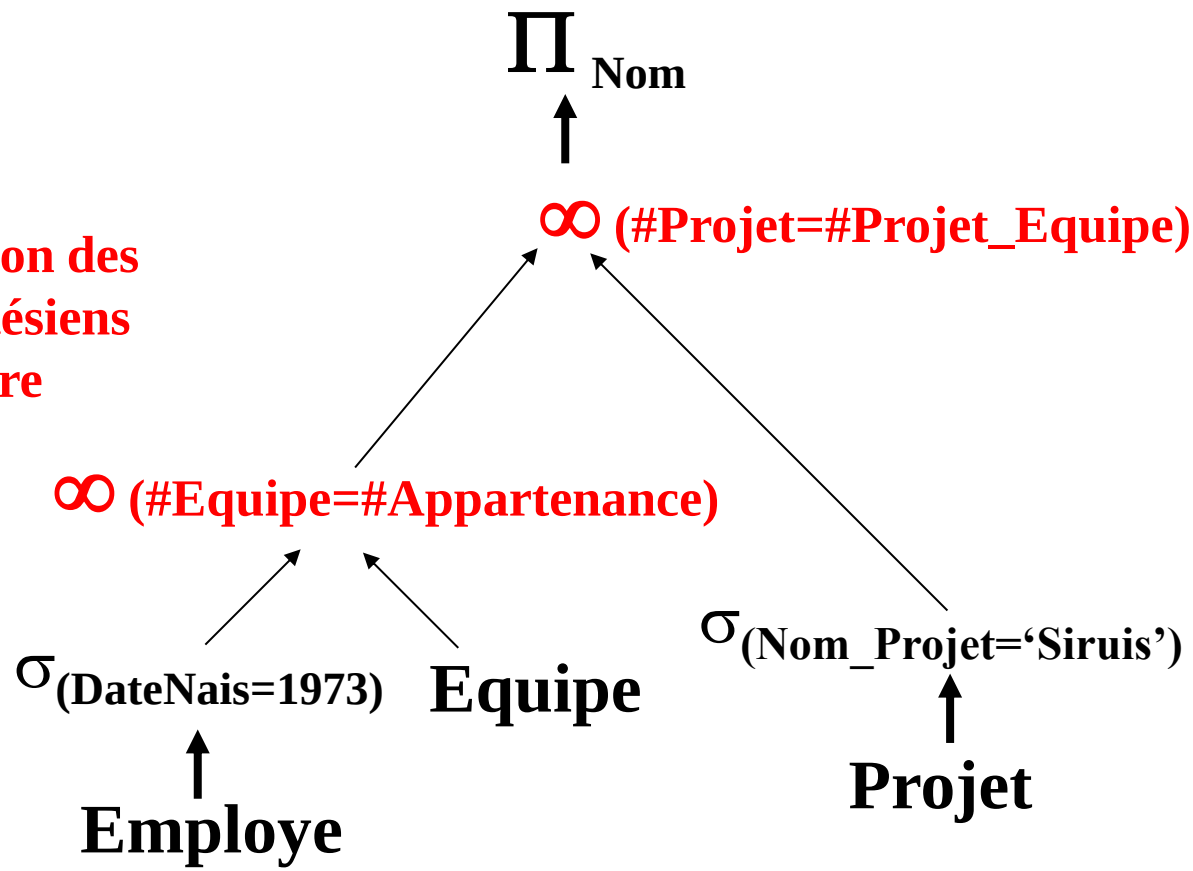
Exemple d'optimisation de requêtes (2/6)

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



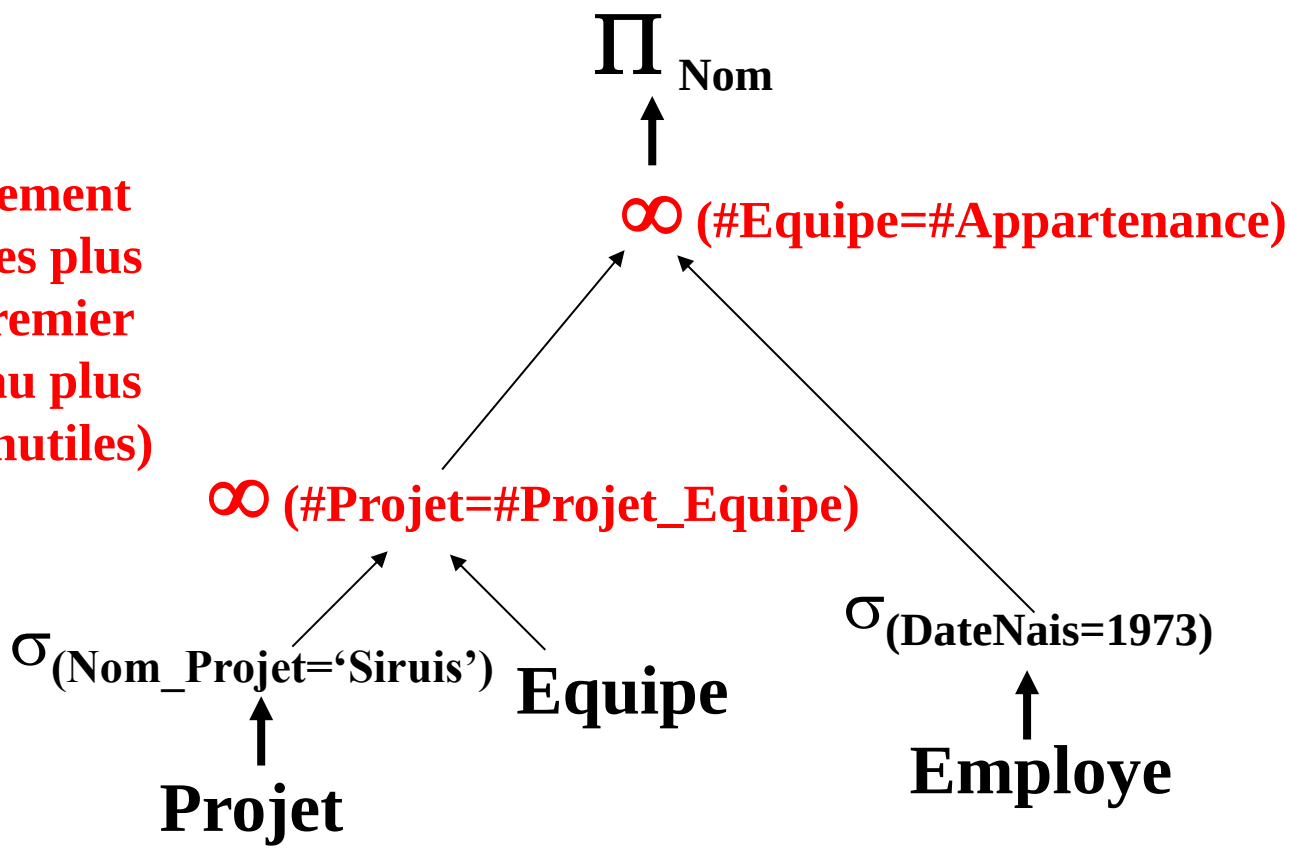
Exemple d'optimisation de requêtes (3/6)

Transformation des produits cartésiens en jointure



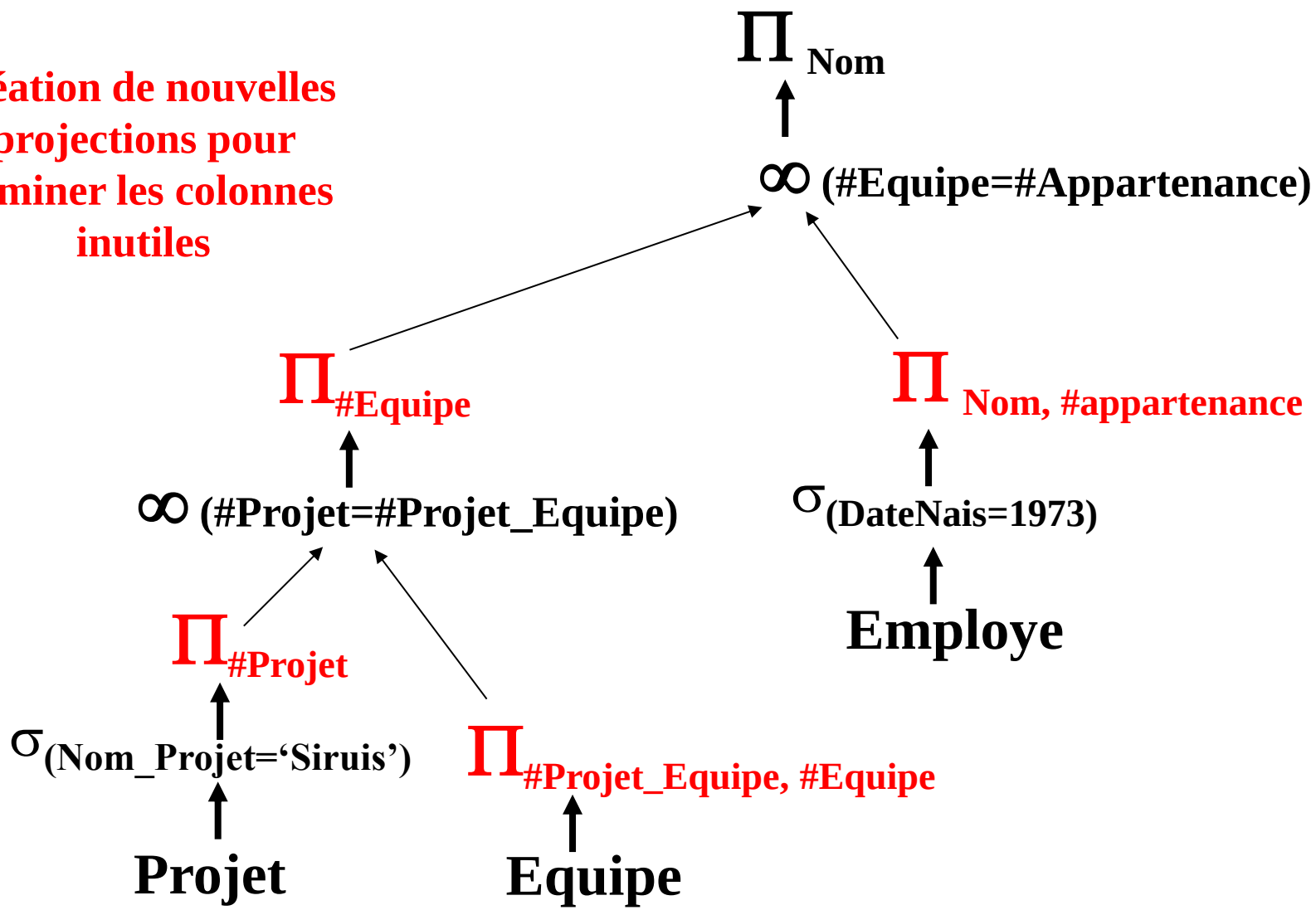
Exemple d'optimisation de requêtes (4/6)

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



Exemple d'optimisation de requêtes (5/6)

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 :

$$\Pi_{Nom}\left[\ \Pi_{Nom,\ \#appartenance}\left[\ \sigma_{(DateNais=1973)}(Employe)\ \right]\right.$$

$$\infty\ (\#Equipe=\#Appartenance)$$

$$\Pi_{\#Equipe}\left(\ \Pi_{\#Projet}\left[\sigma_{(Nom\_Projet='Siruis'}(Projet\ )\right]\right.$$

$$\infty\ (\#Projet=\#Projet\_Equipe)$$

$$\left.\Pi_{\#Projet\_Equipe,\ \#Equipe}(Equipe)\ \ )\ \ \right]$$

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 ANALYSE (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

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')
```

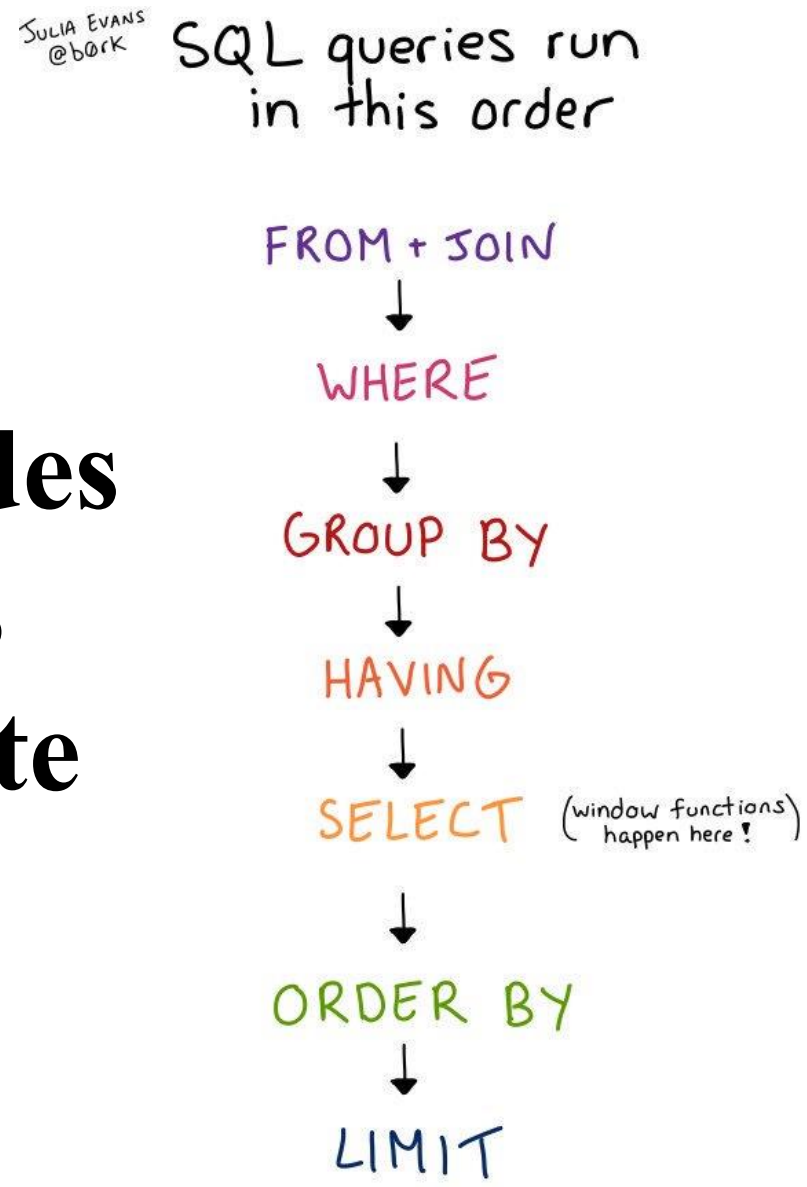
```
select titre from Film
where annee = 1958
and  id_film in
       (select id_film from Rôle
        where id_acteur in
              (select id_acteur
               from Artiste
               where nom='Stewart'
               and prénom='James'))
```

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 !

Ordre d'exécution des opérateurs d'une requête



Division en SQL (1/4)

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID,ISBN, DateEmprunt,EtudiantID)

Etudiant(EtudiantID,Nom, Prenom)

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

$$\Pi_{Titre,EtudiantID}(Livre \bowtie Emprunt) \div \Pi_{EtudiantID}(Etudiant)$$

Les titres des livres et les id des étudiants emprunteurs

Titre	IDEtudiant
Vives les Bases de données	1
Le SQL c'est facile!	1
Vives les Bases de données	3
Le SQL c'est facile!	2
Le SQL c'est facile!	3

Les ID de **tous** les étudiants de la base de données

IDEtudiant
1
2
3

=> Renvoie une table composée d'un attribut *Titre*, contenant les titres des livres, qui dans le résultat de la jointure, sont associés aux id de tous les étudiants

Titre
Le SQL c'est facile!

Division en SQL (2/4)

```
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 :

Pour tous les étudiants *u* (i.e. pour tous les nuplet *u* dans Etudiant)

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

Titre	IDEtudiant
Vives les Bases de données	1
Le SQL c'est facile!	1
Vives les Bases de données	3
Le SQL c'est facile!	2
Le SQL c'est facile!	3

v_1 ← Le SQL c'est facile! 1

v_2 ← Le SQL c'est facile! 2

v_3 ← Le SQL c'est facile! 3

IDEtudiant
1
2
3

u_1 ← 1

u_2 ← 2

u_3 ← 3

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

Division (3/4)

`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 un 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 (4/4)

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID,ISBN, DateEmprunt,EtudiantID)

Etudiant(EtudiantID,Nom, Prenom)

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

{t.Titre / Livre(t) ∧ ¬ [∃ u Etudiant (u) ∧
¬ (∃ v Emprunt(v) ∧
(v.Etudiant_ID=u.Etudiant_ID) ∧ (v.ISBN=t.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
        )
    ) ;
```

Exemple de 2 écritures d'une même division en SQL

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

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 )
    ) ;
```

Autre écriture :

```
SELECT Nom, Prenom FROM Joueur NARURAL JOIN Convocation
GROUP BY Nom, Prenom
HAVING COUNT (DISTINCT MID)
 =   SELECT COUNT(DISTINCT(MID) FROM Match)
```

Autre exemple de division en SQL

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

```
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
```

Ne pas oublier `RETURNIG` dans `WITH` (1/2)

Schema SQL

```
1  CREATE TABLE R2 (
2    Personne varchar(25),
3    Age integer NOT NULL
4    CONSTRAINT CK_R2_Age CHECK (Age>=0),
5    Sexe varchar(1)
6    CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N')),
7    CONSTRAINT PK_R2 PRIMARY KEY(Personne)
8    );
9
10 CREATE TABLE R1 (
11   Parent varchar(25),
12   Enfant varchar(25),
13   CONSTRAINT PK_R1 PRIMARY KEY(Parent,Enfant),
14   CONSTRAINT FK_R1_Parent FOREIGN KEY (Parent) REFERENCES R2(Personne),
```

Text to DDL

Query SQL

```
1 WITH t AS (
2     UPDATE R2 SET age = age +1
3     --RETURNING *
4 )
5 SELECT * FROM t;
```

Results

Query Error: error: WITH query "t" does not have a RETURNING clause

Ne pas oublier `RETURNIG` dans `WITH` (2/2)

Schema SQL

```
1 CREATE TABLE R2 (
2   Personne varchar(25),
3   Age integer NOT NULL
4   CONSTRAINT CK_R2_Age CHECK (Age>=0),
5   Sexe varchar(1)
6   CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N')),
7   CONSTRAINT PK_R2 PRIMARY KEY(Personne)
8   );
9
10 CREATE TABLE R1 (
11   Parent varchar(25),
12   Enfant varchar(25),
13   CONSTRAINT PK_R1 PRIMARY KEY(Parent,Enfant),
14   CONSTRAINT FK_R1_Parent FOREIGN KEY (Parent) REFERENCES R2(Personne),
```

Text to DDL

Query SQL

```
1 WITH t AS (
2     UPDATE R2 SET age = age +1
3     RETURNING *
4 )
5 SELECT * FROM t;
```

Results

Query #1 Execution time: 2ms

personne	age	sexe
Ines	100	F

Clause WITH RECURSIVE

- Pour faire des itérations

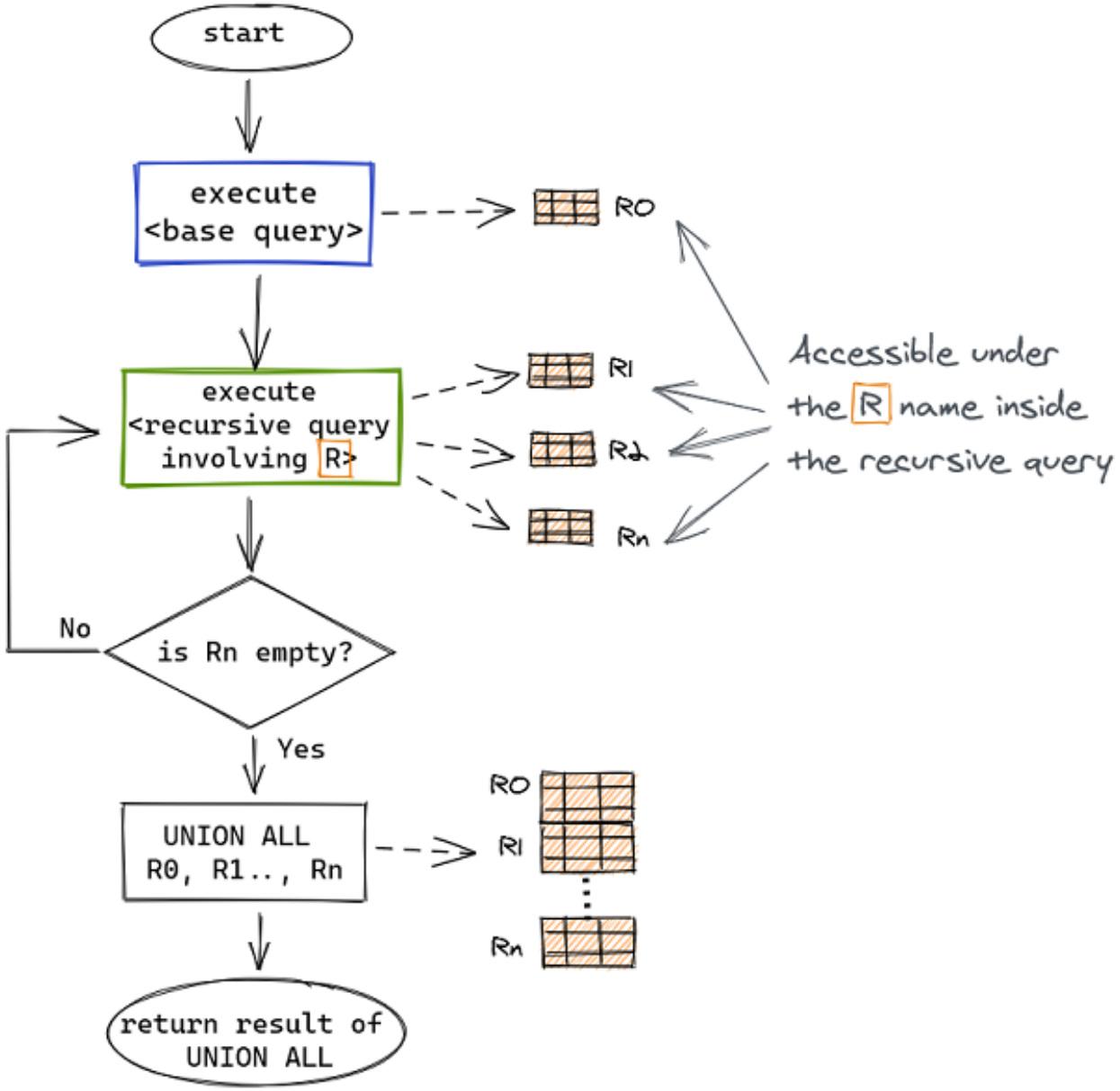
```
WITH RECURSIVE R AS (
      requeteDeBase
    UNION ALL
      requeteFaisantReferenceàR
    )
SELECT ... FROM R
```

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

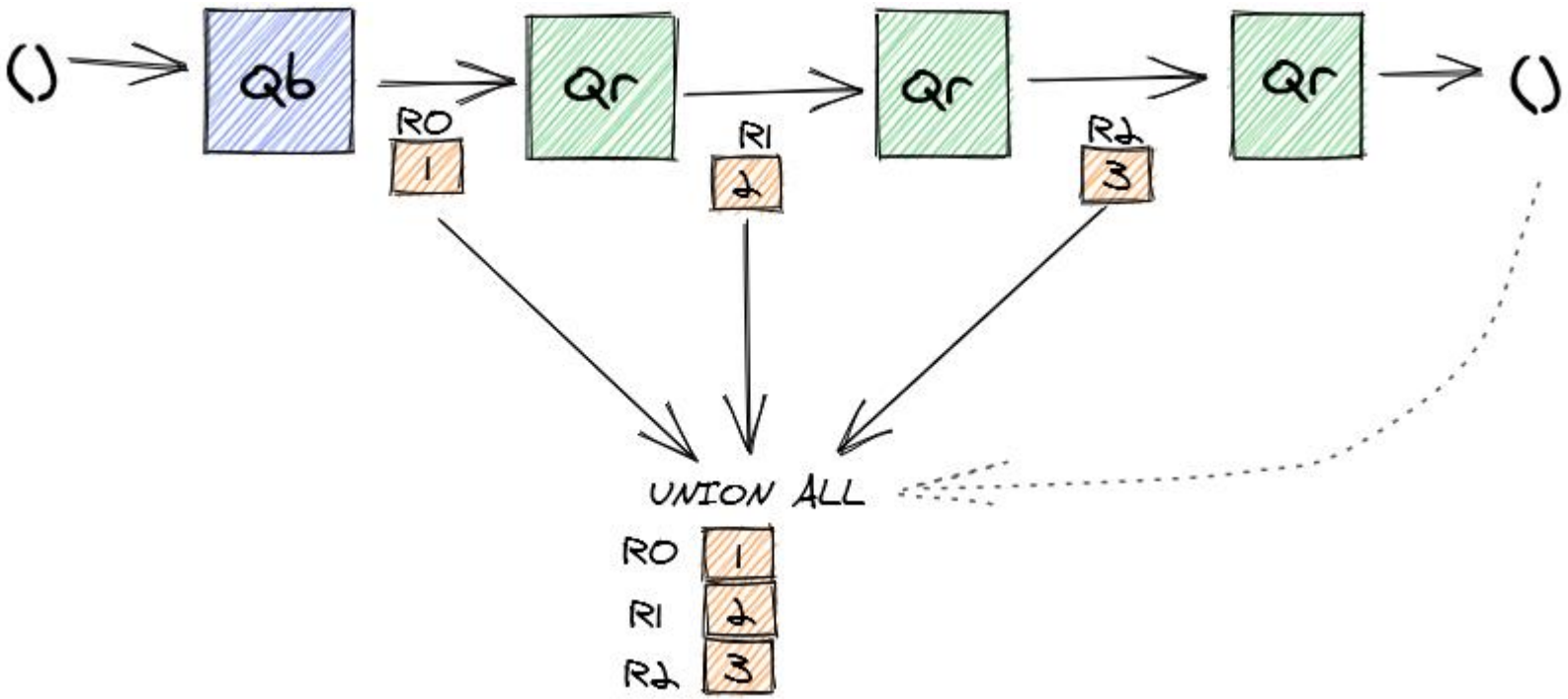
```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

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

Autre de requête avec `WITH RECURSIVE`

Une relation `Employees` :

employeeid	firstname	lastname	city	country	managerid
1	John	Doe	Seattle	USA	null
2	Jane	Smith	New York	USA	1
3	Bob	Johnson	Seattle	USA	1
4	Alice	Brown	Chicago	USA	2
5	Charlie	Davis	Seattle	USA	3
6	Eva	Garcia	Madrid	Spain	3
7	Sophie	Lee	Chicago	USA	2
8	Thomas	Miller	New York	USA	1
9	Hiroshi	Sato	Tokyo	Japan	7
10	Wei	Li	Shanghai	China	7

```
WITH RECURSIVE EmployeeHierarchy (EmployeeID, FirstName, LastName,
ManagerID, Level) AS (
  SELECT EmployeeID, FirstName, LastName, ManagerID,
         0 AS Level
    FROM Employees
      WHERE ManagerID IS NULL
 UNION ALL
   SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID,
          eh.Level + 1
     FROM Employees e
       JOIN EmployeeHierarchy eh
          ON e.ManagerID = eh.EmployeeID
)
SELECT EmployeeID, FirstName, LastName, ManagerID, Level
```

employeeid	firstname	lastname	managerid	level
1	John	Doe	null	0
2	Jane	Smith	1	1
3	Bob	Johnson	1	1
8	Thomas	Miller	1	1
4	Alice	Brown	2	2
5	Charlie	Davis	3	2
6	Eva	Garcia	3	2
7	Sophie	Lee	2	2
9	Hiroshi	Sato	7	3
10	Wei	Li	7	3

Requêtes préparées ou paramétrées

Séparation des requêtes et de leurs paramètres au moment de leur exécution (pour éviter les attaques type injections SQL)

Par exemple, en prenant la base exemple de l'exercice 28 :

Table R2

Personne	age	sexe
Ines	99	F
Paul	101	F
Ahmed	75	M
Raoul	70	M
Yaelle	71	M
Pierre	45	M
Marie	40	F
Berenice	9	F
Jacques	14	M
Samir	50	M
Kim	51	F
Tristan	5	M

Query SQL

```
1 PREPARE recherche_par_sexe(CHAR) as
2 SELECT *
3 FROM R2
4 WHERE Sexe=$1;
5
6 EXECUTE recherche_par_sexe('F');
7
```

personne	age	sexe
Ines	99	F
Paul	101	F
Marie	40	F
Berenice	9	F
Kim	51	F

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 ...

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

ON DELETE RESTRICT

```
CREATE TABLE R2 (
  Personne varchar(25),
  Age integer NOT NULL
  CONSTRAINT CK_R2_Age CHECK (Age>=0),
  Sexe varchar(1)
  CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N')),
  CONSTRAINT PK_R2 PRIMARY KEY(Personne)
  );

 CREATE TABLE R3 (
   Enfant varchar(25),
   Ecole  varchar(25),
   CONSTRAINT PK_R3 PRIMARY KEY(Ecole,Enfant),
   CONSTRAINT FK_R3_Enfant FOREIGN KEY (Enfant) REFERENCES R2(Personne) ON DELETE RESTRICT
   );

INSERT INTO R2 VALUES ('Berenice', 9, 'F');
INSERT INTO R2 VALUES ('Jacques', 14, 'M');
INSERT INTO R2 VALUES('Tristan', 5, 'M');

INSERT INTO R3 VALUES  ('Berenice', 'Ecole Merveilleuse');
INSERT INTO R3 VALUES  ('Tristan', 'Ecole Merveilleuse') ;
INSERT INTO R3 VALUES  ('Jacques', 'Super College') ;
```

personne	age	sexe
Berenice	9	F
Jacques	14	M
Tristan	5	M

enfant	ecole
Berenice	Ecole Merveilleuse
Tristan	Ecole Merveilleuse
Jacques	Super College

```
DELETE FROM R2 WHERE personne = 'Tristan';
```

La suppression d'un nuplet est interdite dans R2 s'il est référencé dans R3

Query Error: error: update or delete on table "r2" violates foreign key constraint "fk_r3_enfant" on table "r3"

ON DELETE CASCADE

```
CREATE TABLE R2 (
  Personne varchar(25),
  Age integer NOT NULL
  CONSTRAINT CK_R2_Age CHECK (Age>=0),
  Sexe varchar(1)
  CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N')),
  CONSTRAINT PK_R2 PRIMARY KEY(Personne)
  );

 CREATE TABLE R3 (
   Enfant varchar(25),
   Ecole  varchar(25),
   CONSTRAINT PK_R3 PRIMARY KEY(Ecole,Enfant),
   CONSTRAINT FK_R3_Enfant FOREIGN KEY (Enfant) REFERENCES R2(Personne) ON DELETE CASCADE
   );

INSERT INTO R2 VALUES ('Berenice', 9, 'F');
INSERT INTO R2 VALUES ('Jacques', 14, 'M');
INSERT INTO R2 VALUES('Tristan', 5, 'M');

INSERT INTO R3 VALUES  ('Berenice', 'Ecole Merveilleuse');
INSERT INTO R3 VALUES  ('Tristan', 'Ecole Merveilleuse') ;
INSERT INTO R3 VALUES  ('Jacques', 'Super College') ;
```

personne	age	sexe
Berenice	9	F
Jacques	14	M
Tristan	5	M

enfant	ecole
Berenice	Ecole Merveilleuse
Tristan	Ecole Merveilleuse
Jacques	Super College

```
DELETE FROM R2 WHERE personne = 'Tristan';
```

La suppression d'un nuplet dans R2 supprime les nuplets correspondants dans R3

```
SELECT * FROM R2;
```

personne	age	sexe
Berenice	9	F
Jacques	14	M

```
SELECT * FROM R3;
```

enfant	ecole
Berenice	Ecole Merveilleuse
Jacques	Super College

Exemples de création de table (2/3)

```
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 (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 )
)
```

SERIAL (1/4)

Pour indiquer que la clé primaire est une clé artificielle :

```
CREATE TABLE R2 (
  PID SERIAL PRIMARY KEY,
  Personne varchar(25),
  Age integer NOT NULL
  CONSTRAINT CK_R2_Age CHECK (Age>=0),
  Sexe varchar(1)
  CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N'))
  );
```

```
INSERT INTO R2(Personne,Age,Sexe) VALUES ('Ines', 99, 'F'),
('Paul', 101, 'F'),('Ahmed', 75, 'M');

SELECT * FROM R2;
```

Insertion en ne donnant que les valeurs des attributs Personne, Age et Sexe

pid	personne	age	sexe
1	Ines	99	F
2	Paul	101	F
3	Ahmed	75	M

La valeur de PID a été générée par le SGBD

SERIAL (2/4)

Attention, on peut insérer plusieurs fois la même personne :

```
CREATE TABLE R2 (
  PID SERIAL PRIMARY KEY,
  Personne varchar(25),
  Age integer NOT NULL
  CONSTRAINT CK_R2_Age CHECK (Age>=0),
  Sexe varchar(1)
  CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N'))
  );
```

```
INSERT INTO R2(Personne,Age,Sexe) VALUES ('Ines', 99, 'F'),('Paul', 101, 'F'),
('Ahmed', 75, 'M'),('Ahmed', 75, 'M'),('Ahmed', 75, 'M'),('Ahmed', 75, 'M'),('Ahmed', 75, 'M');
```

```
SELECT * FROM R2;
```

pid	personne	age	sexe
1	Ines	99	F
2	Paul	101	F
3	Ahmed	75	M
4	Ahmed	75	M
5	Ahmed	75	M
6	Ahmed	75	M
7	Ahmed	75	M

SERIAL (3/4)

Avec une clé artificielle, il est préférable d'ajouter une contrainte d'unicité pour interdire l'insertion de plusieurs fois la même information :

```
CREATE TABLE R2 (
  PID SERIAL PRIMARY KEY,
  Personne varchar(25),
  Age integer NOT NULL
  CONSTRAINT CK_R2_Age CHECK (Age>=0),
  Sexe varchar(1)
  CONSTRAINT CK_R2_Sexe CHECK (Sexe IN ('M','F','N')),
  CONSTRAINT Macontrainte UNIQUE(Personne,Age,Sexe)
  );

INSERT INTO R2(Personne,Age,Sexe) VALUES ('Ines', 99, 'F'),('Paul', 101, 'F'),
('Ahmed', 75, 'M'),('Ahmed', 75, 'M'),('Ahmed', 75, 'M'),('Ahmed', 75, 'M'),('Ahmed', 75, 'M');
```

duplicate key value violates unique constraint "macontrainte"

SERIAL (4/4)

L'implantation des clés artificielles n'est pas la même dans tous les SGBD :

```
CREATE TABLE apprentis
(
  apprentis_id serial NOT NULL,
  nom character varying(25) NOT NULL,
  prenom character varying(25) NOT NULL,
  age integer DEFAULT 20,
  CONSTRAINT pk_apprentis PRIMARY KEY (apprentis_id),
  CONSTRAINT macontrainte UNIQUE(nom,prenom)
);
```

On veut donner la valeur 2 à la clé artificielle →

```
INSERT INTO apprentis(nom, prenom) VALUES ('Gamotte','Albert');
INSERT INTO apprentis VALUES (2,'Odent','Jamal',24);
INSERT INTO apprentis(nom, prenom) VALUES ('Onette','Camille');
```

- **Sous MySQL** : pas d'erreur, les clés artificielles générées auront pour valeur 1 pour *Albert*, 2 pour *Jamal* et 3 pour *Camille*
- **Sous PostgreSQL**, comme les clés artificielles sont gérées par des séquences, on aura l'erreur

  Schema Error: error: duplicate key value violates unique constraint "pk_apprentis"

  car le SGBD va vouloir générer la valeur 2 pour Camille.

Vous pouvez tester différents SGBD sous https://www.db-fiddle.com/

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

Possibilité de définir des contraintes en dehors du mot-clé CONSTRAINT

```
CREATE TABLE Ingredient
( IID SERIAL PRIMARY KEY,
  Nom varchar(25) NOT NULL UNIQUE,
  CO2 float NOT NULL DEFAULT 0
);

CREATE TABLE Plat
( PID SERIAL PRIMARY KEY,
  Nom varchar(50) NOT NULL UNIQUE,
  CO2 float NOT NULL DEFAULT 0
);

CREATE TABLE CompositionPlat
( IID integer NOT NULL REFERENCES Ingredient(IID) ON DELETE CASCADE,
  PID integer NOT NULL REFERENCES Plat(PID) ON DELETE CASCADE,
  CONSTRAINT PK_Composition PRIMARY KEY(PID,IID)
);
```

Définition de la clé primaire

Contrainte d'unicité

Définition d'une clé étrangère

Quand la contrainte ne touche qu'un seul attribut

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)]

corps du déclencheur écrit dans un langage propre au SGBD mélangeant SQL et mots-clés

événement_déclencheur = INSERT, UPDATE, DELETE

- **Déclencheur de *niveau instruction* : pas de clause FOR EACH ROW**
- **Déclencheur de *niveau ligne* : clause FOR EACH ROW**

  2 variables *:new* ou *NEW* et *:old* ou *OLD*

  - **INSERT : *:new* ou *NEW* représente le nuplet en cours d'insertion**
  - **UPDATE : *:old* ou *OLD* représente le nuplet en cours de mise à jour avant modification et *:new* ou *NEW* le nuplet après modification**
  - **DELETE : *:old* ou *OLD* représente le nuplet en cours de suppression**

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;
```

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) ;
```

Exemple de corps de 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'';
   -- il faut une double quote pour délimiter le début et la fin de la
   chaîne pas des guillemets
   ELSE RETURN NEW;
   END IF; -- un END IF ; par IF
END;'  -- la quote délimite le début et la fin du corps de la fonction
LANGUAGE 'plpgsql';
```

Définition SQL du 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 affichant un message (1/2)

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 affichant un message (2/2)

Sous PostgreSQL :

```
-- Fonction contenant le corps du trigger
 CREATE OR REPLACE Function 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 modifiant des nuplets (1/2)

Sous PostgreSQL :

```
CREATE TABLE Ingredient
( IID SERIAL PRIMARY KEY,
  Nom varchar(25) NOT NULL UNIQUE,
  CO2 float NOT NULL DEFAULT 0
);

CREATE TABLE Categorie
( CID SERIAL PRIMARY KEY,
  Nom varchar(25) NOT NULL UNIQUE
);

CREATE TABLE Plat
( PID SERIAL PRIMARY KEY,
  Nom varchar(50) NOT NULL UNIQUE,
  CO2 float NOT NULL DEFAULT 0,
  CID integer NOT NULL,
  CONSTRAINT FK_Plat FOREIGN KEY (CID) REFERENCES Categorie(CID)
);

CREATE TABLE CompositionPlat
( PID integer NOT NULL,
  IID integer NOT NULL,
  CONSTRAINT PK_Composition PRIMARY KEY(PID,IID),
  CONSTRAINT FK_CompositionIID FOREIGN KEY (IID) REFERENCES Ingredient(IID) ON DELETE CASCADE,
  CONSTRAINT FK_CompositionPlatPID FOREIGN KEY (PID) REFERENCES Plat(PID) ON DELETE CASCADE
);
```

Exemple de déclencheur modifiant des nuplets (2/2)

Sous PostgreSQL :

```
CREATE OR REPLACE FUNCTION FunctionTriggerCompositionPlat() RETURNS trigger AS
'DECLARE
 BEGIN
    UPDATE Plat SET CO2 = CO2 + (SELECT CO2 FROM Ingredient WHERE IID=NEW.IID) WHERE PID = NEW.PID ;
  RETURN NEW;
 END;'
LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_CompositionPlat
BEFORE INSERT ON CompositionPlat
FOR EACH ROW
EXECUTE PROCEDURE FunctionTriggerCompositionPlat();
```

Une instance de la relation Ingrédient

iid	nom	co2
1	legumes de saison	53.4
2	huile olive (1/2 c.s.)	18.2
3	poulet	774
4	riz	84.6
5	beurre	94.9

Une instance de la relation Plat après déclenchement du trigger :

pid	nom	co2	cid
1	legumes à la grecque	71.6	1
2	poulet au riz	953.5	2

Exemple de déclencheur testant une condition et insérant un nuplet (1/2)

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 testant une condition et insérant un nuplet (2/2)

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**
- **Exercice 1 de l'examen de janvier 2020**
- **Exercice 1 de l'examen de mai 2019**
- **Exercice 1 de l'examen de mai 2018**
- **Etc.**

Modification du schéma

```
ALTER TABLE table
    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	numsaison	idcandidat	nomcandidat	nomepreuve	numsemaine	datedebut	datefin	nomgateau	tablierbleu	elimine	classement	nbtabliersbleus	nbmeilleurs 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 fraiches
- Possibilité de rafraîchir les données :

```
REFRESH MATERIALIZED VIEW NomVue;
```

Différence vue et vue matérialisée (1/3)

```
CREATE VIEW AgeMoyen AS SELECT AVG(Age) FROM R2 ;

SELECT * FROM AgeMoyen ;

INSERT INTO R2 VALUES('Toto', 95, 'M');
INSERT INTO R2 VALUES('Tutu', 95, 'M');
INSERT INTO R2 VALUES('Titi', 95, 'M');

SELECT * FROM AgeMoyen ;
```

Query #2 **Execution time: 0ms**

avg
52.5000000000000000

Query #3 **Execution time: 1ms**

There are no results to be displayed.

Query #4 **Execution time: 0ms**

There are no results to be displayed.

Query #5 **Execution time: 0ms**

There are no results to be displayed.

Query #6 **Execution time: 0ms**

avg
61.0000000000000000

En cas de mise à jour, la vue (non matérialisée) est recalculée

Différence vue et vue matérialisée (2/3)

```
CREATE MATERIALIZED VIEW AgeMoyen AS SELECT AVG(Age) FROM R2 ;

SELECT * FROM AgeMoyen ;

INSERT INTO R2 VALUES('Toto', 95, 'M');
INSERT INTO R2 VALUES('Tutu', 95, 'M');
INSERT INTO R2 VALUES('Titi', 95, 'M');

SELECT * FROM AgeMoyen ;
```

Query #2 **Execution time: 0ms**

avg
52.5000000000000000

Query #3 **Execution time: 1ms**

There are no results to be displayed.

Query #4 **Execution time: 0ms**

There are no results to be displayed.

Query #5 **Execution time: 0ms**

There are no results to be displayed.

Query #6 **Execution time: 0ms**

avg
52.5000000000000000

En cas de mise à jour, la vue matérialisée n'est pas recalculée

Différence vue et vue matérialisée (3/3)

```
CREATE MATERIALIZED VIEW AgeMoyen AS SELECT AVG(Age) FROM R2 ;

SELECT * FROM AgeMoyen ;

INSERT INTO R2 VALUES('Toto', 95, 'M');
INSERT INTO R2 VALUES('Tutu', 95, 'M');
INSERT INTO R2 VALUES('Titi', 95, 'M');

REFRESH MATERIALIZED VIEW AgeMoyen;
SELECT * FROM AgeMoyen ;
```

avg
52.5000000000000000

Query #3 **Execution time: 0ms**

There are no results to be displayed.

Query #4 **Execution time: 1ms**

There are no results to be displayed.

Query #5 **Execution time: 0ms**

There are no results to be displayed.

Query #6 **Execution time: 2ms**

There are no results to be displayed.

Query #7 **Execution time: 0ms**

avg
61.0000000000000000

La vue matérialisée n'est mise à jour que si on le demande explicitement

CREATE SCHEMA

- **Schéma de base de données :** ensemble des déclarations décrivant une base de données au niveau logique, i.e. relations, vues, fonctions, déclencheurs etc.
- **Sous PostgreSQL,** il existe un schéma par défaut : `public`
- **Commande SQL pour créer un schéma** : **`CREATE SCHEMA nom_schema ;`**

```
CREATE SCHEMA TP ; -- pour créer un schéma nommé TP
```

- 2 options **pour utiliser un schéma** :
  - **Soit préfixer le nom des relations, vues et fonctions du nom du schéma - mais pas des déclencheurs** :

```
CREATE TABLE TP.Departement
(
 Departement_id     serial,
 Nom_Departement   varchar(25) NOT NULL,
 CONSTRAINT UN_Nom_Departement UNIQUE (nom_departement),
 CONSTRAINT PK_Departement PRIMARY KEY(Departement_ID)
);

CREATE OR REPLACE VIEW TP.Email_Etudiant
AS SELECT Nom, Prenom, Email FROM TP.Etudiant;

CREATE OR REPLACE FUNCTION TP.GetDepartement_ID(text) RETURNS integer AS
'SELECT Departement_ID FROM TP.Departement WHERE Nom_Departement = $1'
LANGUAGE SQL;
```

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

INSERT INTO TP.Departement(Nom_Departement) VALUES ('MIDO');

 SELECT * FROM TP.Departement ;
```

SET SCHEMA

- **Soit possibilité de positionner sur un schéma spécifique par la commande :**
  `SET SCHEMA nom_schema ;`
- **Sous PostgreSQL, pour se positionner sur un schéma** :
  `SET search_path TO nom_schema ;`
- Inutile ensuite de préfixer les noms des relations etc. par le nom du schéma

```
CREATE SCHEMA TP ; -- pour créer un schéma nommé TP

SET search_path TO TP; -- Pour se positionner sur le schéma TP ;

CREATE TABLE Departement
(
 Departement_id     serial,
 Nom_Departement   varchar(25) NOT NULL,
 CONSTRAINT UN_Nom_Departement UNIQUE (nom_departement),
 CONSTRAINT PK_Departement PRIMARY KEY(Departement_ID)
);

INSERT INTO Departement(Nom_Departement) VALUES ('MIDO');
```

```
SET search_path TO TP;

SELECT * FROM Departement ;
```

-  sous *dbfiddle* il faut faire la commande `SET` du côté de la création du schéma et du côté des requêtes d'interrogation.

Mélanger plusieurs schémas

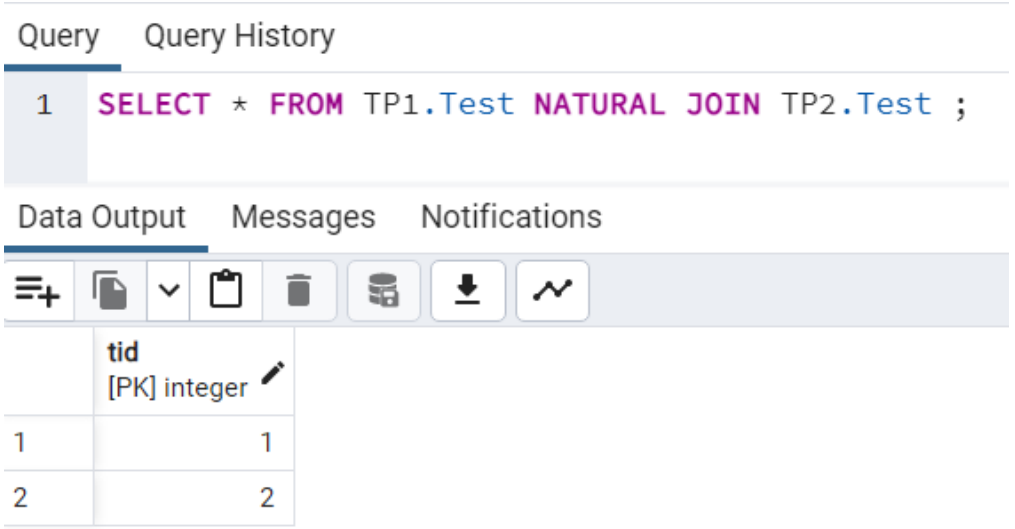
```
CREATE SCHEMA TP1 ;

CREATE TABLE TP1.Test
(
  TID integer PRIMARY KEY
) ;

CREATE SCHEMA TP2 ;

CREATE TABLE TP2.Test
(
  TID integer PRIMARY KEY
) ;

INSERT INTO TP1.test VALUES (1),(2),(3) ;
INSERT INTO TP2.test VALUES (1),(2) ;
```



Interroger le schéma d'une base de données

```sql
-- Pour voir la liste des tables stockées dans la base
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public';

-- Pour voir la liste des atttributs d'une table
SELECT column_name
FROM information_schema.columns
WHERE table_name ='etudiant';

-- Pour voir la liste des tables de la base
-- avec leurs attributs et le type des attributs
SELECT table_name, column_name, data_type
FROM information_schema.columns
WHERE table_schema = 'public';
```

```sql
-- Pour voir la liste des vues
select * from information_schema.views
where table_schema='public'
```

```sql
-- Pour lister les contraintes des tables
select *
from information_schema.table_constraints
where table_schema='public';
```

Chap VII – Passage au relationnel

Pour créer une base de données :

<u>Étape N° 1</u> : 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)

<u>Étape N° 2</u> : 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

CREATE SCHEMA

- **Schéma de base de données :** ensemble des déclarations décrivant une base de données au niveau logique, i.e. relations, vues, fonctions, déclencheurs etc.
- **Sous PostgreSQL,** il existe un schéma par défaut : `public`
- **Commande SQL pour créer un schéma** : **`CREATE SCHEMA nom_schema ;`**

```
CREATE SCHEMA TP ; -- pour créer un schéma nommé TP
```

- 2 options **pour utiliser un schéma** :
  - **Soit préfixer le nom des relations, vues et fonctions du nom du schéma - mais pas des déclencheurs** :

```
CREATE TABLE TP.Departement
(
 Departement_id     serial,
 Nom_Departement   varchar(25) NOT NULL,
 CONSTRAINT UN_Nom_Departement UNIQUE (nom_departement),
 CONSTRAINT PK_Departement PRIMARY KEY(Departement_ID)
);

CREATE OR REPLACE VIEW TP.Email_Etudiant
AS SELECT Nom, Prenom, Email FROM TP.Etudiant;

CREATE OR REPLACE FUNCTION TP.GetDepartement_ID(text) RETURNS integer AS
'SELECT Departement_ID FROM TP.Departement WHERE Nom_Departement = $1'
LANGUAGE SQL;
```

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

INSERT INTO TP.Departement(Nom_Departement) VALUES ('MIDO');

 SELECT * FROM TP.Departement ;
```

SET SCHEMA

- **Soit possibilité de positionner sur un schéma spécifique par la commande :**
  `SET SCHEMA nom_schema ;`
- **Sous PostgreSQL, pour se positionner sur un schéma** :
  `SET search_path TO nom_schema ;`
- Inutile ensuite de préfixer les noms des relations etc. par le nom du schéma

```
CREATE SCHEMA TP ; -- pour créer un schéma nommé TP

SET search_path TO TP; -- Pour se positionner sur le schéma TP ;

CREATE TABLE Departement
(
 Departement_id     serial,
 Nom_Departement   varchar(25) NOT NULL,
 CONSTRAINT UN_Nom_Departement UNIQUE (nom_departement),
 CONSTRAINT PK_Departement PRIMARY KEY(Departement_ID)
);

INSERT INTO Departement(Nom_Departement) VALUES ('MIDO');
```

```
SET search_path TO TP;

SELECT * FROM Departement ;
```

-  sous *dbfiddle* il faut faire la commande `SET` du côté de la création du schéma et du côté des requêtes d'interrogation.

Mélanger plusieurs schémas

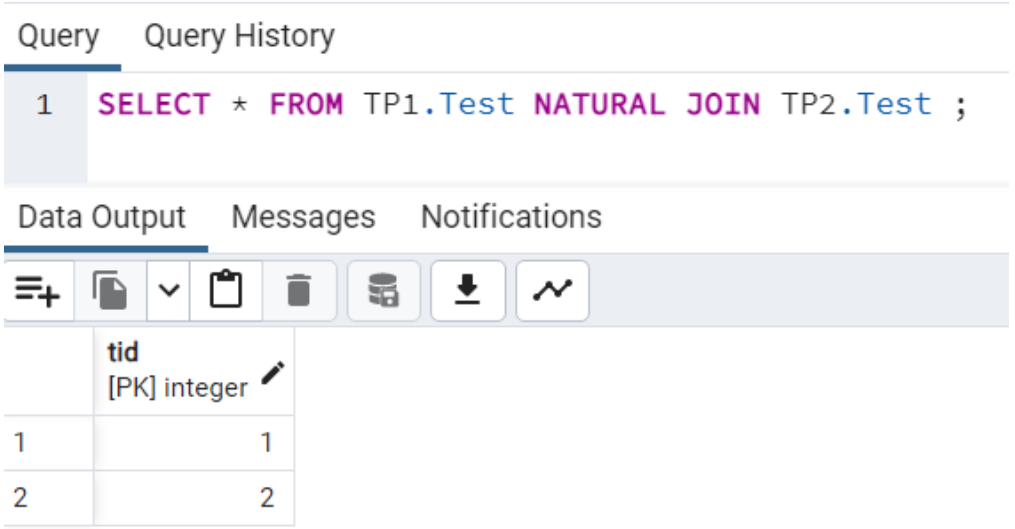
```
CREATE SCHEMA TP1 ;

CREATE TABLE TP1.Test
(
  TID integer PRIMARY KEY
) ;

CREATE SCHEMA TP2 ;

CREATE TABLE TP2.Test
(
  TID integer PRIMARY KEY
) ;

INSERT INTO TP1.test VALUES (1),(2),(3) ;
INSERT INTO TP2.test VALUES (1),(2) ;
```



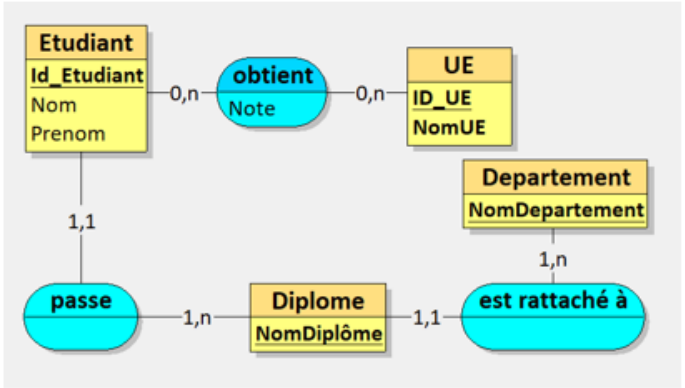
Chap. VII – Modélisation et passage au relationnel

Monde ou problème à modéliser

Analyse conceptuelle : analyse humaine

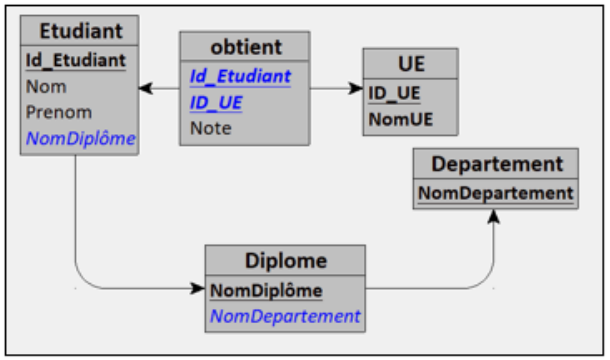
Modèle conceptuel de données (MCD)

Modèle Entité-Association (formalisme)



Règle de dérivations

Modèle relationnel



Génération de scripts

Script SQL

```
CREATE TABLE Departement(
  NomDepartement VARCHAR(50),
  PRIMARY KEY(NomDepartement)
);

CREATE TABLE UE(
  ID_UE COUNTER,
  NomUE VARCHAR(50) NOT NULL,
  PRIMARY KEY(ID_UE),
  UNIQUE(NomUE)
);

CREATE TABLE Diplome(
  NomDiplôme VARCHAR(50),
  NomDepartement VARCHAR(50) NOT NULL,
  PRIMARY KEY(NomDiplôme),
  FOREIGN KEY(NomDepartement) REFERENCES Departement(NomDepartement)
);

CREATE TABLE Etudiant(
  Id_Etudiant COUNTER,
  Nom VARCHAR(50) NOT NULL,
  Prenom VARCHAR(50) NOT NULL,
  NomDiplôme VARCHAR(50) NOT NULL,
  PRIMARY KEY(Id_Etudiant),
  FOREIGN KEY(NomDiplôme) REFERENCES Diplome(NomDiplôme)
);

CREATE TABLE obtient(
```

Base de données relationnelles

Exécution

Requêtes

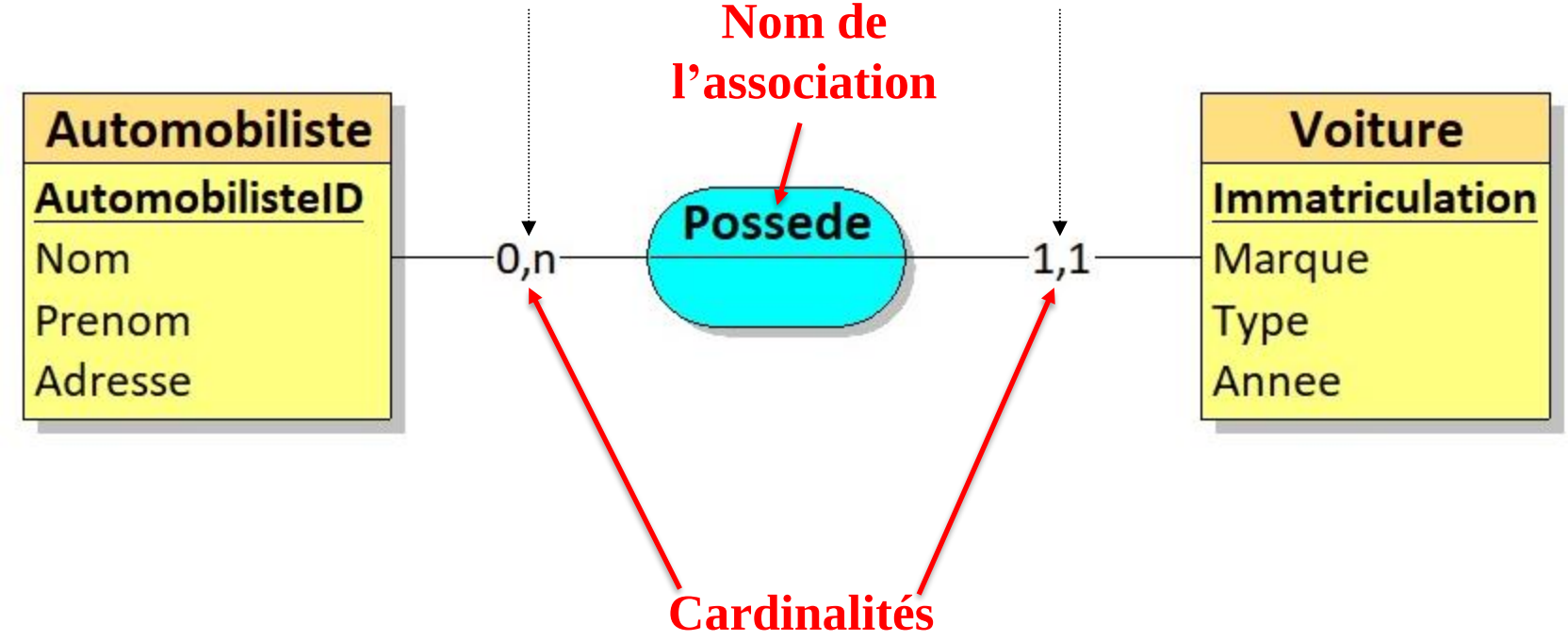
Résultat

Utilisatrice /teur

Modélisation Entité/Association (Format Merise)

Un automobiliste possède entre zéro et n voitures

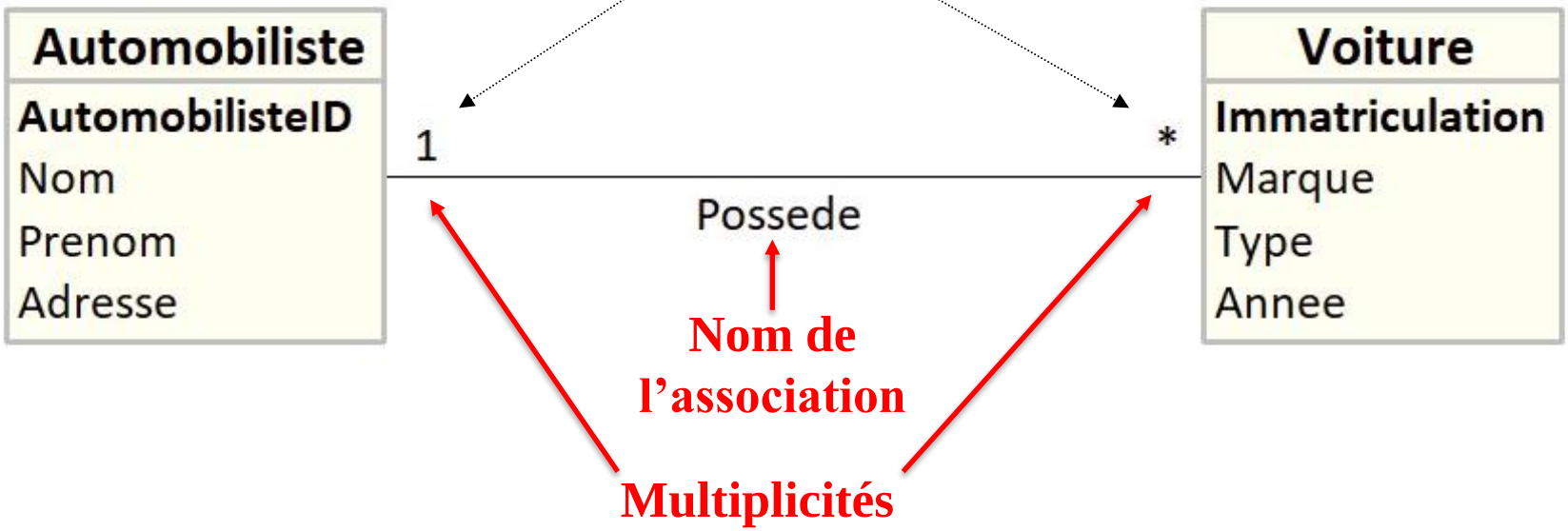
Une voiture a un et un seul propriétaire



Modélisation UML

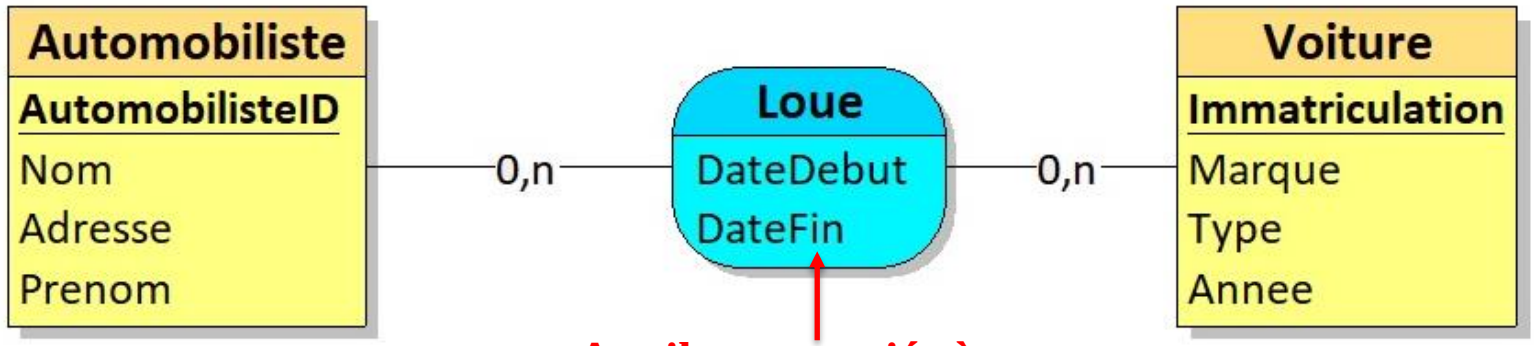
Un automobiliste possède entre zéro et n voitures

Une voiture a un et un seul propriétaire

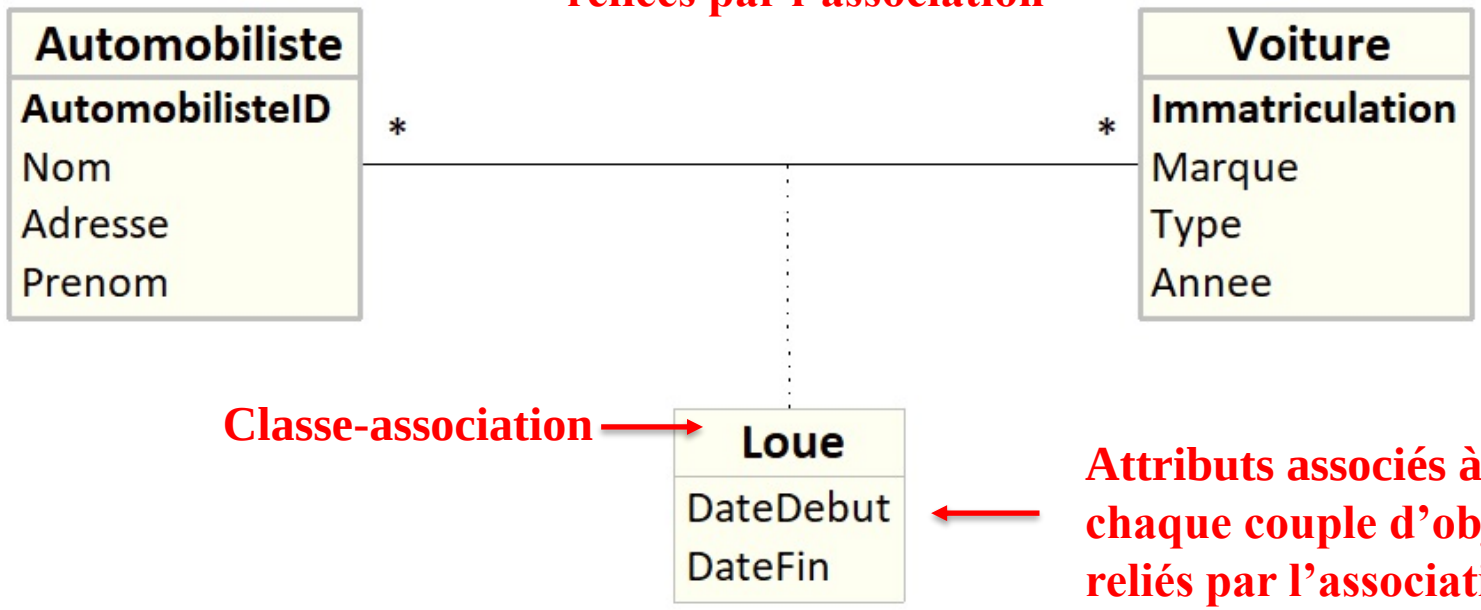


Ensembles d'associations avec des attributs

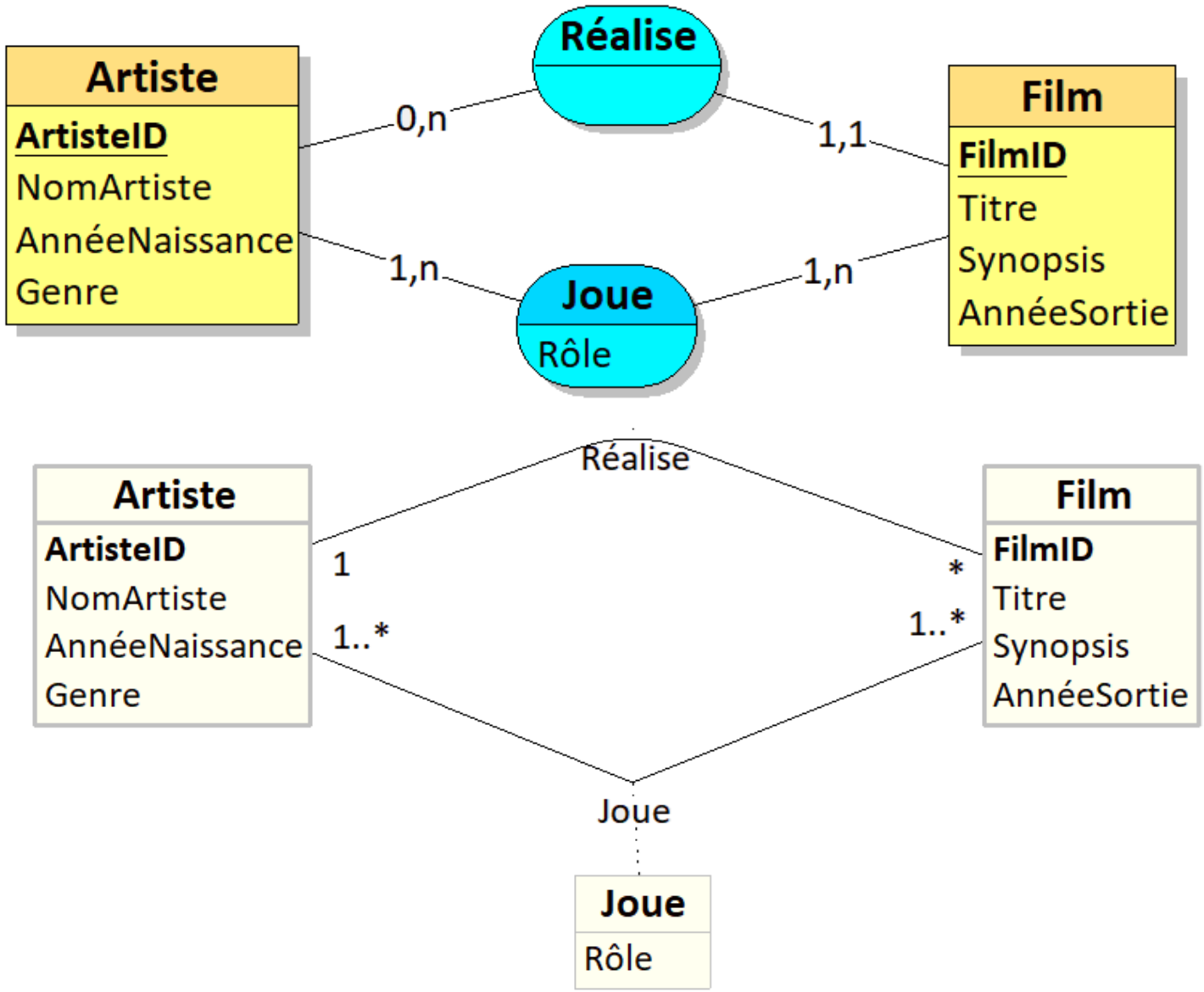
Modèle E/A



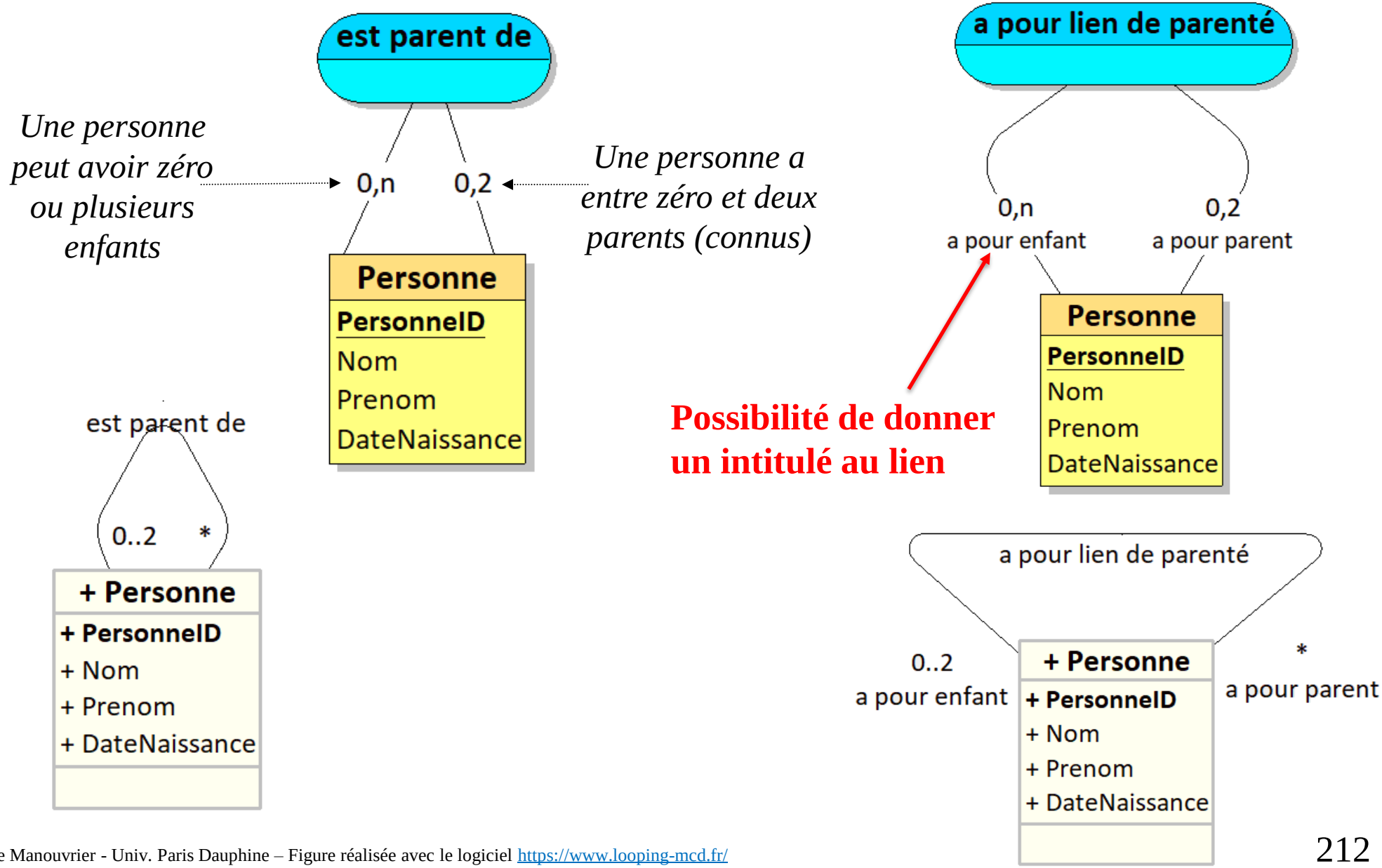
Modèle UML



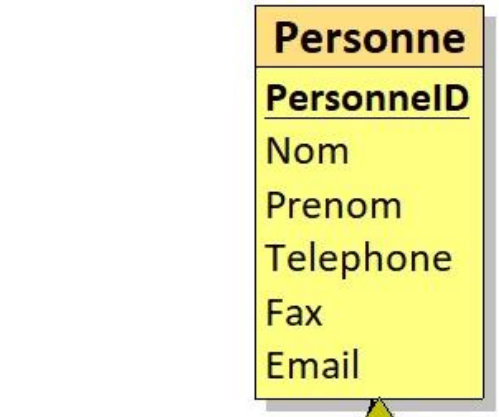
Plusieurs associations entre un même ensemble d'entités /classes



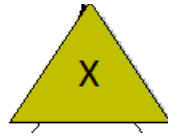
Association réflexive



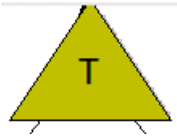
Généralisation/Spécialisation (E/A - Merise)



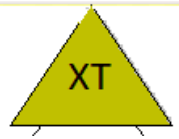
Ensemble d'entités généralisantes



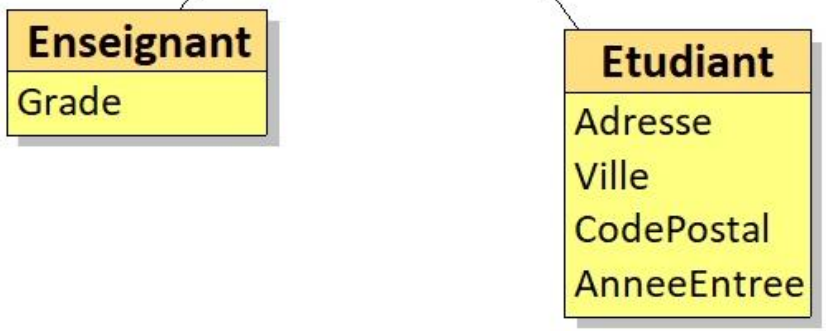
Contrainte d'exclusion : aucune entité ne peut se trouver dans plusieurs spécialisations à la fois



Contrainte de totalité : chaque occurrence à appartenir au moins à une des entités spécialisées

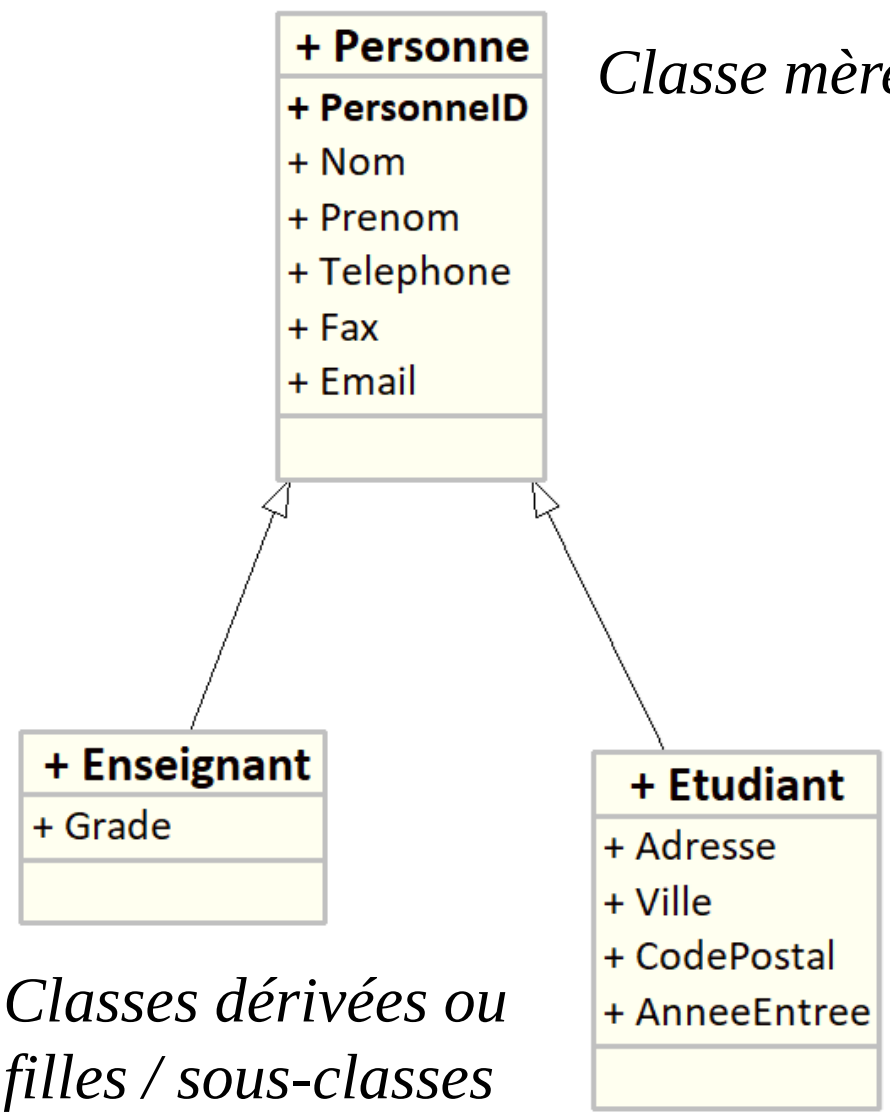


Contrainte de partition : exclusion et totalité



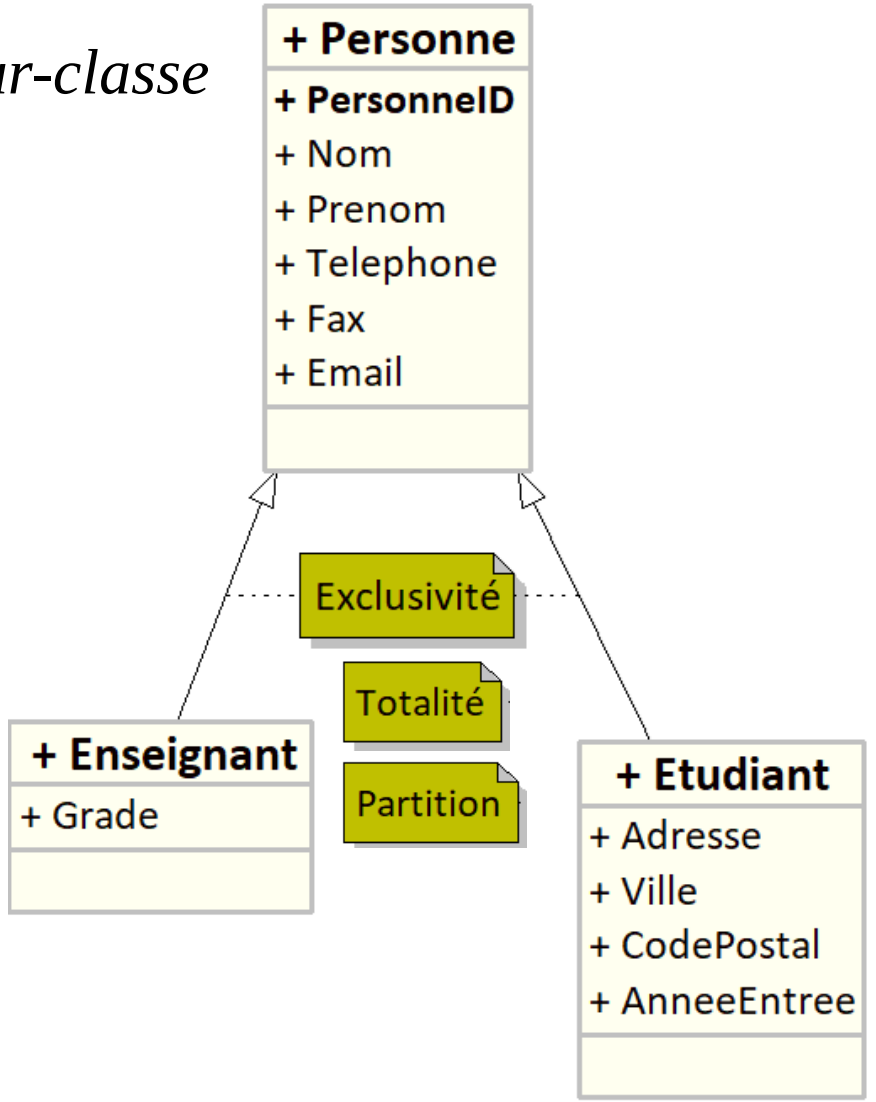
Ensemble d'entités spécialisées

Héritage (UML)



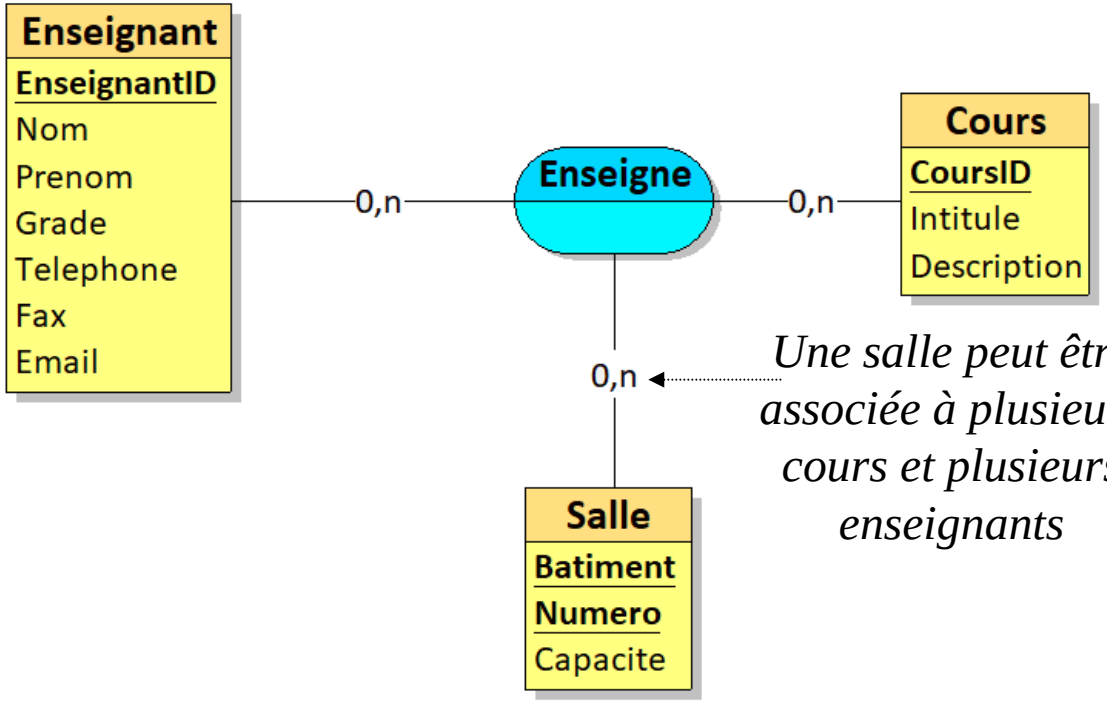
Classe mère / Sur-classe

Classes dérivées ou filles / sous-classes



Exemple d'association n-aire

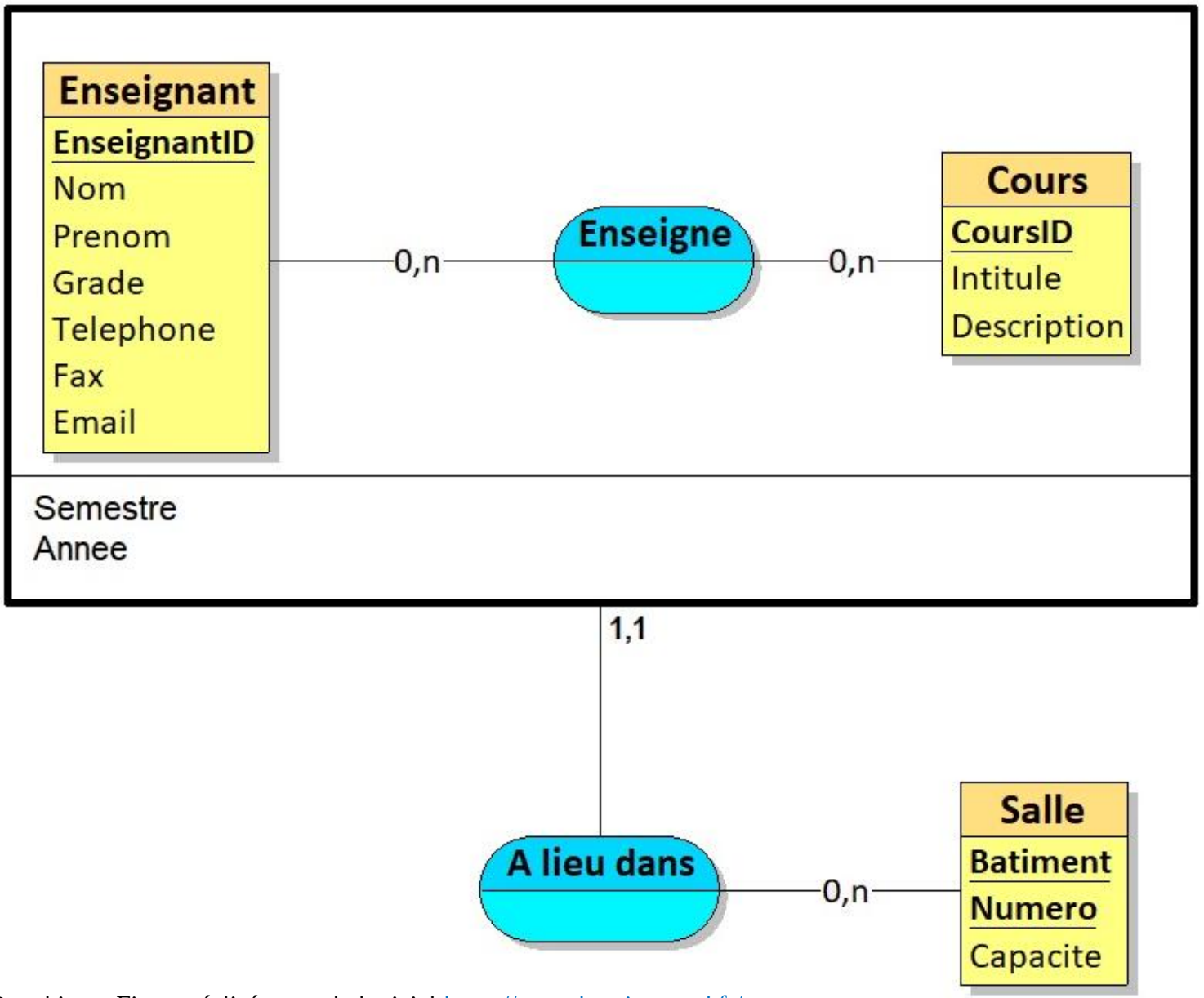
Exemple d'association ternaire :



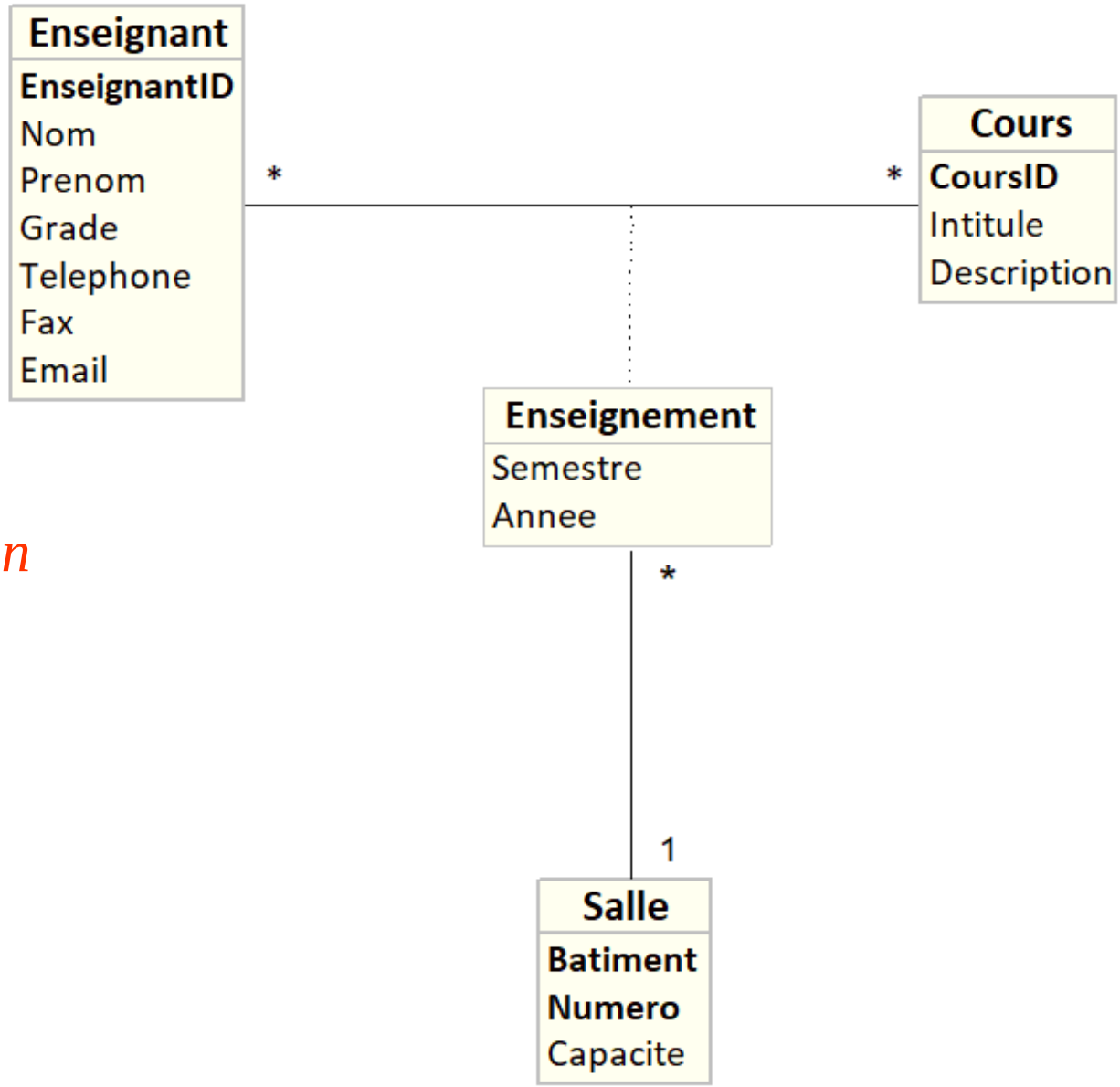
Souvent complexe à interpréter et ne permet pas de tout modéliser (ex. si une même salle doit toujours être associée à un enseignant pour un cours donné).
*Préférer l'**agrégations d'associations** ou **décomposer l'association n-aire en associations binaires.***

Agrégat (E/A - Merise)

On peut nommer ou non l'agrégat

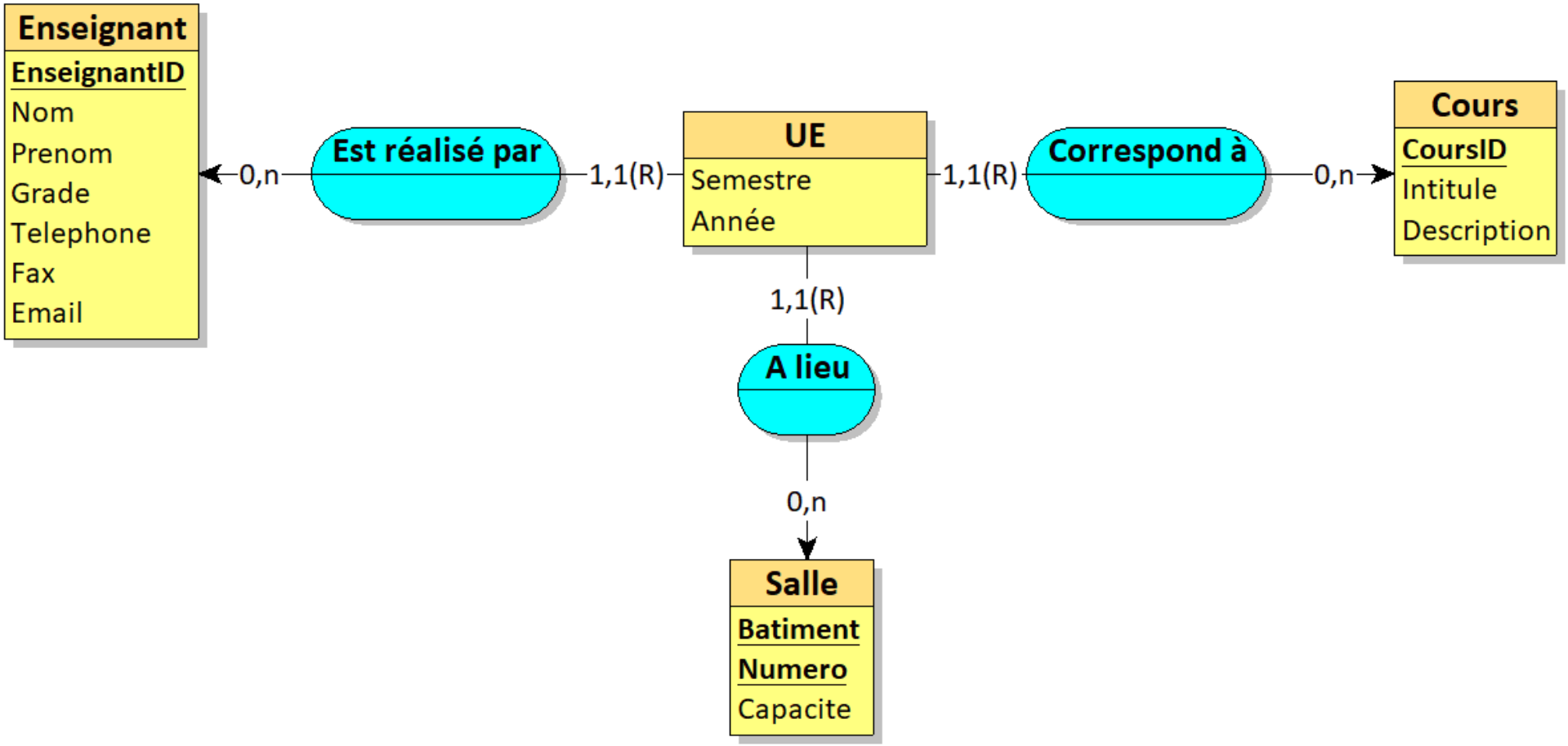


Classe-Association (UML)

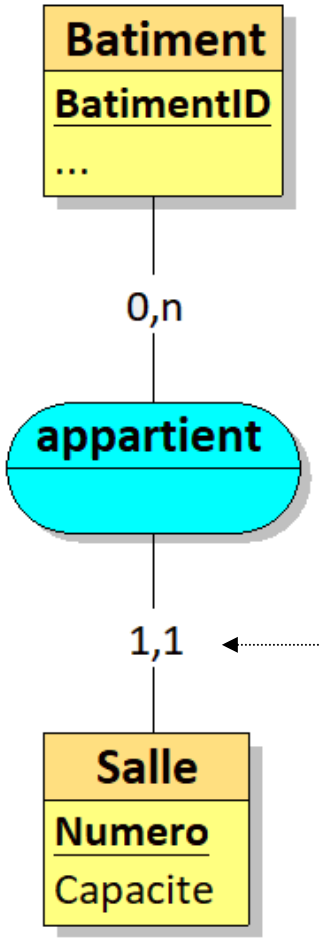


On peut nommer ou non la classe-association

Décomposition d'une association n-aire

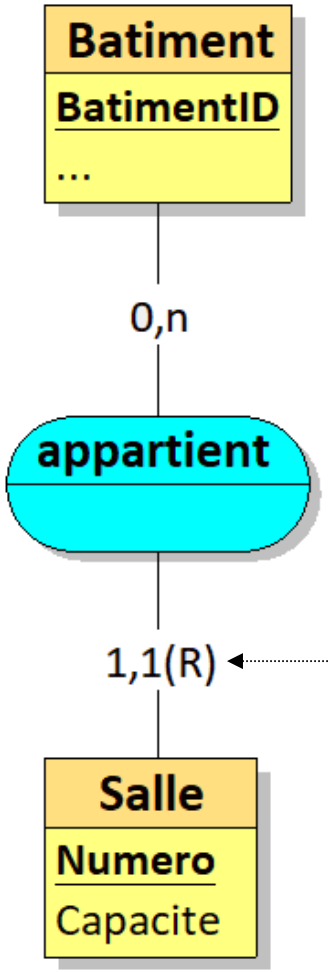


Entité Faible ou Relative



Chaque salle a un numéro unique quel que soit le bâtiment

Ex. Une seule Salle 1, une seule Salle 2 etc.

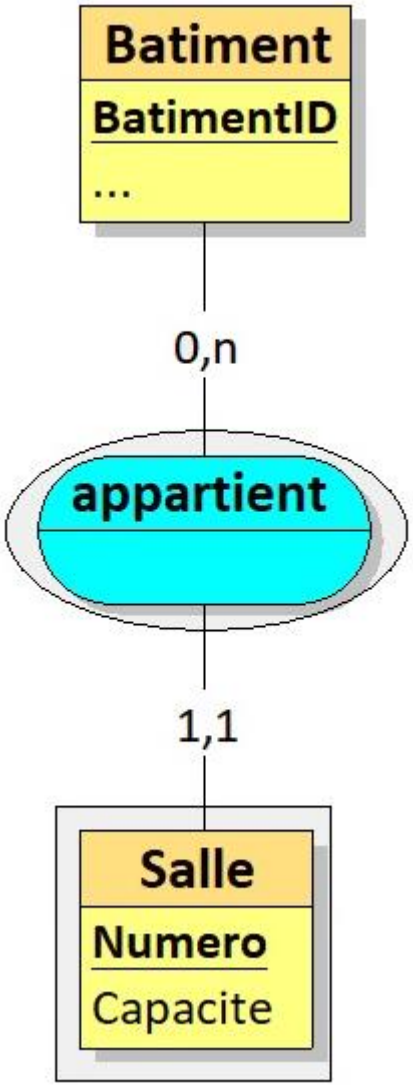


Chaque salle a un numéro unique dans un bâtiment donné

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

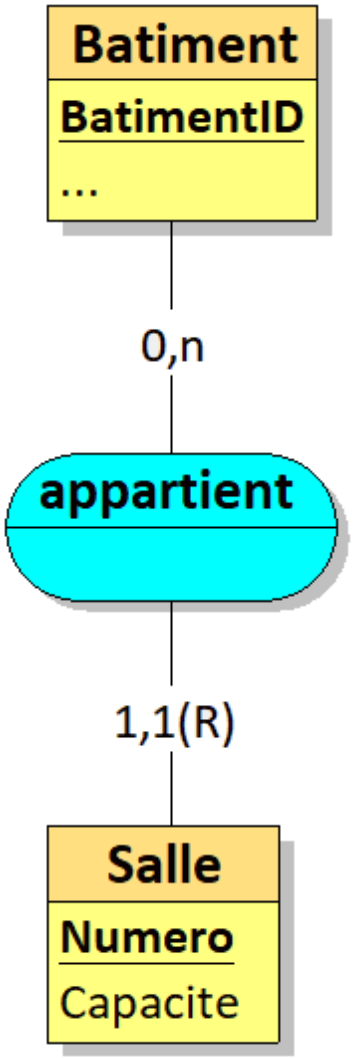
Entité Faible ou Relative (E/A - Merise)



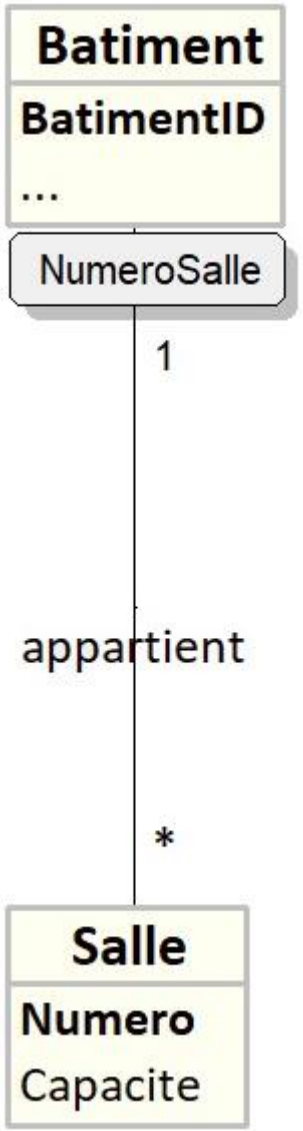
Chaque salle a un numéro unique dans un bâtiment donné

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

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

Voiture
Immatriculation
Marque
Type
Annee

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
EnseignantID
Nom
Prenom
Grade
Telephone
Fax
Email

EnseignantID → Nom, Prénom, Position …

Nom, Prénom, Grade, … → EnseignantID

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

Telephone → EnseignantID

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) **Modification de relations créées à l'étape 1**

   b) **Ajout de nouvelles relations**

   En fonction des cardinalités/multiplicités

Passage au relationnel (2/2)

Transformation des ensembles d'entités :

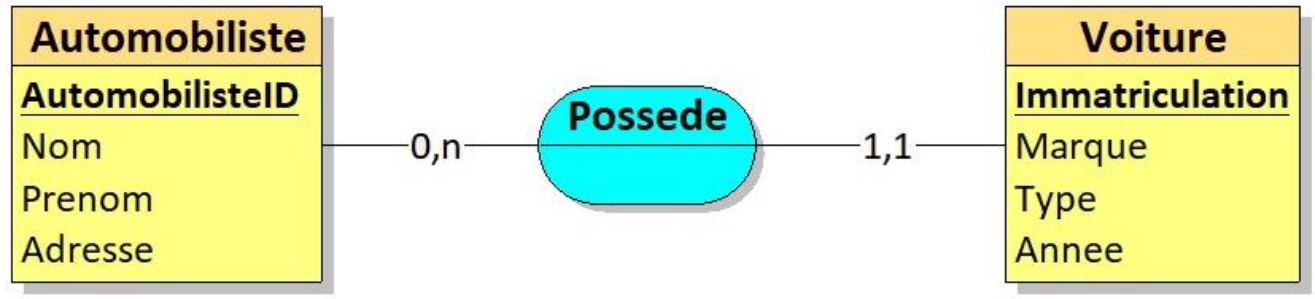
- **chaque ensemble d'entités/classes *E* ⇒**
  - une relation R avec les mêmes attributs que E
  - l'identificateur de E devient la clé primaire de R
- **chaque ensemble d'entités faibles/associations qualifiées *E* ⇒**

  une relation R qui comprend tous les attributs de E +

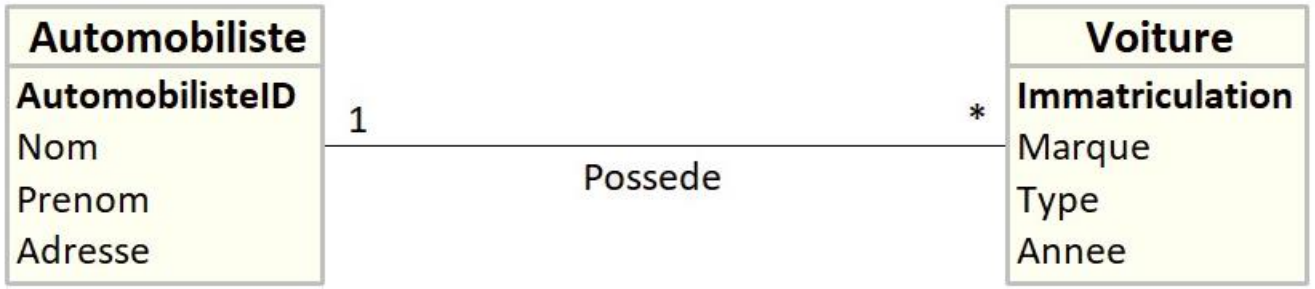
  l'identificateur de l'ensemble d'entités fortes/classes associé(e)s
- **généralisation-spécialisation/héritage ⇒**
  - l'ensemble d'entités généralisante/classe mère *E* ⇒ une relation *R*
  - chaque ensemble d'entités spécialisé/classe fille E_i
    ⇒ une relation R_i dans laquelle l'identifiant est 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'entités ou classes

Modèle E/A



Modèle UML



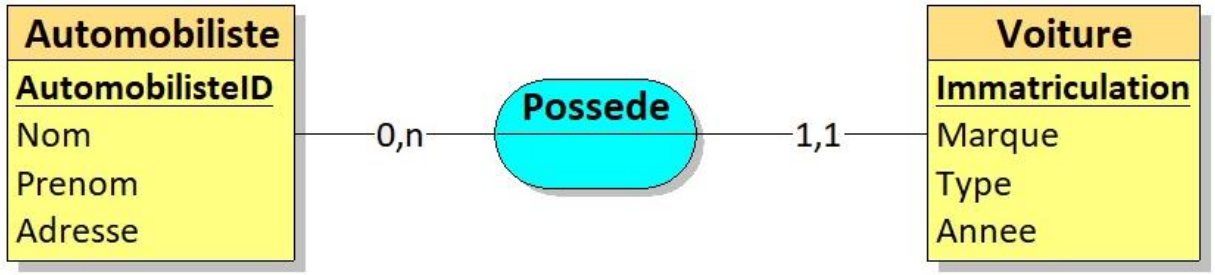
Etape 1 : une relation par ensemble d'entités / classe

Automobiliste (AutomobilisteID, Nom, Prénom, Adresse)

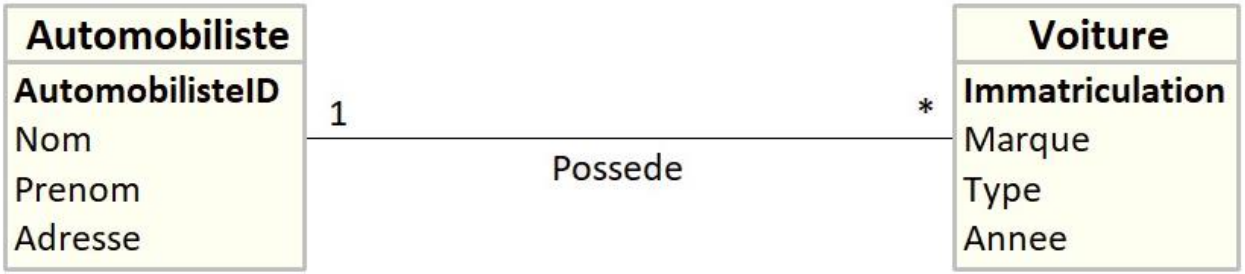
Voiture (Immatriculation, Marque, Type, Puissance, Année)

Transformation des ensembles d'associations

Modèle E/A



Modèle UML

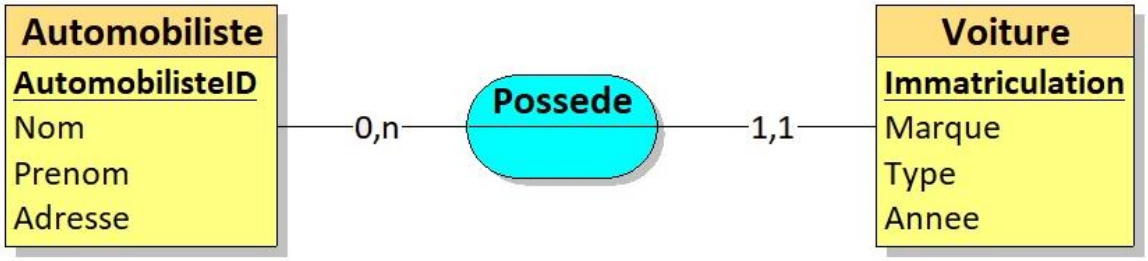


Etape 2 : transformation de l'association par une clé étrangère

Automobiliste (AutomobilisteID, Nom, Prénom, Adresse)
Voiture (Immatriculation, Marque, Puissance, Type, Annee,
#AutomobilisteID)
#AutomobilisteID fait référence à la clé primaire de Automobiliste
Possibilité d'appeler la clé étrangère ProprioID

Transformation d'une association en clé étrangère

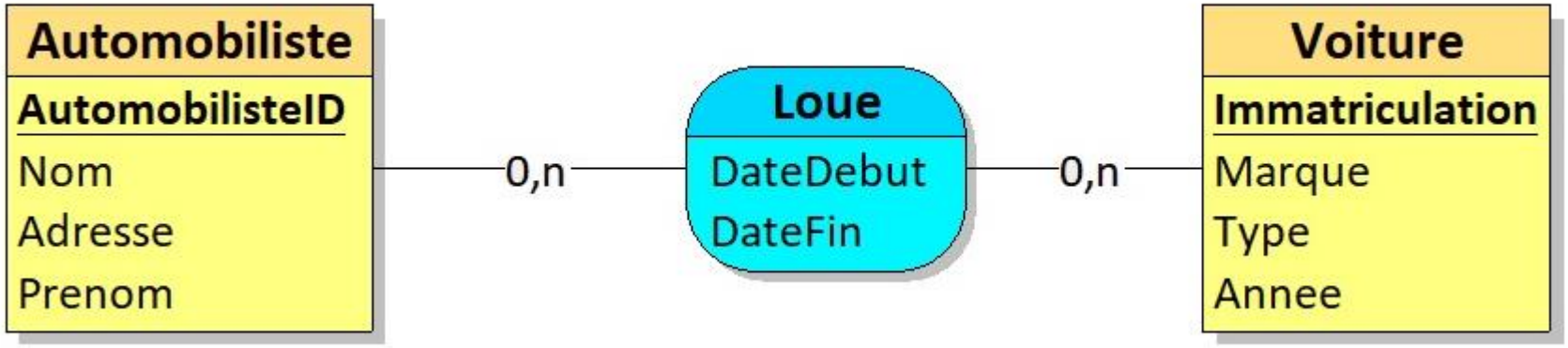
Modèle E/A



Modèle relationnel sous looping

AutomobilisteID est une clé étrangère faisant référence à la clé primaire de Automobiliste
Possibilité d'appeler la clé étrangère ProprioID

Transformation des ensembles d'associations E/A par ajout d'une nouvelle relation



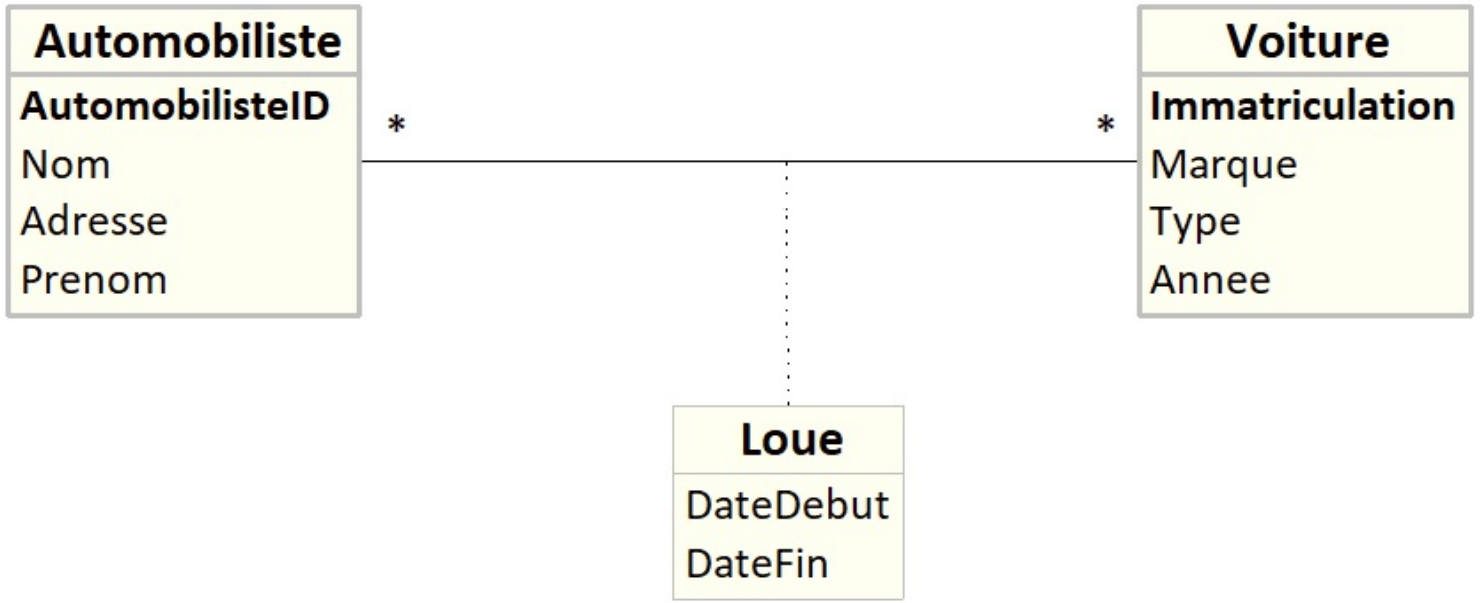
Automobiliste (AutomobilisteID, Nom, Prenom, Adresse)

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

Location (#AutomobilisteID, #Immatriculation, DateDebut, DateFin)
ou ***Location*** (LocID, #AutomobilisteID, #Immatriculation, DateDebut, DateFin)

#AutomobilisteID fait référence à la clé primaire de Automobiliste
#Immatriculation fait référence à la clé primaire de Voiture
Avec la clé artificielle LocID : contrainte d'unicité sur le triplet (#AutomobilisteID, #Immatriculation, DateDebut)

Transformation des associations UML par ajout d'une nouvelle relation



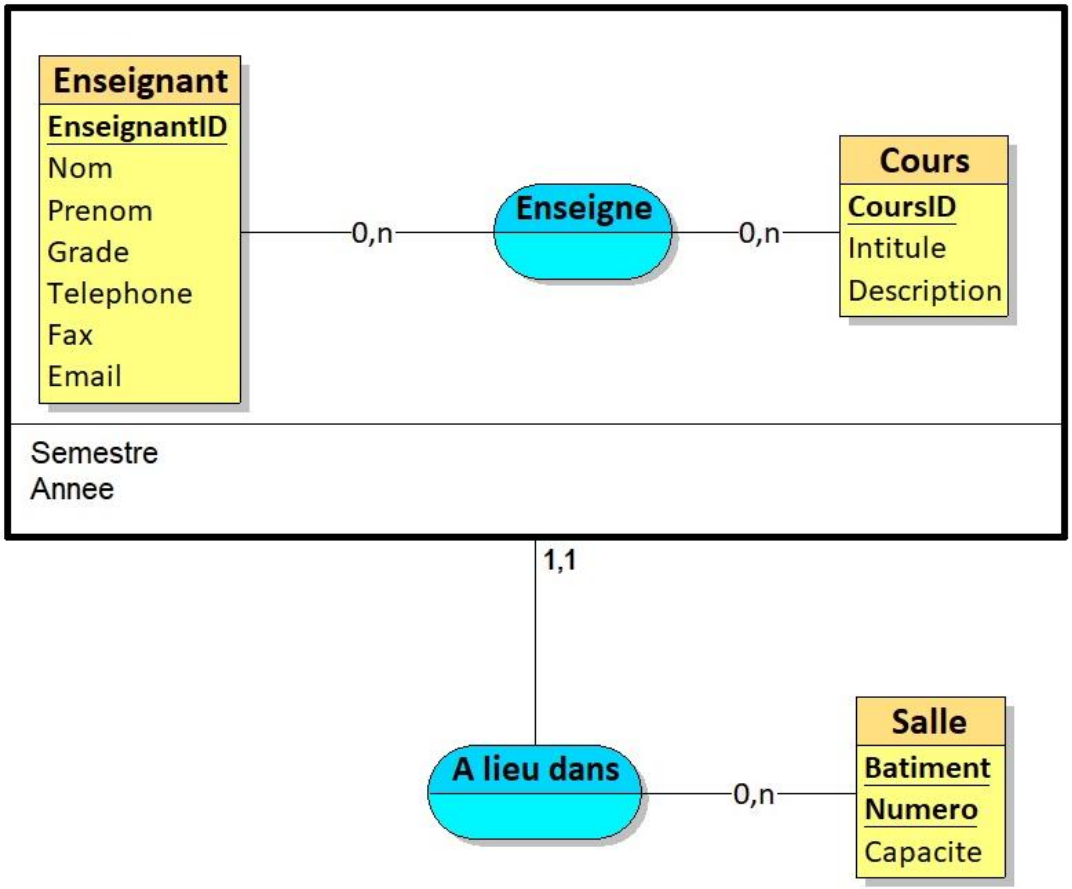
Automobiliste (AutomobilisteID, Nom, Prenom, Adresse)

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

Location (#AutomobilisteID, #Immatriculation, DateDebut, DateFin)
ou ***Location*** (LocID, #AutomobilisteID, #Immatriculation, DateDebut, DateFin)

#AutomobilisteID fait référence à la clé primaire de Automobiliste
#Immatriculation fait référence à la clé primaire de Voiture
Avec la clé artificielle LocID : contrainte d'unicité sur le triplet (#AutomobilisteID, #Immatriculation, DateDebut)

Transformation des agrégats / classes associations



Enseignant (EnseignantID, Nom,Prenom, ...)

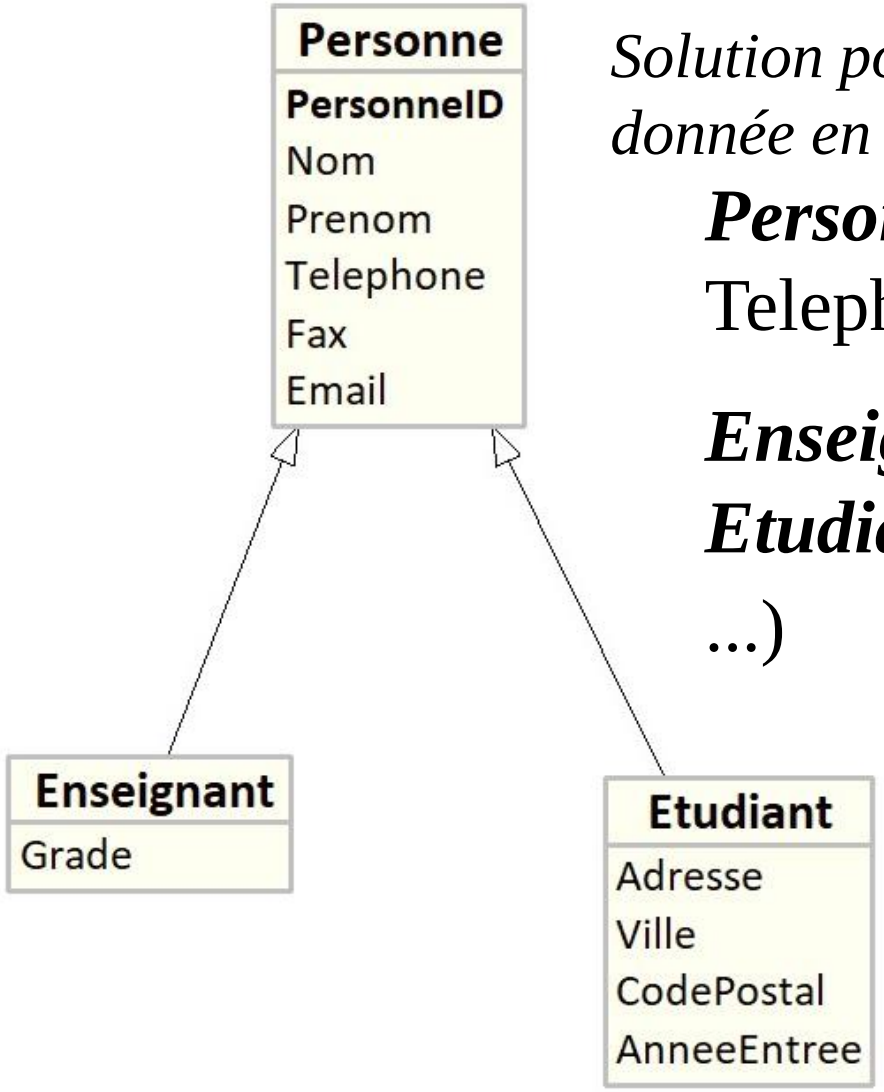
Cours (CoursID, Intitule, ...)

Enseignement (EnseignementID,#EnseignantID, #CoursID, Semestre,Année, #Batiment, #Numero)

Salle (Batiment,Numero, Capacite)

NB *(#Batiment, #Numero) dans Enseignement fait référence à la clé primaire de Salle*

Transformation des Généralisation-Spécialisation / Héritage



Solution possible (une autre sera donnée en cours) :

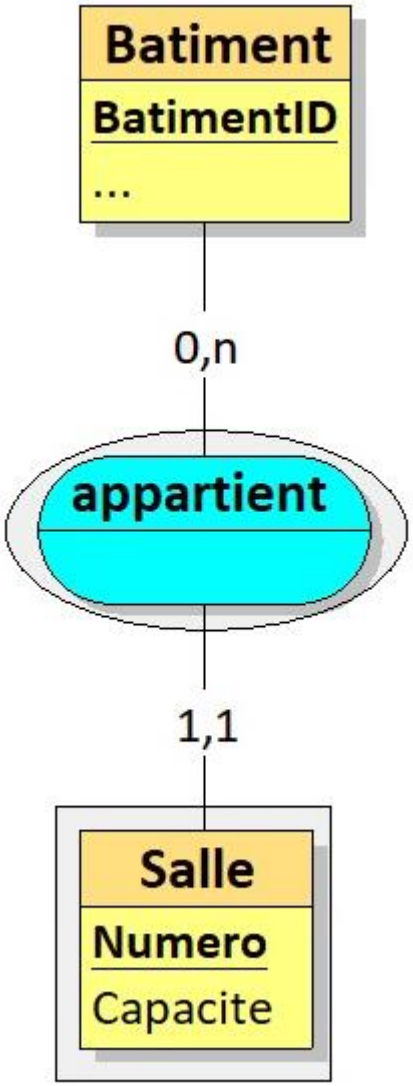
Personne (PersonneID, Nom,Prenom, Telephone ...)

Enseignant (#PersonneID, Grade)
Etudiant (#PersonneID, Adresse, Ville ...)

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

Possibilité de les appeler #EnseignantID et #EtudiantID

Transformation des entités faibles E/A



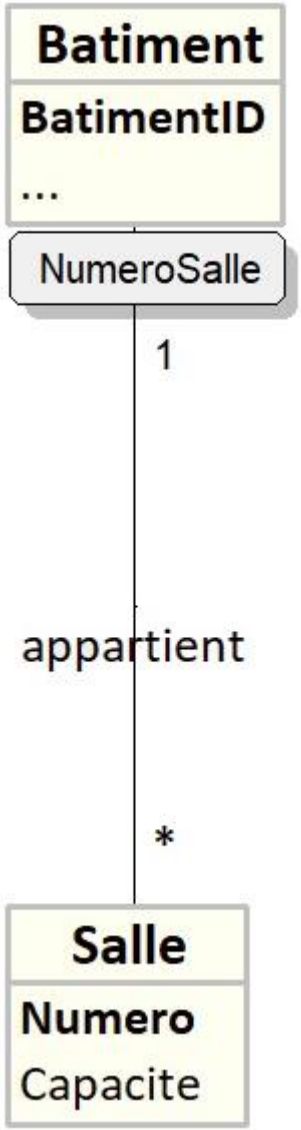
Bâtiment (Batiment_ID, ...)

Salle (Numero, #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, ...)

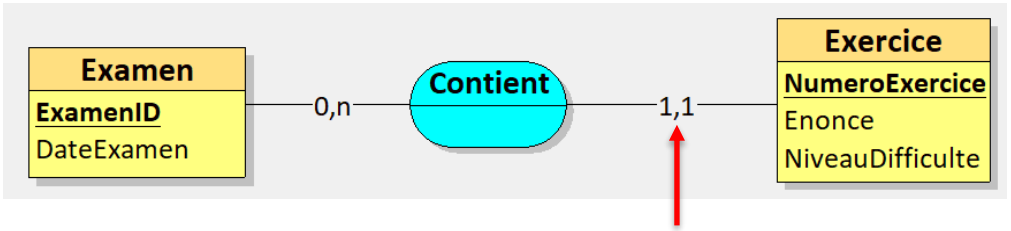
Salle (Numero, #Batiment_ID, Capacité)

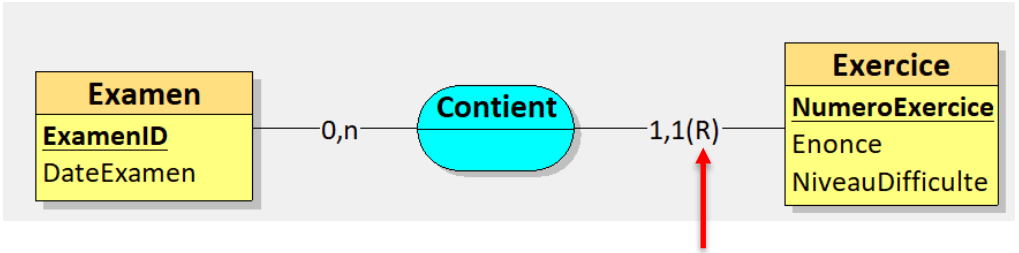
NB : *Une salle est identifiée par le couple (Numéro,#Bâtiment_ID) ;*

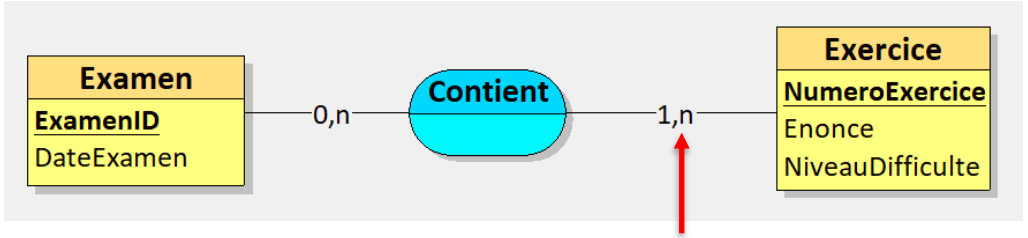
#Bâtiment_ID fait référence à Bâtiment_ID de Bâtiment

Transformation des associations en fonction des cardinalités

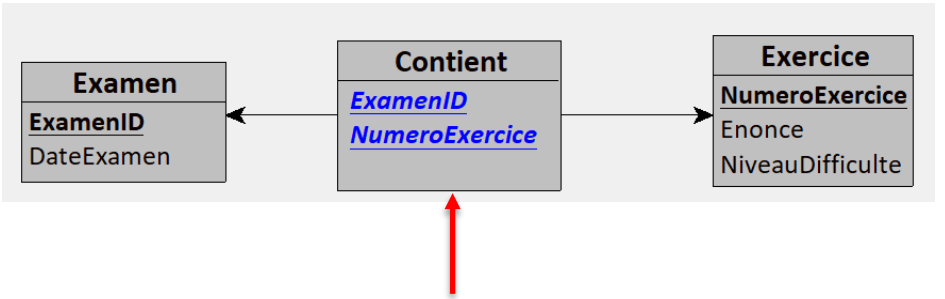
Modèle Entité - Association





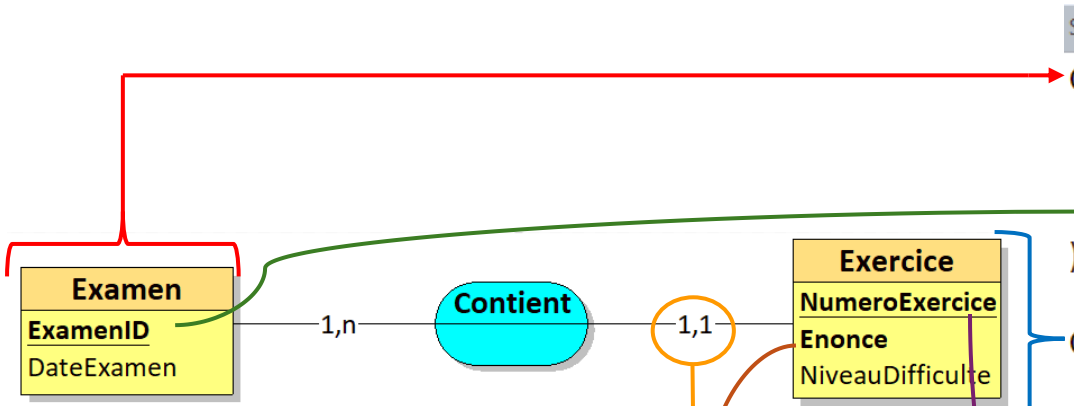


Modèle Relationnel sous Looping



Attribut en bleu = clé étrangère

Script SQL généré par Looping : Exemple 1

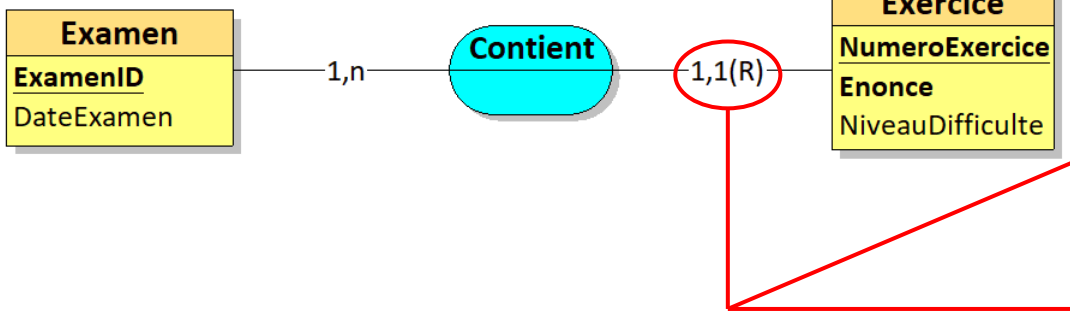


SQL

```
CREATE TABLE Examen(
   ExamenID INT,
   DateExamen VARCHAR(50) NOT NULL,
   PRIMARY KEY(ExamenID)
);

CREATE TABLE Exercice(
   NumeroExercice INT,
   Enonce VARCHAR(50) NOT NULL,
   NiveauDifficulte VARCHAR(50),
   ExamenID INT NOT NULL,
   PRIMARY KEY(NumeroExercice),
   UNIQUE(Enonce),
   FOREIGN KEY(ExamenID) REFERENCES Examen(ExamenID)
);
```

Script SQL généré par Looping: Exemple 2

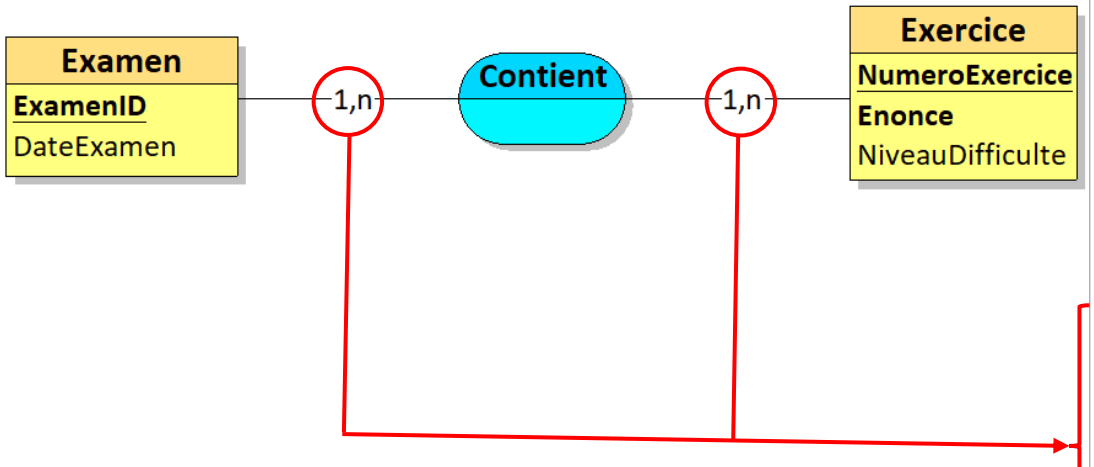


SQL

```
CREATE TABLE Examen(
  ExamenID INT,
  DateExamen VARCHAR(50) NOT NULL,
  PRIMARY KEY(ExamenID)
);

CREATE TABLE Exercice(
  ExamenID INT,
  NumeroExercice INT,
  Enonce VARCHAR(50) NOT NULL,
  NiveauDifficulte VARCHAR(50),
  PRIMARY KEY(ExamenID, NumeroExercice),
  UNIQUE(Enonce),
  FOREIGN KEY(ExamenID) REFERENCES Examen(ExamenID)
);
```

Script SQL généré par *looping* : Exemple 3

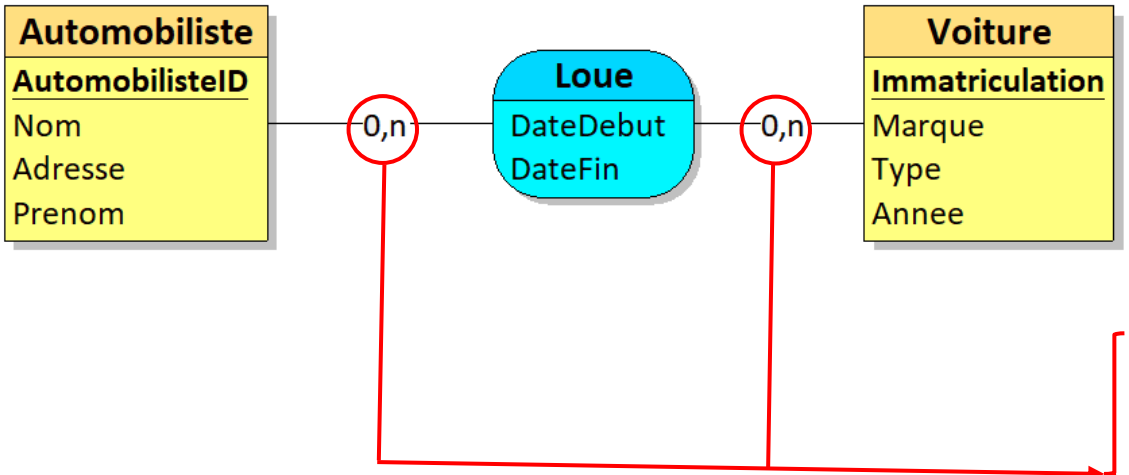



```
SQL
CREATE TABLE Examen(
  ExamenID INT,
  DateExamen VARCHAR(50) NOT NULL,
  PRIMARY KEY(ExamenID)
);

CREATE TABLE Exercice(
  NumeroExercice INT,
  Enonce VARCHAR(50) NOT NULL,
  NiveauDifficulte VARCHAR(50),
  PRIMARY KEY(NumeroExercice),
  UNIQUE(Enonce)
);

CREATE TABLE Contient(
  ExamenID INT,
  NumeroExercice INT,
  PRIMARY KEY(ExamenID, NumeroExercice),
  FOREIGN KEY(ExamenID) REFERENCES Examen(ExamenID),
  FOREIGN KEY(NumeroExercice) REFERENCES Exercice(NumeroExercice)
);
```

Script SQL généré par *looping* : Exemple 4




```sql
CREATE TABLE Automobiliste(
   AutomobilisteID INT,
   Nom VARCHAR(50) NOT NULL,
   Adresse VARCHAR(50),
   Prenom VARCHAR(50) NOT NULL,
   PRIMARY KEY(AutomobilisteID)
);

CREATE TABLE Voiture(
   Immatriculation INT,
   Marque VARCHAR(50) NOT NULL,
   Type VARCHAR(50),
   Annee DATE NOT NULL,
   PRIMARY KEY(Immatriculation)
);

CREATE TABLE Loue(
   AutomobilisteID INT,
   Immatriculation INT,
   DateDebut DATE NOT NULL,
   DateFin DATE NOT NULL,
   PRIMARY KEY(AutomobilisteID, Immatriculation),
   FOREIGN KEY(AutomobilisteID) REFERENCES Automobiliste(AutomobilisteID),
   FOREIGN KEY(Immatriculation) REFERENCES Voiture(Immatriculation)
);
```

Dépendances fonctionnelles

Ne pas oublier de définir les DF :

- ***Automobiliste*** (AutomobilisteID, Nom, Prénom, Adresse)

  AutomobilisteID → Nom, Prénom, Adresse

- ***Voiture*** (Immatriculation, Marque, Puissance, Type, Année, #ProprioID)

  Immatriculation → Marque, Type, Puissance, Année

  Type → Marque

  Immatriculation → ProprioID ProprioID ↛ Immatriculation

- ***Location*** (#AutomobilisteID, #Immatriculation, DateDebut, DateFin)

  AutomobilisteID, Immatriculation, DateDebut → DateFin

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 par 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 complétement 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 définie sur un schéma et est donc vérifiée pour toutes les instances associées à un schéma.

Règles

Axiomes de Amstrong :

- **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**

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

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

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

- **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 $\mathbf{[X]^+ = X_i}$

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 $\mathbf{F^+ = G^+}$
- 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 $F'_{i+1} = F'_i$ (*<u>refaire éventuellement l'étape 2</u>)*

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, …, A_n) et F une famille de DF associée à R

- Un sous-ensemble K de {A_1, A_2, …, A_n} est une **clé minimale** de R si et seulement si :

  ① La dépendance fonctionnelle $K \rightarrow A_1, A_2, \ldots, A_n \in F^+$ ou $[K]^+ = \{A_1, A_2, \ldots, A_n\}$

  ② $\forall\ X \subset K$, on a pas $X \rightarrow A_1, A_2, \ldots, A_n$

- Si K n'est pas un ensemble minimal alors K est une **surclé**

- Les dépendances fonctionnelles permettent de déduire les clés minimales des relations

Pour déterminer les clés minimales

Soit F une famille **minimale** 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 connait 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, .., R_n est sans perte d'information si et seulement si **∀r**, instance de R :

  $$r = \Pi_{R_1}(r) \infty \Pi_{R_2}(r) \infty ... \infty \Pi_{R_n}(r)$$

- **Décomposition sans perte de dépendances**

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

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

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 ¬ SPI

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

Une instance *r* de *R*

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

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

A	B
a_1	b_1
a_2	b_1

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

B	C
b_1	c_1
b_1	c_2

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

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 ¬ SPI car :

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

Décomposition SPI

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

$\}$ $\forall$ **r**

Soit $t\,(a_1, a_2, ..., a_{n'})$, un nuplet générique issu de $\infty_{i=1}^{n} \prod_{R_i} (r)$

t est construit à partir de $t_1 \infty\ t_2 \infty ... \infty t_n$

avec $t_1 \in \prod_{R_1} (r)$

$t_2 \in \prod_{R_2} (r)$

...

$t_n \in \prod_{R_n} (r)$

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

Exemples de décomposition SPI (1/2)

Exemple :

R(Fournisseur, Adresse, Produit, Prix)

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

R est décomposée en :

R_1*(Fournisseur, Adresse)* et R_2*(Fournisseur, Produit, Prix)*

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

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

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 :

Fournisseur	*Adresse*	*Produit*	*Prix*
f	a	x_{13}	x_{14}
f	x_{22}	po	pi

Exemples de décomposition SPI (2/2)

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

Fournisseur	*Adresse*	*Produit*	*Prix*
f	*a*	x_{13}	x_{14}
f	~~x_{22}~~ a	*po*	*pi*

Donc t $\in$ *r*, donc $\forall$**r,** $\mathbf{\Pi_{R_1}(r) \infty \Pi_{R_2}(r) \subseteq r}$
Et comme on a toujours $\forall$**r,** $r \subseteq \Pi_{R_1}(r) \infty \Pi_{R_2}(r)$
Par conséquent : $\forall r,\ r = \Pi_{R_1}(r) \infty \Pi_{R_2}(r)$
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→ 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 $\Pi_{R_1}(r) \infty \Pi_{R_2}(r)$:

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

A→ B ne permet pas de déduire les inconnues

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

$r \neq \Pi_{R_1}(r) \infty \Pi_{R_2}(r) \infty ... \infty \Pi_{R_n}(r)$

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

$r_1 \infty r_2$:

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 ¬ SPI car :

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

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 $\Rightarrow$

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

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

Exemples de décompositions SPD

Exemple 1 : $R(Rue, CP, Ville)$ 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 $F_2 = \varnothing$

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

La décomposition est donc ¬SPD.

Exemple 2 : $R(Nom, Bureau, Telephone)$

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(Cru, Pays, Région, Qualité)
avec F = { Cru, Pays → Région ; Cru, Pays → Qualité ;
Région → Pays}

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

Exemple 2 : *R(A, B, C) avec F = {A→ B ; B→ A ; B→ 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 :** $R_1, R_2, \ldots, R_n$ 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 $F_1 = \{X \rightarrow A, \ldots\}$ **qui doit être 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 BNCF (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* $F_{22} = \varnothing$ $K_{22} = BD$, 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 BNCF (2/2)

Exemple 2 : *R(Cru, Pays, Région, Qualité)*

avec F = {Cru, Pays → Région ; Cru, Pays → Qualité ; Région → Pays}

K=(Cru, Pays) ou *(Cru, Région)* ¬ BCNF

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

$R_2(C, P, Re)$ *avec* $F_2 = \{C, P \rightarrow Re ; Re \rightarrow P\}$

$K_2=(C,P)$ ou *(C, Re)* , ¬ BCNF

$R_{21}(Re, P)$ *avec* $F_{21} = \{Re \rightarrow P\}$ $K_{21}=Re$, BCNF

$R_{22}(C, Re)$ *avec* $F_{22} = \varnothing$ $K_{22}= (C, Re)$, BCNF

Donc *R* est décomposée en 3 relations BCNF :

$R_1(C, P, Q)$, $R_{21}(Re, P)$ et $R_{22}(C, Re)$

La décomposition est SPI mais ¬ 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) 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) ou (BCD)
  $A \rightarrow B$ est 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) 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 ¬3NF et F une famille **minimale** de DF associée à R
- **Sortie :** $R_1, R_2, \ldots, R_n$ 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 $F_i = \{X \rightarrow A\ ;\ \ldots\}$ *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* $F_4 = \varnothing$, 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

Objectifs principaux de la décomposition de schéma

- La 3NF n'élimine pas la redondance d'information
- Objectifs 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

$\Rightarrow$ 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 :

  ∀ 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 ¬ (∃ une dépendance fonctionnelle partielle)**
- Un schéma de relation R est **3NF** si :
  - le schéma est 2NF
  - **et ¬ (∃ une dépendance fonctionnelle transitive)**

Les 4 formes normales vues en cours