Dauphine | PSL
UNIVERSITÉ PARIS

MIDO
MATHEMATIQUES ET INFORMATIQUE DE LA DÉCISION ET DES ORGANISATIONS

M2 Informatique des Organisations

2022-2023

Bases de données relationnelles

Mise à Niveau

Maude Manouvrier

- Modèle relationnel
- SQL
- Passage d'un modèle Entité/Association ou UML en relationnel

http://www.lamsade.dauphine.fr/~manouvri/MAN_BD/

BIBLIOGRAPHIE

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

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

T. 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, disponible à la BU 005.72 SQL

R. Ramakrishnan et J. Gehrke, *Database Management Systems*, Second Edition; McGraw-Hill, 2002, ISBN: 0-07-232206-3, disponible à la BU 055.7 RAM

A. Silberschatz, H.F. Korth et S. Sudarshan, *Database System Concepts*, McGraw-Hill, 6ème édition, 2010, ISBN: 978-0073523323,

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

BIBLIOGRAPHIE

Autres ouvrages de référence, disponibles à la BU :

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

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

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

Ouvrages pédagogiques contenant des exercices corrigés :

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

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

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

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

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

Chap. I - Introduction

Bases de données :

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

Exemple 1 :

Organisation : une bibliothèque

Données : les livres, les emprunts, les emprunteurs

Exemple 2 :

Organisation : une Université

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

SGBD (1/3)

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

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

Classement des moteurs de bases de données

383 systems in ranking, February 2022

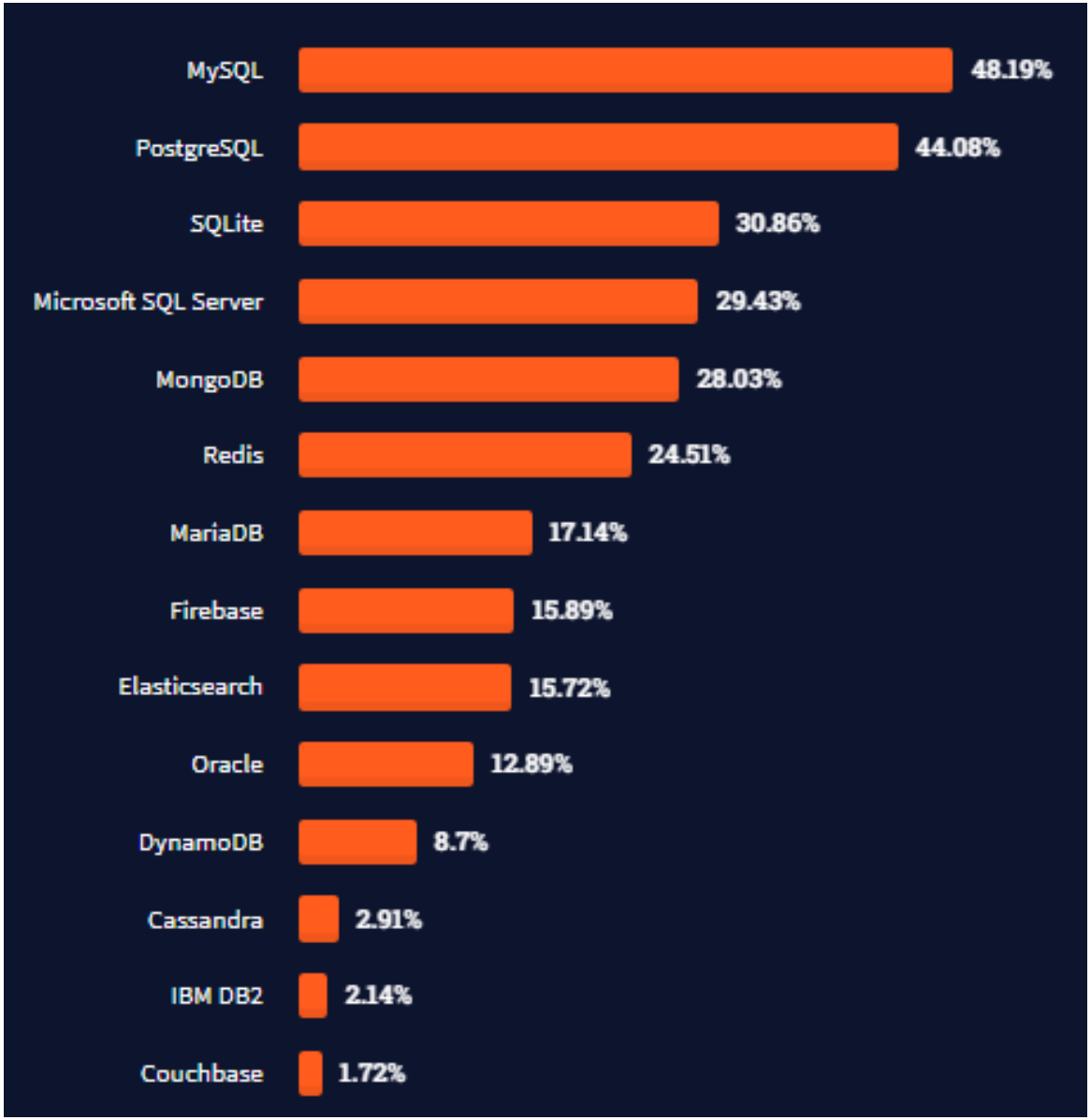
Rank			DBMS	Database Model	Score		
Feb 2022	Jan 2022	Feb 2021			Feb 2022	Jan 2022	Feb 2021
1.	1.	1.	Oracle	Relational, Multi-model	1256.83	-10.05	-59.84
2.	2.	2.	MySQL	Relational, Multi-model	1214.68	+8.63	-28.69
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	949.05	+4.24	-73.88
4.	4.	4.	PostgreSQL	Relational, Multi-model	609.38	+2.83	+58.42
5.	5.	5.	MongoDB	Document, Multi-model	488.64	+0.07	+29.69
6.	6.	↑ 7.	Redis	Key-value, Multi-model	175.80	-2.18	+23.23
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	162.88	-1.32	+5.26
8.	8.	8.	Elasticsearch	Search engine, Multi-model	162.29	+1.54	+11.29
9.	9.	↑ 11.	Microsoft Access	Relational	131.26	+2.31	+17.09
10.	10.	↓ 9.	SQLite	Relational	128.37	+0.94	+5.20
11.	11.	↓ 10.	Cassandra	Wide column	123.98	+0.43	+9.36
12.	12.	12.	MariaDB	Relational, Multi-model	107.11	+0.69	+13.22

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

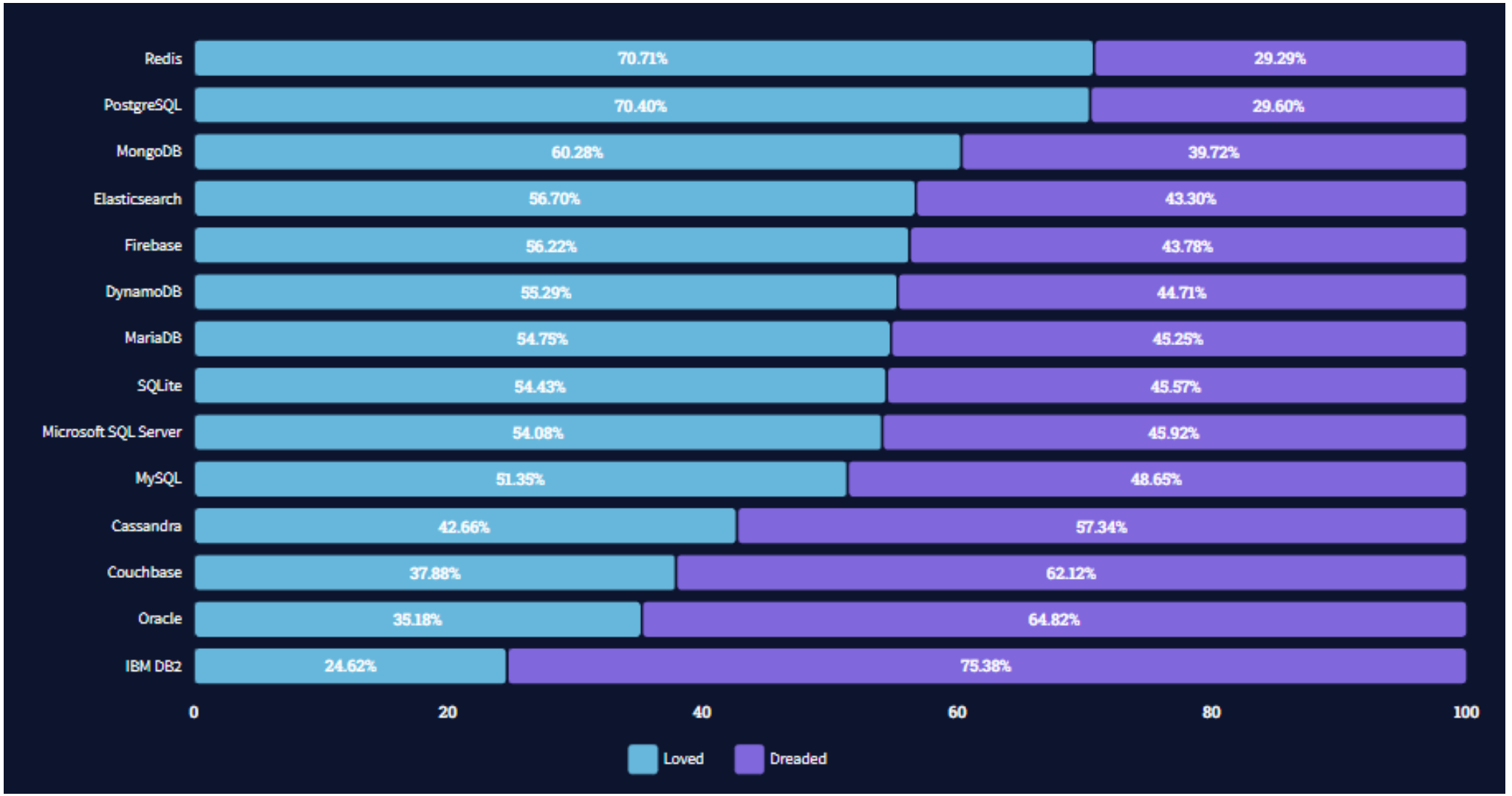
Bases de données les plus populaires en 2021

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

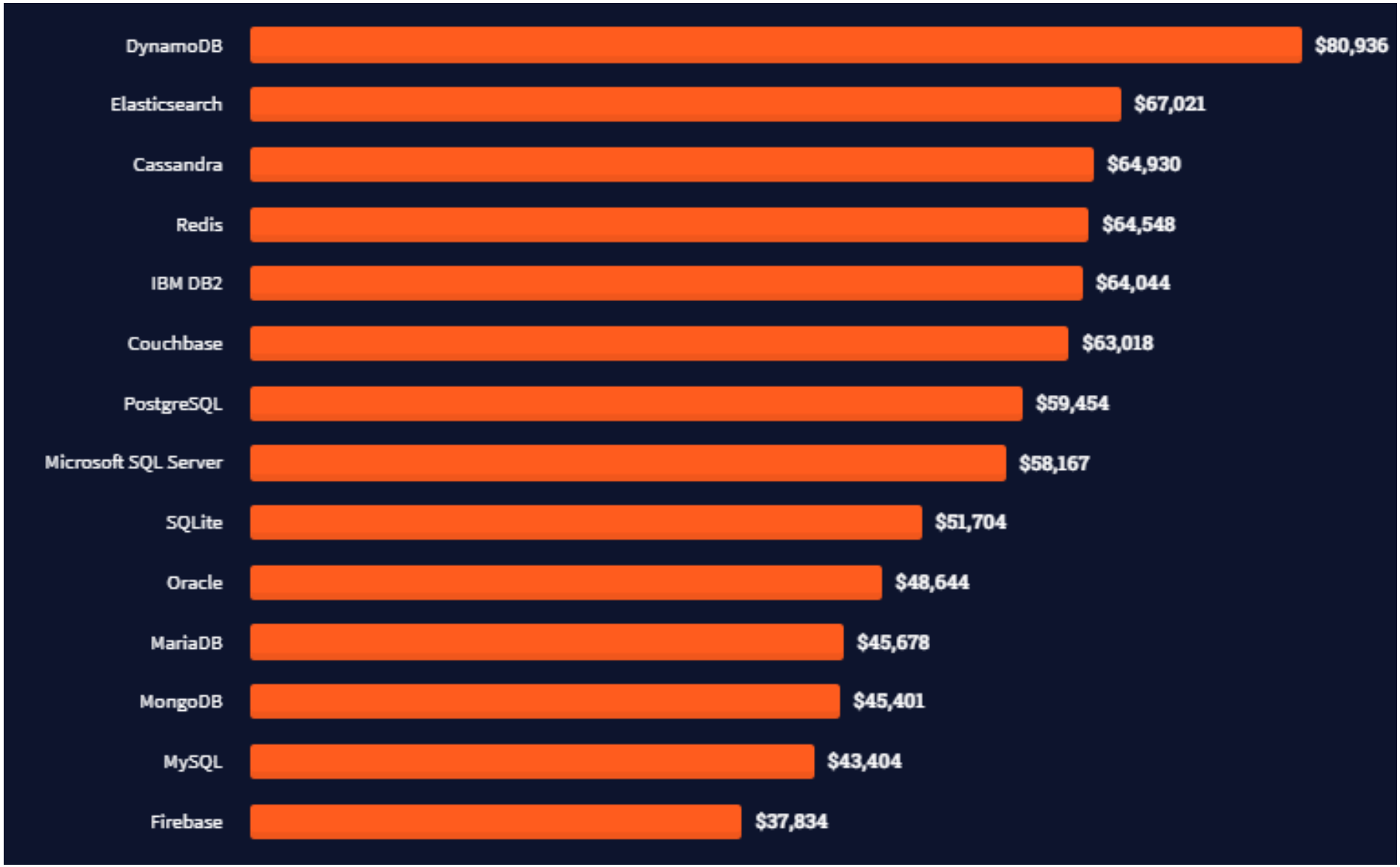
https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies



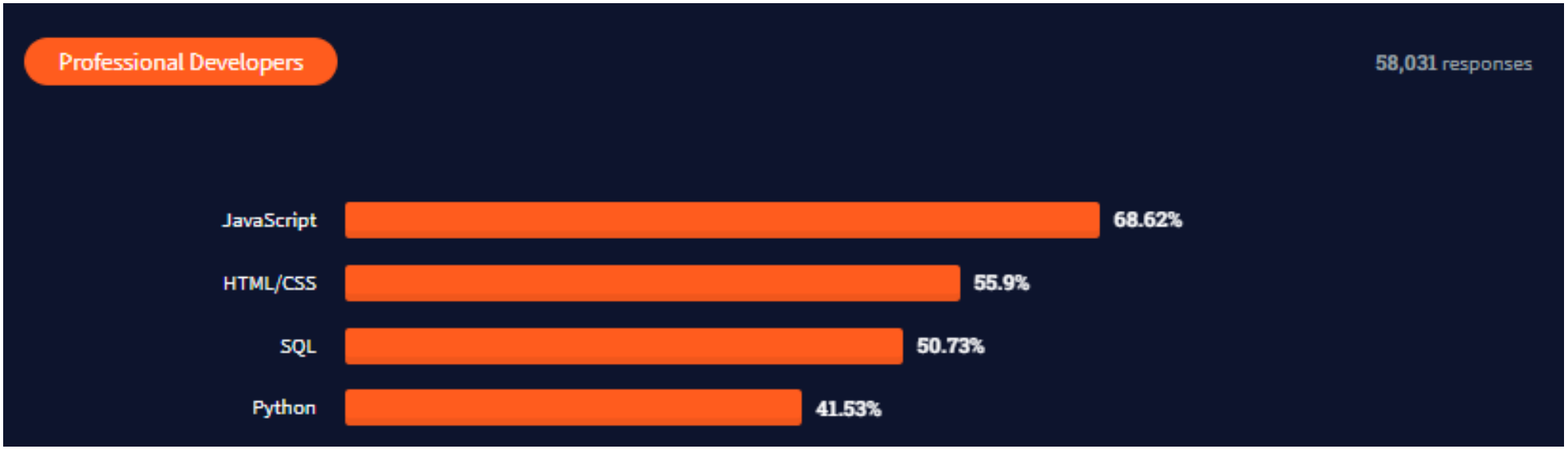
Appréciation/Détestation des bases de données en 2021



Profils BD les mieux payés en 2021



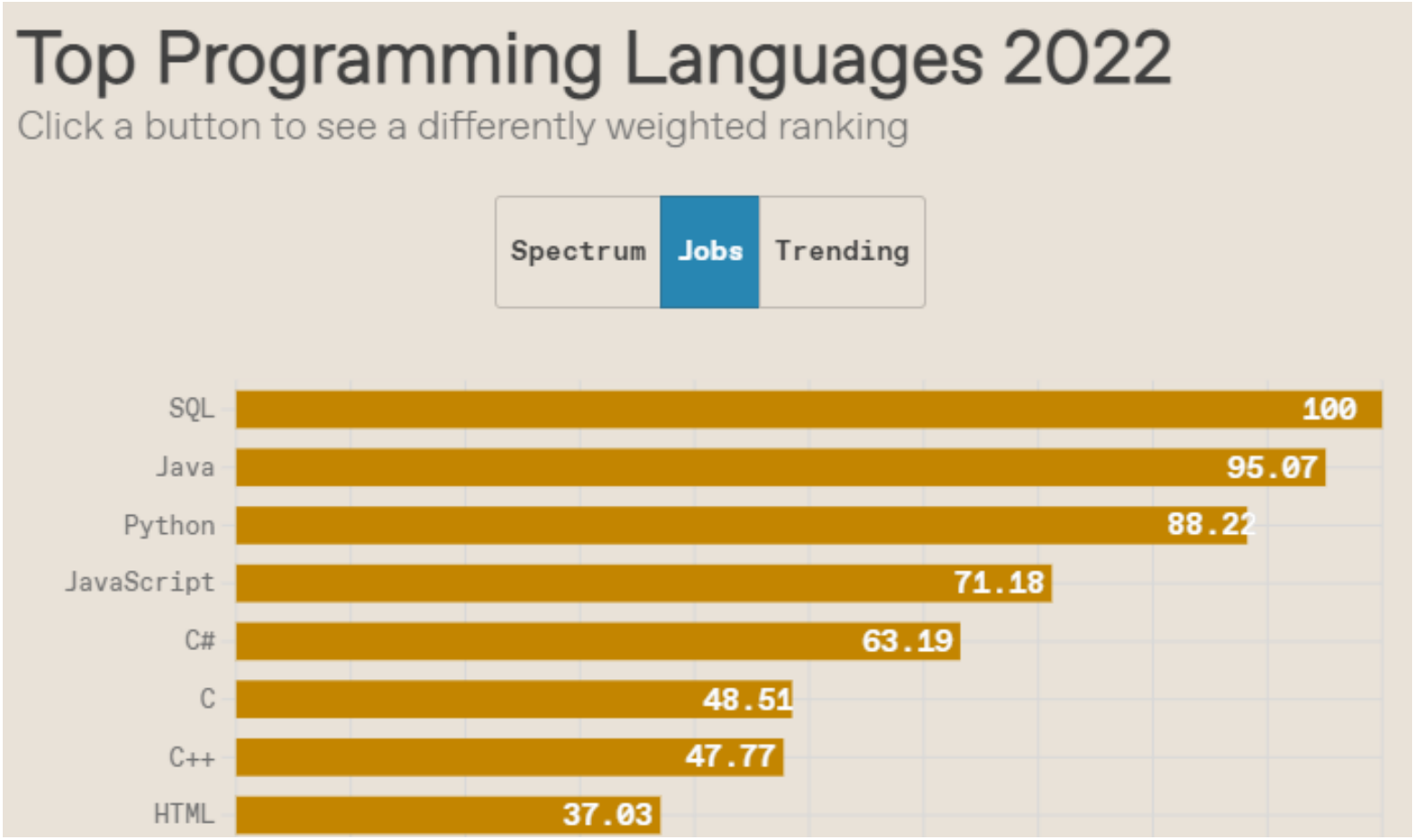
Position du SQL parmi les technologies les plus populaires en 2021



Position du SQL parmi les tendances des offres d'emploi en 2022

"The Rise of SQL It's become the second programming language everyone needs to know"

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 jours doivent être pérennes même en cas de panne.

Abstraction des données

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

Instances et schéma

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

Petit historique

- **1960** : systèmes de gestion de fichiers
- **1970** : début des SGBD réseaux et hiérarchiques proches des systèmes de gestion de fichiers ⇒ pas d'interrogation sans savoir où est l'information recherchée ("navigation") et sans écrire de programmes
- **1970** : papier fondateur de CODD sur la théorie des relations

  fondement de la théorie des bases de données relationnelles

  INGRES à Berkeley - langage QUEL

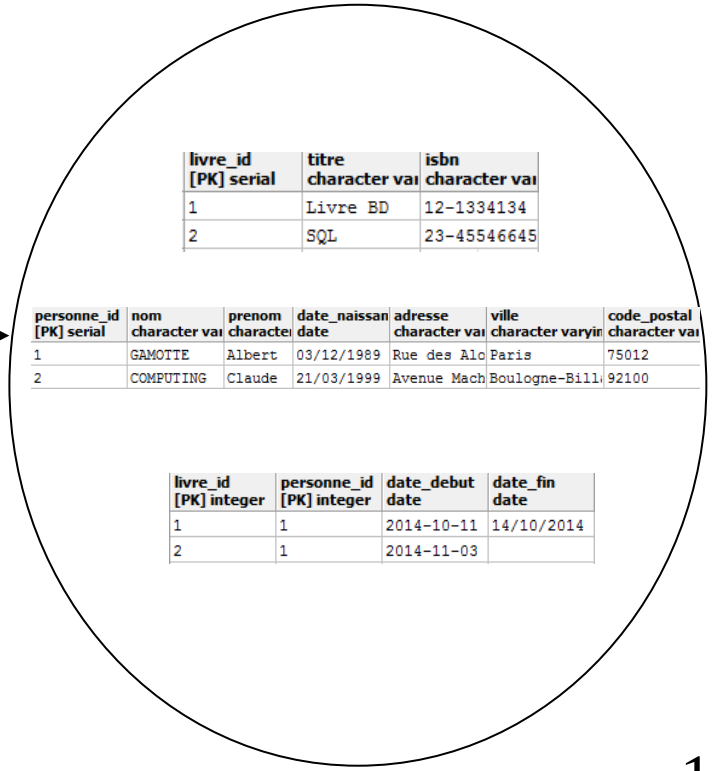
  System R IBM à San Jose, Ca. - langages SEQUEL et QBE
- **1980** : **Apparition des SGBD relationnels sur le marché** (Oracle, Ingres, Informix, Sybase, DB2 …)
- **1990** : **début des SBGD orientés objet** (Gemstone, O_2, Orion, Objectstore, Versant, Matisse...).
- **Aujourd'hui** : **relationnel-objet, NoSQL et NewSQL**
- **cf . L'histoire des bases de données** https://www.youtube.com/watch?v=iu8z5QtDQhY

Chap II - Modèle relationnel

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

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

Modèle relationnel



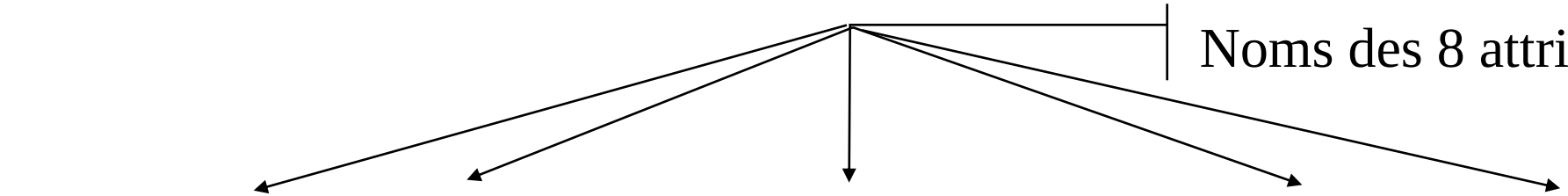
livre_id [PK] serial	titre character var	isbn character var
1	Livre BD	12-1334134
2	SQL	23-45546645

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	

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

Intégrité structurelle

- **Unicité des clés**
  - **Ensemble minimal d'attributs** dont la connaissance des valeurs permet d'identifier un nuplet unique de la relation considérée
  - ***R* a pour clé *K* si : $\forall t_1, t_2$ nuplets d'une instance de *R***

    $$t_1.K \neq t_2.K$$

  - **Clé minimale :** si K est une clé minimale alors $\nexists$ K' $\subset$ K tel que K' est une clé
  - **Clé primaire :** une clé parmi toutes les clés minimales

Clé / Clé minimale

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

Clés primaires possibles : PersonneID ou NSS

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

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

Clé primaire : Immatriculation

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

Location (PersonneID, Immatriculation, Date)

Clé primaire : (Personne_ID, Immatriculation, Date)

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

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

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

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

Problèmes :

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

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

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

La solution ? Diviser les données en plusieurs relations

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

⇒ Stockage unique des informations de chaque livre

livre_id [PK] serial	titre character 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 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

- 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	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*
  - Ou autorisée mais avec suppression/mise à jour en cascade des nuplets correspondant dans *Emprunt*

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

Relation *Emprunt*

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

Relation *Personne*

personne_id [PK] serial	nom character 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 Emrpunt

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 var	prenom characte	date_naissan date	adresse character var	ville character varyin	code_postal character var
~~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 structurelle

- **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

<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)

Démarche :

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

Étape N° 2 : Définir le modèle relationnel

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

Démarche :

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

Quelques règles

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

Exemple 1:

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*

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*

Exemple 2:

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)

Exemple 3:

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

Requête et langage d'interrogation

- **Différents types de requête :**
  - Requêtes d'interrogation
  - Requêtes d'insertion, de mise à jour et de suppression des données
  - Requêtes de définition de schéma
- **Plusieurs langages d'interrogation** :
  - **Algèbre relationnelle** : exprime comment le résultat des requêtes est calculé par le SGBD
  - **SQL** (*Structured Query Language* – Langage de bases de données standard) : décrit sous forme logique le résultat de la requête calculée par le SGBD

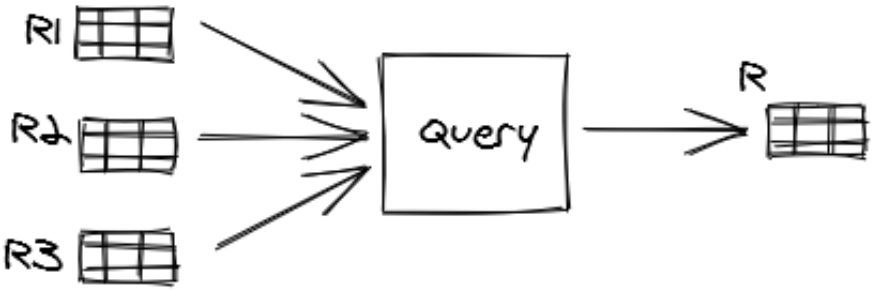
Requête d'interrogation

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)



Chap III - SQL

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

SQL

- **Basé sur la logique du 1er ordre**
- Requête d'interrogation : décrit sous forme logique le résultât 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/**
- Rappels des commandes SQL : **http://sql.sh/**

Forme générale d'une requête d'interrogation

SELECT [DISTINCT] *
FROM table_1 [variable_1], table_2 [variable _1], …
[WHERE prédicat_1
AND [ou OR] prédicat_2 …]

SELECT [DISTINCT] exp_1 [AS nom_1], exp_2 ...
FROM table_1 [variable _1], table_2 [variable _1], …
[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 ' _AM% ';

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
```

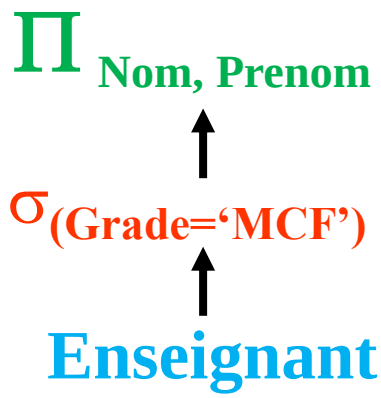
L..	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	NAIJA	Yosr
3	BAHRI	Afef
4	LIMAM	Medhi
5	MyTaylor	IsRich
6	RIGAUX	Philippe
7	CHAKHAR	Salem
8	MURAT	Cécile

```
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);
```

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

Clause EXISTS

Clause EXISTS :

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

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

Exemples de requêtes d'agrégation

Fonctions statistique et de groupage :

COUNT, MIN, MAX, AVG, SUM, EVERY, …

ORDER BY, GROUP BY

```
SELECT COUNT(*)
  FROM Etudiant ;

SELECT AVG(Capacite), SUM(Capacite)
  FROM Salle ;

SELECT Departement_ID, Nom, Prenom
  FROM Enseignant
    ORDER BY Departement_ID DESC, Nom, Prenom ;

SELECT Departement_Name, COUNT(*)
  FROM Reservation_Salle
    GROUP BY Departement_Name HAVING COUNT(*)  >= 2 ;
```

Department_Name	count
Math	2

Exemple d'ORDER BY

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

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 CASE … WHEN

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	nbticketsach 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 NbTicketsAchetes 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

Jointure

Jointure : *forme générale*

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

Jointure interne :

```
SELECT Nom, Prénom, Nom_Département
  FROM Enseignant INNER JOIN Département
   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

OUTER JOIN – NATURAL JOIN et CROSS JOIN

Jointure externe :

SELECT Nom, Prénom, Nom_Département
FROM Enseignant LEFT OUTER JOIN Departement ON
Enseignant.Departement_ID = Derpartement.Departement_ID ;

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

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

Jointure naturelle :

SELECT Nom, Prénom, Nom_Département
FROM Enseignant NATURAL JOIN Département

Produit cartésien:

SELECT Nom, Prénom, Nom_Département
FROM Enseignant CROSS JOIN Département

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

Différentes manières d'écrire une jointure

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

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

```
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 #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

Union, intersection et différence

```
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!

Exemples 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

Exemples 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

Voir le plan d'exécution d'une requête

Exemple sous PostgreSQL en utilisant `NATURAL JOIN`

postgres on postgres@PostgreSQL 9.6

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

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Hash Join (cost=2.14..3.25 rows=1 width=334) (actual time=0.077..0.07...
2	Hash Cond: (((enfant.nom)::text = (atelier.nom)::text) AND (inscription.a...
3	-> Hash Join (cost=1.07..2.14 rows=3 width=108) (actual time=0.030..0....
4	Hash Cond: (enfant.enfantid = inscription.enfantid)
5	-> Seq Scan on enfant (cost=0.00..1.03 rows=3 width=104) (actual time...
6	-> Hash (cost=1.03..1.03 rows=3 width=8) (actual time=0.011..0.011 ro...
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on inscription (cost=0.00..1.03 rows=3 width=8) (actual tim...
9	-> Hash (cost=1.03..1.03 rows=3 width=288) (actual time=0.029..0.029 ...
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on atelier (cost=0.00..1.03 rows=3 width=288) (actual time...
12	Planning time: 0.424 ms
13	Execution time: 0.154 ms

Plan d'exécution d'une requête

Exemple de jointure écrite avec des virgules et un `WHERE`

postgres on postgres@PostgreSQL 9.6

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

Data Output | Explain | Messages | History

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

Plan d'exécution d'une requête

Exemple sous PostgreSQL en utilisant `INNER JOIN`

postgres on postgres@PostgreSQL 9.6

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

Data Output | Explain | Messages | History

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

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)
```

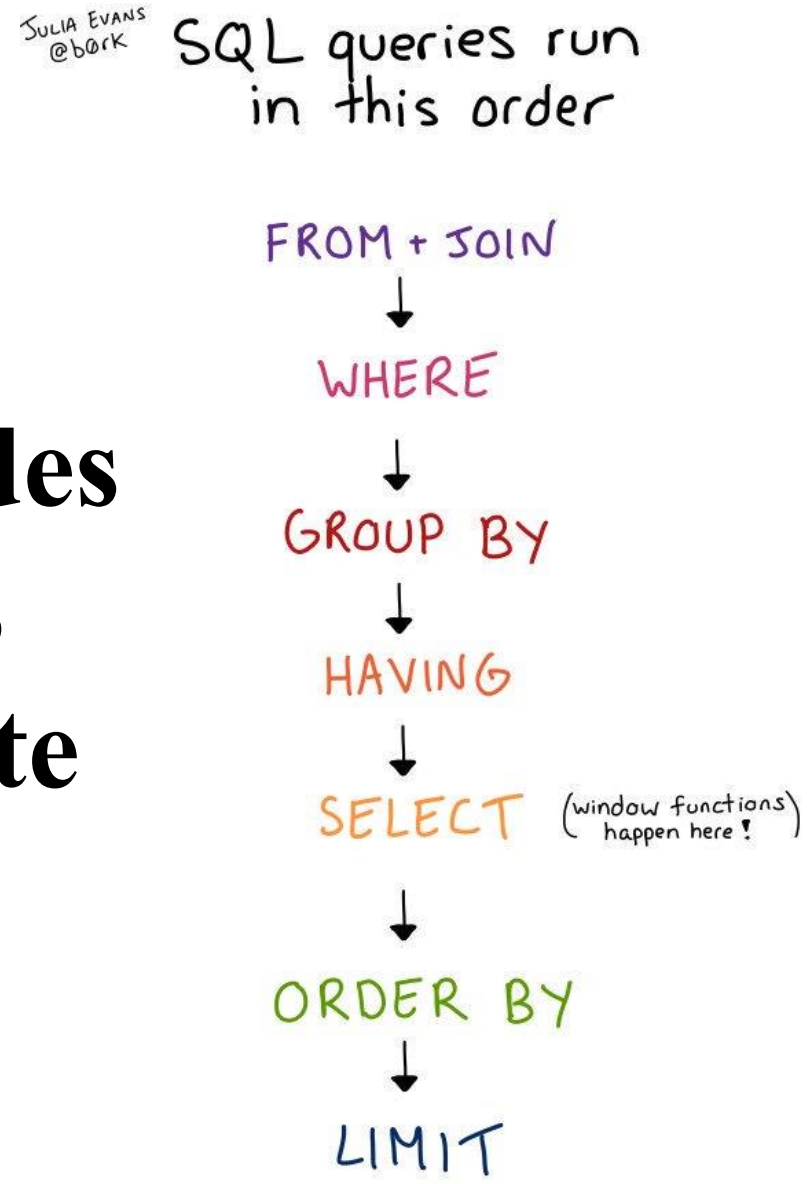
```
select titre
from   Film f, Rôle r
where  f.id_film = r.id_film
and    f.annee = 1958
and    r.id_acteur in (select id_acteur
                       from Artiste
                       where nom='Stewart'
                       and prénom='James')
```

```
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.

Ordre d'exécution des opérateurs d'une requête



Commandes sous PostgreSQL

- **VACUUM :** pour supprimer l'espace non utilisé (suppression lignes supprimées ou rendues obsolètes par une mise à jour) – cf. https://docs.postgresql.fr/13/sql-vacuum.html
- **ANALYZE :** pour mettre à jour les statistiques - se combine avec `VACUUM` – cf. https://docs.postgresql.fr/13/sql-analyze.html
- **EXPLAIN :** pour afficher le plan d'exécution d'une requête – cf. https://docs.postgresql.fr/13/sql-explain.html
- **VERBOSE :** pour afficher un rapport détaillé - se combine avec `VACUUM`, `ANALYZE` et `EXPLAIN`
- **SET ENABLE_SEQSCAN TO OFF :** pour interdire l'utilisation du parcours séquentiel (pour forcer l'utilisation des index) – cf. https://docs.postgresql.fr/13/runtime-config-query.html#RUNTIME-CONFIG-QUERY-ENABLE
- **CREATE INDEX "Nom_Index" ON Relation USING btree (nom) :** pour créer un index en précisant son type

Analyse du plan d'exécution

postgres on postgres@PostgreSQL 9.6

```
1  EXPLAIN ANALYSE
2  SELECT * FROM services_big;
```

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Seq Scan on services_big (cost=10000000000.00..10000000686.04 rows=40004 width=29) (actual time=0.022..4.597 rows=40004 loops...
2	Planning time: 0.088 ms
3	Execution time: 6.434 ms

De gauche à droite :

- Algorithme utilisé (ici lecture séquentielle de la table)
- Coût estimé du lancement, i.e. temps passé avant que l'affichage de la sortie ne commence (ex. le temps de faire un tri …) ~ coût de l'affichage de la 1ere ligne
- Coût total estimé
- Nombre de lignes estimé en sortie
- Largeur moyenne estimée (en octets) des lignes en sortie

Cf. doc https://docs.postgresql.fr/13/using-explain.html#USING-EXPLAIN-ANALYZE

Choix de l'algorithme de jointure

Le SGBD choisit en fonction de statistiques le « meilleur » algorithme de jointure et d'utiliser ou non un index

postgres on postgres@PostgreSQL 9.6

```
1  EXPLAIN ANALYSE SELECT matricule, nom, prenom, nom_service, fonction, localisation
2  FROM   employes emp
3  JOIN   services ser ON (emp.num_service = ser.num_service)
4  WHERE  ser.localisation = 'Nantes';
```

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Hash Join (cost=1.06..2.27 rows=4 width=175) (actual time=0.068..0.075 rows=5 loops=1)
2	Hash Cond: (emp.num_service = ser.num_service)
3	-> Seq Scan on employes emp (cost=0.00..1.14 rows=14 width=162) (actual time=0.026..0.028 rows=14 loops=1)
4	-> Hash (cost=1.05..1.05 rows=1 width=21) (actual time=0.032..0.032 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on services ser (cost=0.00..1.05 rows=1 width=21) (actual time=0.025..0.026 rows=1 loops=1)
7	Filter: ((localisation)::text = 'Nantes'::text)
8	Rows Removed by Filter: 3
9	Planning time: 0.284 ms
10	Execution time: 0.126 ms

Forcer l'utilisation d'un index

postgres on postgres@PostgreSQL 9.6

```
1  SET ENABLE_SEQSCAN TO OFF;
2  EXPLAIN ANALYSE SELECT matricule, nom, prenom, nom_service, fonction, localisation
3  FROM   employes emp
4  JOIN   services ser ON (emp.num_service = ser.num_service)
5  WHERE  ser.localisation = 'Nantes';
```

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Merge Join (cost=10000000001.54..10000000013.74 rows=4 width=48) (actual time=0.067..0.071 rows=5 loops=1)
2	Merge Cond: (emp.num_service = ser.num_service)
3	-> Sort (cost=10000000001.41..10000000001.44 rows=14 width=35) (actual time=0.051..0.053 rows=14 loops=1)
4	Sort Key: emp.num_service
5	Sort Method: quicksort Memory: 26kB
6	-> Seq Scan on employes emp (cost=10000000000.00..10000000001.14 rows=14 width=35) (actual time=0.018..0.022 rows=14 loops=1)
7	-> Materialize (cost=0.13..12.20 rows=1 width=21) (actual time=0.011..0.011 rows=1 loops=1)
8	-> Index Scan using services_pkey on services ser (cost=0.13..12.20 rows=1 width=21) (actual time=0.007..0.007 rows=1 loops=1)
9	Filter: ((localisation)::text = 'Nantes'::text)
10	Rows Removed by Filter: 3
11	Planning time: 0.315 ms
12	Execution time: 0.109 ms

Forcer l'utilisation d'un algorithme

postgres on postgres@PostgreSQL 9.6

```
1  SET ENABLE_HASHJOIN TO OFF;
2  EXPLAIN ANALYSE SELECT matricule, nom, prenom, nom_service, fonction, localisation
3  FROM   employes emp
4  JOIN   services ser ON (emp.num_service = ser.num_service)
5  WHERE  ser.localisation = 'Nantes';
```

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Nested Loop (cost=10000000000.13..10000000013.55 rows=4 width=175) (actual time=0.027..0.033 rows...
2	Join Filter: (emp.num_service = ser.num_service)
3	Rows Removed by Join Filter: 9
4	-> Seq Scan on employes emp (cost=10000000000.00..10000000001.14 rows=14 width=162) (actual time=...
5	-> Materialize (cost=0.13..12.21 rows=1 width=21) (actual time=0.001..0.001 rows=1 loops=14)
6	-> Index Scan using services_pkey on services ser (cost=0.13..12.20 rows=1 width=21) (actual time=0.008....
7	Filter: ((localisation)::text = 'Nantes'::text)
8	Rows Removed by Filter: 3
9	Planning time: 0.145 ms
10	Execution time: 0.055 ms

Forcer l'utilisation d'un index

postgres on postgres@PostgreSQL 9.6

```
1  SET enable_nestloop TO OFF;
2  EXPLAIN ANALYSE SELECT matricule, nom, prenom, nom_service, fonction, localisation
3  FROM   employes emp
4  JOIN   services ser ON (emp.num_service = ser.num_service)
5  WHERE  ser.localisation = 'Nantes';
```

Data Output | Explain | Messages | History

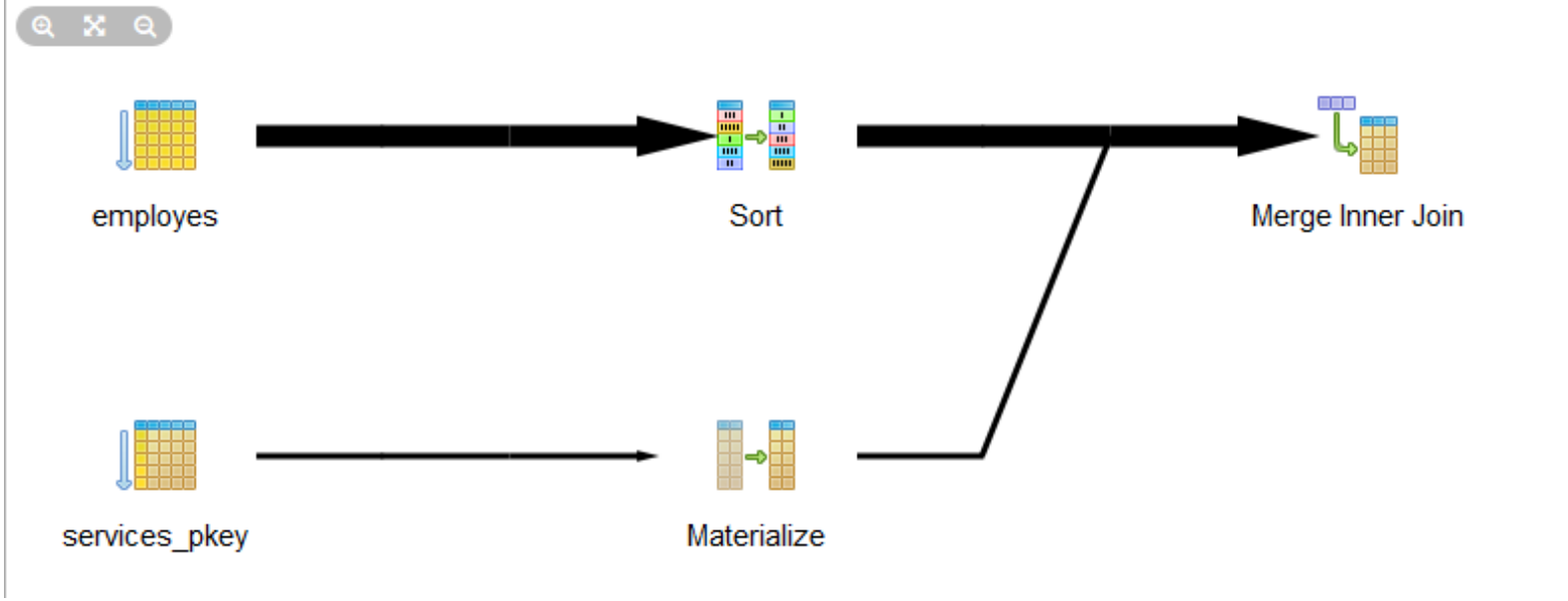
	QUERY PLAN text
1	Merge Join (cost=10000000001.54..10000000013.69 rows=4 width=175) (actual time=0.043..0.046 rows=5 l...
2	Merge Cond: (emp.num_service = ser.num_service)
3	-> Sort (cost=10000000001.41..10000000001.44 rows=14 width=162) (actual time=0.032..0.033 rows=14 lo...
4	Sort Key: emp.num_service
5	Sort Method: quicksort Memory: 26kB
6	-> Seq Scan on employes emp (cost=10000000000.00..10000000001.14 rows=14 width=162) (actual time=0...
7	-> Index Scan using services_pkey on services ser (cost=0.13..12.20 rows=1 width=21) (actual time=0.008.....
8	Filter: ((localisation)::text = 'Nantes'::text)
9	Rows Removed by Filter: 3
10	Planning time: 0.167 ms
11	Execution time: 0.079 ms

Visualisation du plan d'exécution

postgres on postgres@PostgreSQL 9.6

```
1  EXPLAIN (FORMAT JSON) SELECT matricule, nom, prenom, nom_service, fonction, localisation
2  FROM   employes emp
3  JOIN   services ser ON (emp.num_service = ser.num_service)
4  WHERE  ser.localisation = 'Nantes';
```

Data Output | Explain | Messages | History



Visualisation du plan d'exécution en JSON

postgres on postgres@PostgreSQL 9.6

```
1  EXPLAIN (FORMAT JSON) SELECT matricule, nom, prenom, nom_service, fonction, localisation
2  FROM   employes emp
3  JOIN   services ser ON (emp.num_service = ser.num_service)
4  WHERE  ser.localisation = 'Nantes';
```

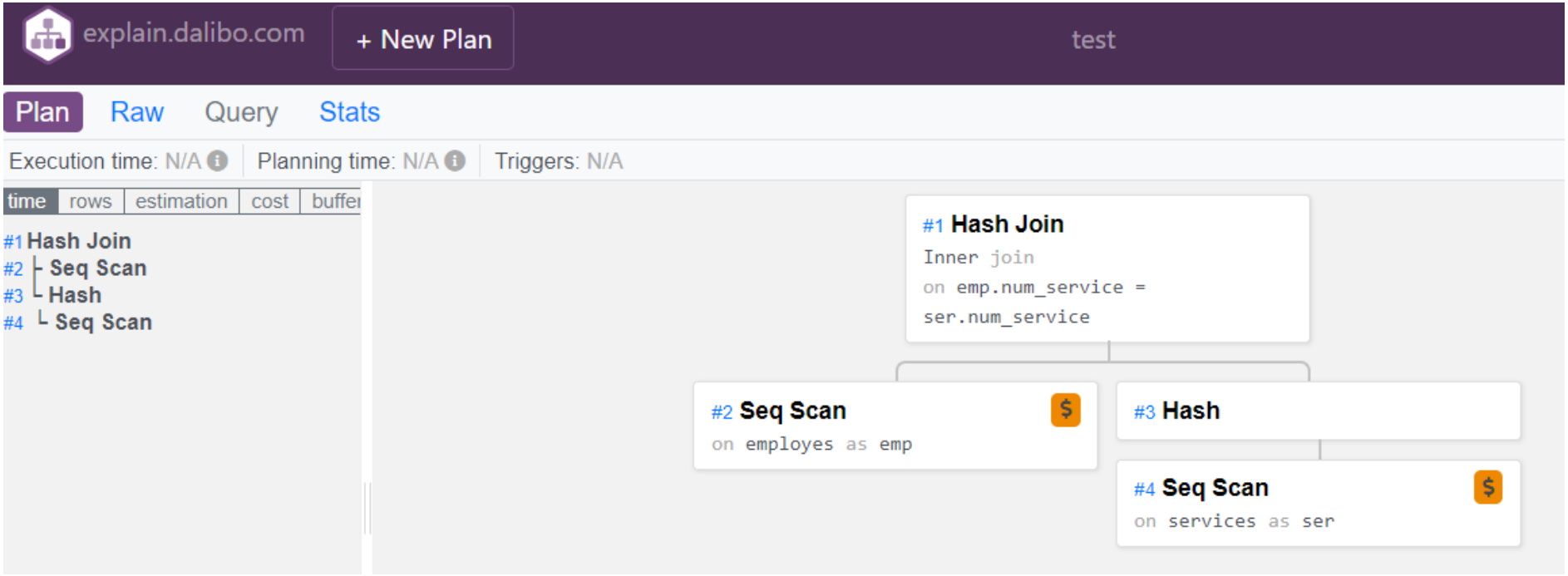
Data Output | Explain | Messages | History

	QUERY PLAN json
1	[{ "Plan": { "Startup Cost": 10000000001.54, "Plans": [{ "Startup Cost": 10000000001.41, "Plans": [{ "Startup Cost": 100000000...

[{ "Plan": { "Startup Cost": 10000000001.54, "Plans": [{ "Startup Cost": 10000000001.41, "Plans": [{ "Startup Cost": 10000000000, "Node Type": "Seq Scan", "Plan Rows": 14, "Relation Name": "employes", "Alias": "emp", "Parallel Aware": false, "Parent Relationship": "Outer", "Plan Width": 35, "Total Cost": 10000000001.14 }], "Sort Key": ["emp.num_service"], "Node Type": "Sort", "Plan Rows": 14, "Parallel Aware": false, "Parent Relationship": "Outer", "Plan Width": 35, "Total Cost": 10000000001.44 }, { "Startup Cost": 0.13, "Plans": [{ "Filter": "((localisation)::text = 'Nantes'::text)", "Startup Cost": 0.13, "Scan Direction": "Forward", "Plan Width": 21, "Node Type": "Index Scan", "Plan Rows": 1, "Relation Name": "services", "Alias": "ser", "Parallel Aware": false, "Parent Relationship": "Outer", "Total Cost": 12.2, "Index Name": "services_pkey" }], "Node Type": "Materialize", "Plan Rows": 1, "Parallel Aware": false, "Parent Relationship": "Inner", "Plan Width": 21, "Total Cost": 12.2 }], "Node Type": "Merge Join", "Plan Rows": 4, "Join Type": "Inner", "Parallel Aware": false, "Plan Width": 48, "Merge Cond": "(emp.num_service = ser.num_service)", "Total Cost": 10000000013.74 } }]

Visualisation du plan d'exécution à partir d'un import JSON

https://explain.dalibo.com/ *: outils en ligne pour visualiser un plan d'exécution PostgreSQL à partir de son import JSON*



Division

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

`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
        )
    );
```

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

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 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

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

Ligne	etudiant_id (int4)
1	1

Insertion, suppression et mise à jour

- **Insertion**

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

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

- **Suppression**

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

- **Mise à jour**

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

- **Transactions : `COMMIT`, `ROLLBACK [TO]`, `SAVE POINT`**

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

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

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


DELETE FROM Produits ;

DELETE FROM Produits WHERE prix < 0.99;

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

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

Exemple :

```
-- 2 requêtes de mises à jour indépendantes
UPDATE compte SET solde = solde - 100 where compteid = x;
UPDATE compte SET solde = solde + 100 where compteid = y;

-- une transaction composée de 2 requêtes de mises à jour
BEGIN;
  UPDATE compte SET solde = solde - 100 where compteid = x;
  UPDATE compte SET solde = solde + 100 where compteid = y;
COMMIT;
```

Clause WITH

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

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

SELECT ...
```

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

Clause WITH + SELECT

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

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

Clause WITH + INSERT ou UPDATE

Exemples :

```
WITH lignes_deplacees AS (
    DELETE FROM produits  -- suppressions des nuplets
    WHERE
        "date" >= '2010-10-01' AND
        "date" < '2010-11-01'
    RETURNING * -- en retournant les nuplets supprimés
)
INSERT INTO log_produits
   SELECT * FROM lignes_deplacees;
-- clause WITH attachée à l'INSERT
-- la requête déplace les nuplets de produits vers log_produits

WITH t AS (
    UPDATE produits SET prix = prix * 1.05
    RETURNING *
)
SELECT * FROM t; -- affiche les données mises à jour
```

Clause `WITH` pour créer des variables

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

Listing 1: Creating and updating variables.

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

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

x
2

Clause `WITH` pour définir des fonctions

- Possibilité d'intégrer des fonctions locales

Listing 2: Calling functions.

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

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

temperature
80.32999999999998
-153.67000000000002

Clause `WITH` et structure conditionnelle

- Simulation d'une structure conditionnelle

```python
from random import random

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

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

temperatures = [300, 170]

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

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

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

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

SELECT * FROM condition
```

renvoie `true` ou `false`

Clause WITH RECURSIVE

- Pour faire des itérations

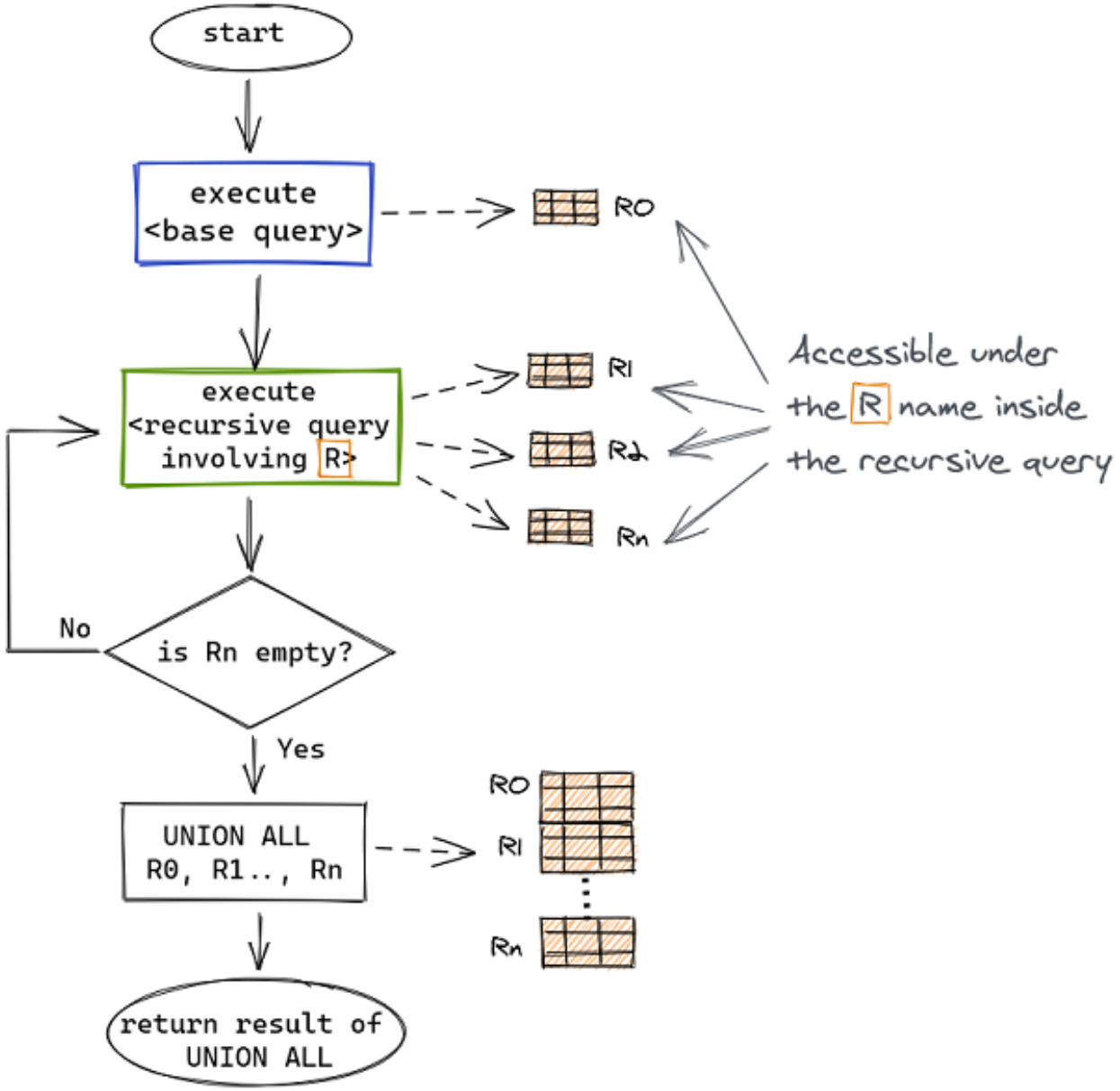
```
WITH RECURSIVE R AS (
      requeteDeBase
    UNION ALL
      requeteFaisantReferenceàR
    )
SELECT ... FROM R
```

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

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

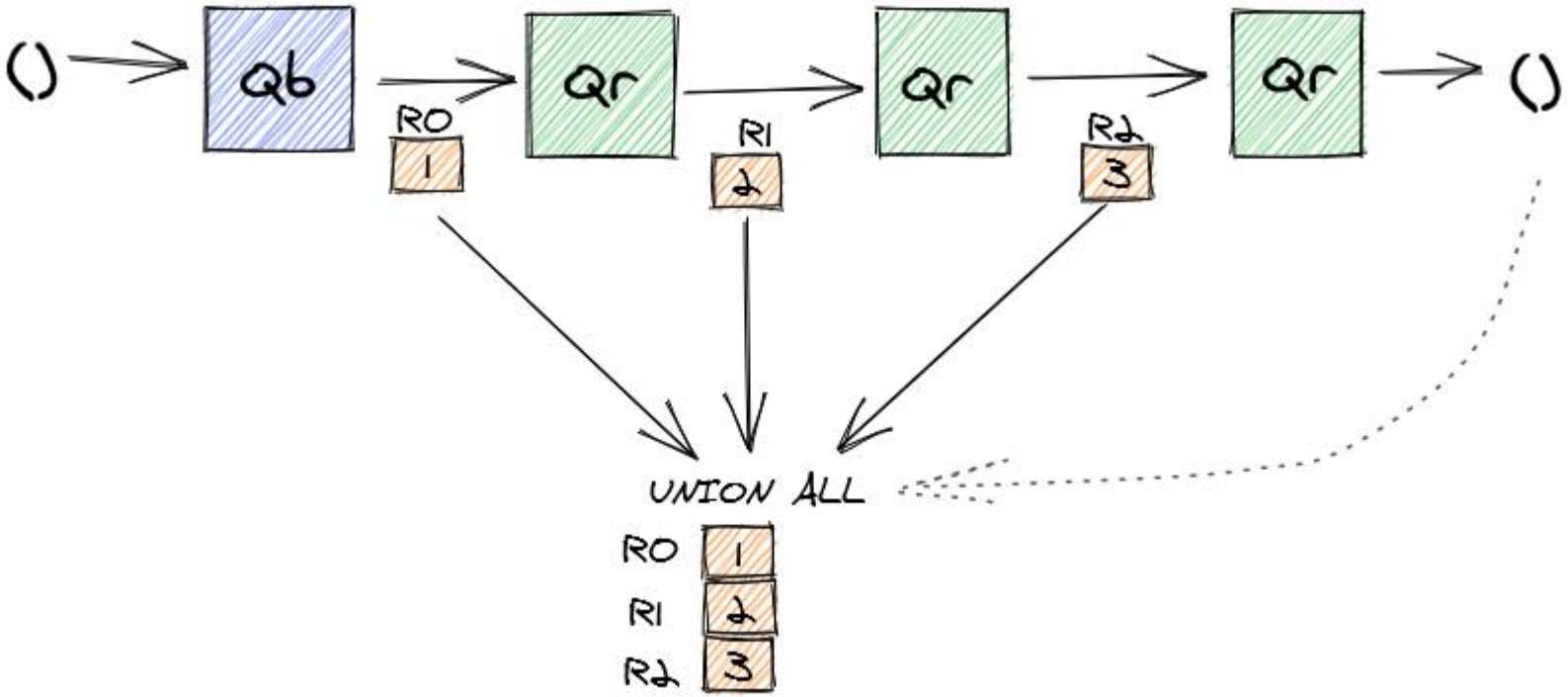
```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Fonctionnement de la clause `WITH RECURSIVE`



Exemple simple de clause `WITH RECURSIVE`

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



Exemple de clause `WITH RECURSIVE`

Relation `ParentEnfant` :

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

Requête qui affiche les ancêtres de *Jessy Kacontact* :

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

                UNION ALL

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

SELECT * FROM ancetre;
```

nomparent	prenomparent
Onette	Marion
Teste	Otto
Onette	Camille

Utilisation de `WITH RECURSIVE` pour itérer

Listing 4: Loop without recursive reference in a subquery.

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

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

e_pow_x
4.4816890703380645
54.59814947621461

x	i	e_pow_x
1.5	1	1
4	1	1
1.5	2	2.5
4	2	5
1.5	3	3.625
4	3	13
1.5	4	4.1875
4	4	23.666666666666664
1.5	5	4.3984375
4	5	34.33333333333333
1.5	6	4.46171875
4	6	42.86666666666666
1.5	7	4.4775390625
4	7	48.55555555555555
1.5	8	4.480929129464286
4	8	51.8063492063492
1.5	9	4.48156476702009
4	9	53.43174603174603
1.5	10	4.481670706612723
4	10	54.15414462081129

Boucle avec référence récursive

Listing 5: Loop with recursive references in subqueries.

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

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


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

Création de tables

CREATE TABLE table (col1 type 1 [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 ...

Exemple de CREATE TABLE

```
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

Exemples de CONSTRAINTS CHECK

```
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)
);
```

SERIAL, UNIQUE et MATCH SIMPLE/PARTIAL/FULL

MATCH SIMPLE/PARTIAL/FULL:

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

MATCH SIMPLE/PARTIAL/FULL

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

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

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

- **MATCH SIMPLE (par défaut) :**
  si une colonne au moins possède un marqueur NULL, la contrainte ne s'applique pas

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

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

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

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

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

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

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

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

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

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

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

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

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

UNDER

Héritage de table (norme SQL99) :

```
CREATE TABLE Personne
(    PID Integer,
     Nom                    varchar(25) NOT NULL,
     Prenom                 varchar(25) NOT NULL,
     CONSTRAINT PK_Personne PRIMARY KEY (PID),
)
```

Tables héritées de Personne :

```
CREATE TABLE Etudiant UNDER Personne
(  NumCarte  varchar(25) NOT NULL)


CREATE TABLE Enseignant UNDER Personne
( Grade                  varchar(25)
  CONSTRAINT CK_Enseignant_Grade
     CHECK (Grade IN ('Vacataire', 'Moniteur','ATER', 'MCF', 'PROF'))
)
```

GENERATED ALWAYS

Colonne calculée (norme SQL2003) :

```
CREATE TABLE Etudiant
(
 Etudiant_ID     integer,
 Ville              varchar(25),
 Nom                 varchar(25) NOT NULL,
 Prenom              varchar(25) NOT NULL,
 Date_Naissance  date NOT NULL,
 Rue                 varchar(50) DEFAULT NULL,
 CP                  varchar(9) DEFAULT NULL,
 Adresse GENERATED ALWAYS AS (rue || ' ' || CP || ' ' || Ville)
 Telephone       varchar(10) DEFAULT NULL,
 Fax             varchar(10) DEFAULT NULL,
 Email           varchar(100) DEFAULT NULL,
 CONSTRAINT PK_Etudiant PRIMARY KEY (Etudiant_ID)
);
```

Vue

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;
```

Contrainte ASSERTION

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

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

Déclencheur

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

***événement_déclencheur* = INSERT, UPDATE, DELETE**

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

Exemple de déclencheur sous Oracle

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

Exemple d'utilisation du `WHEN` (sous Oracle)

```
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

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

Définition SQL du déclencheur sous PostgreSQL

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

Exemple de déclencheur (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 (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();
```

Autre exemple de déclencheur (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);
```

Autre exemple de déclencheur (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();
```

Modification de schéma et création de vue et d'index

ALTER TABLE table
ADD (col1 type1, col2 type2 …)
| MODIFY (col1 type1, col2 type2 …)
| DROP PRIMARY KEY
| 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;
```

Chap IV - Modélisation

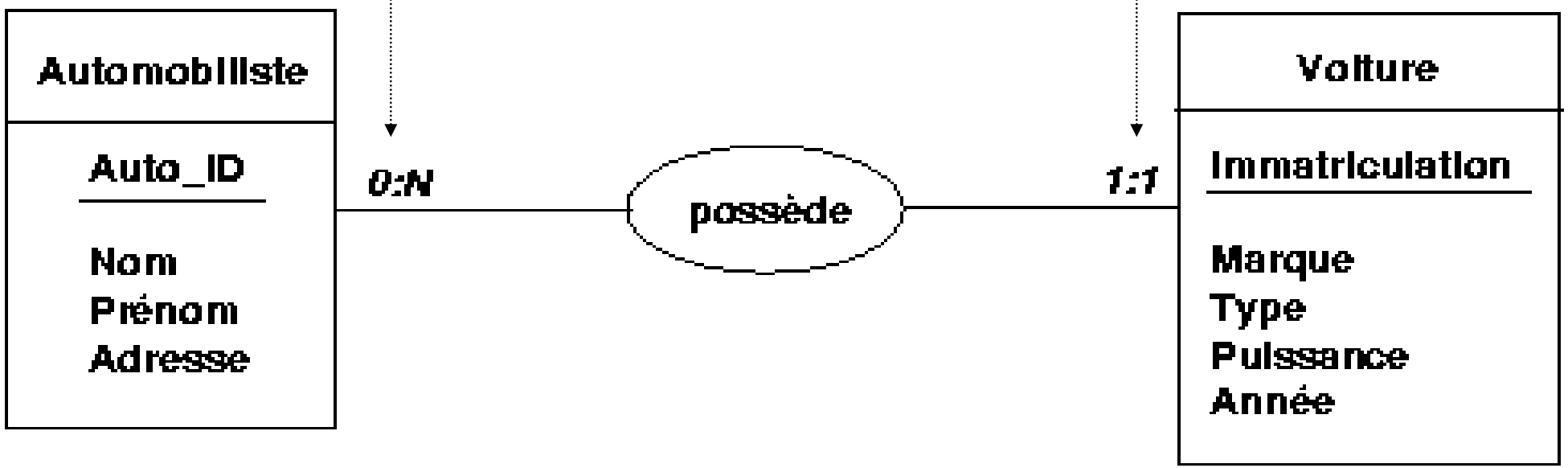
Méthodologie à suivre pour modéliser un problème

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

Modélisation Entité/Association (Format Merise)

Un automobiliste possède entre zéro et N voitures

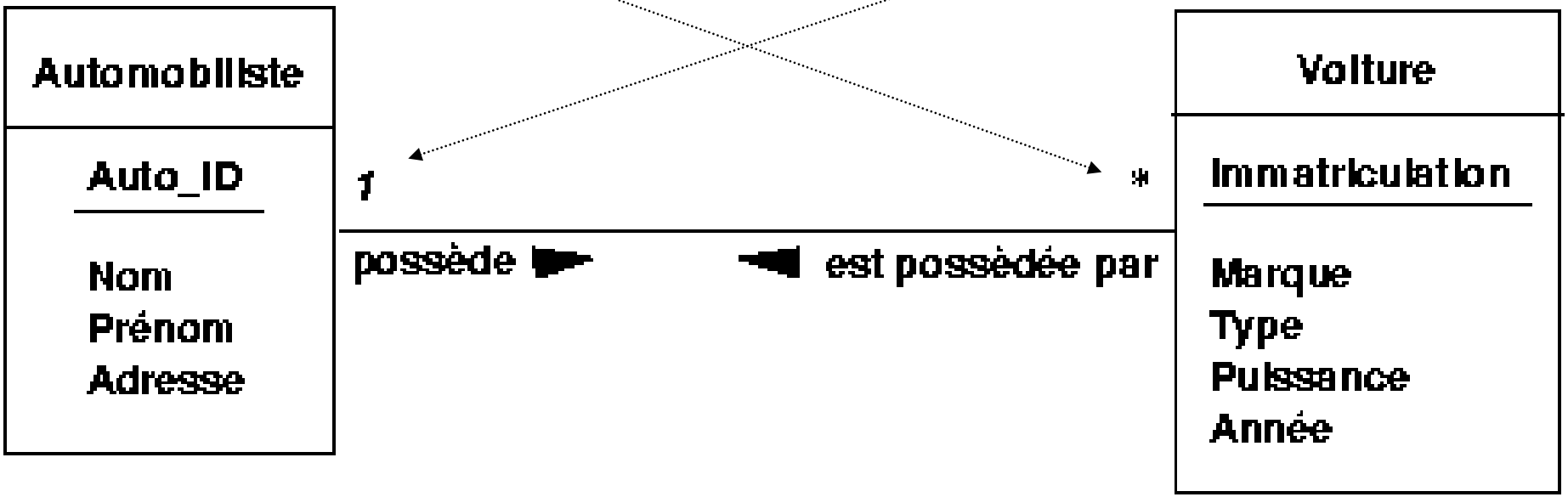
Une voiture a un et un seul propriétaire



Modélisation UML

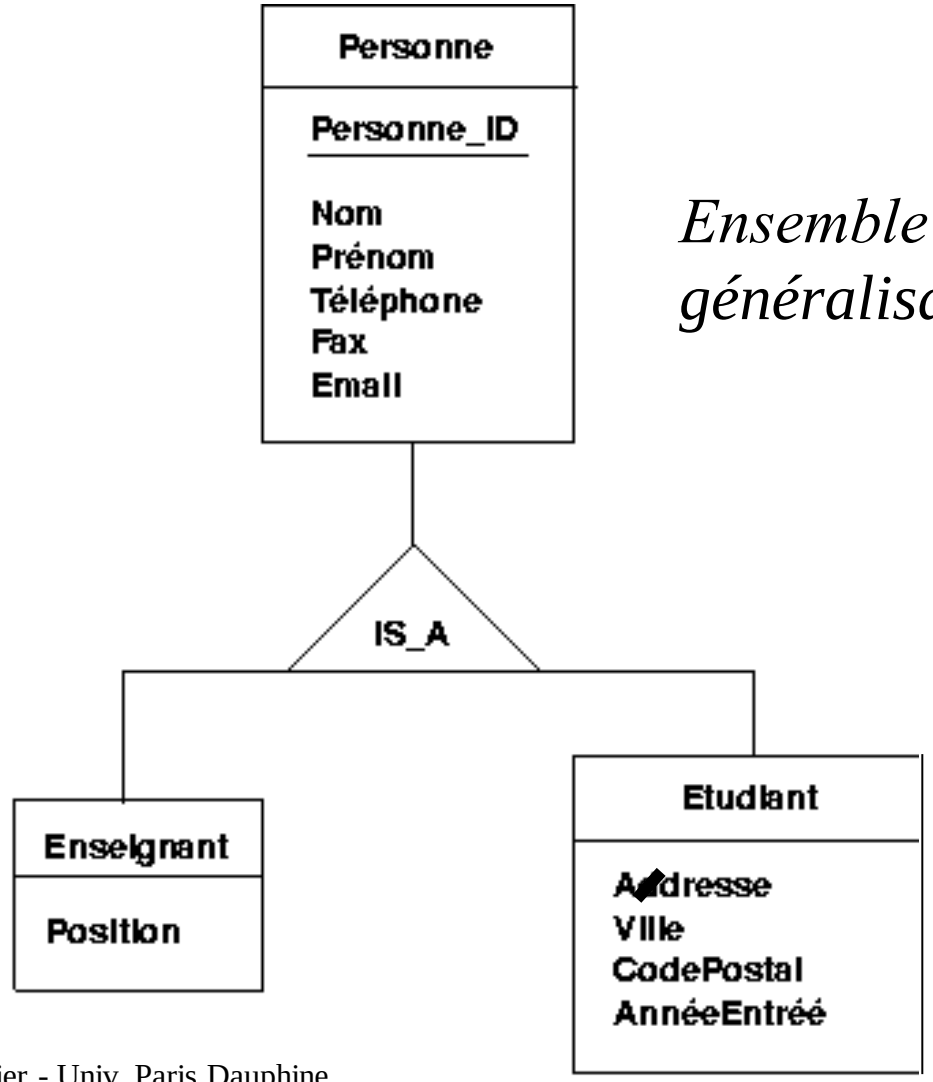
Un automobiliste possède entre zéro et N voitures

Une voiture a un et un seul propriétaire



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

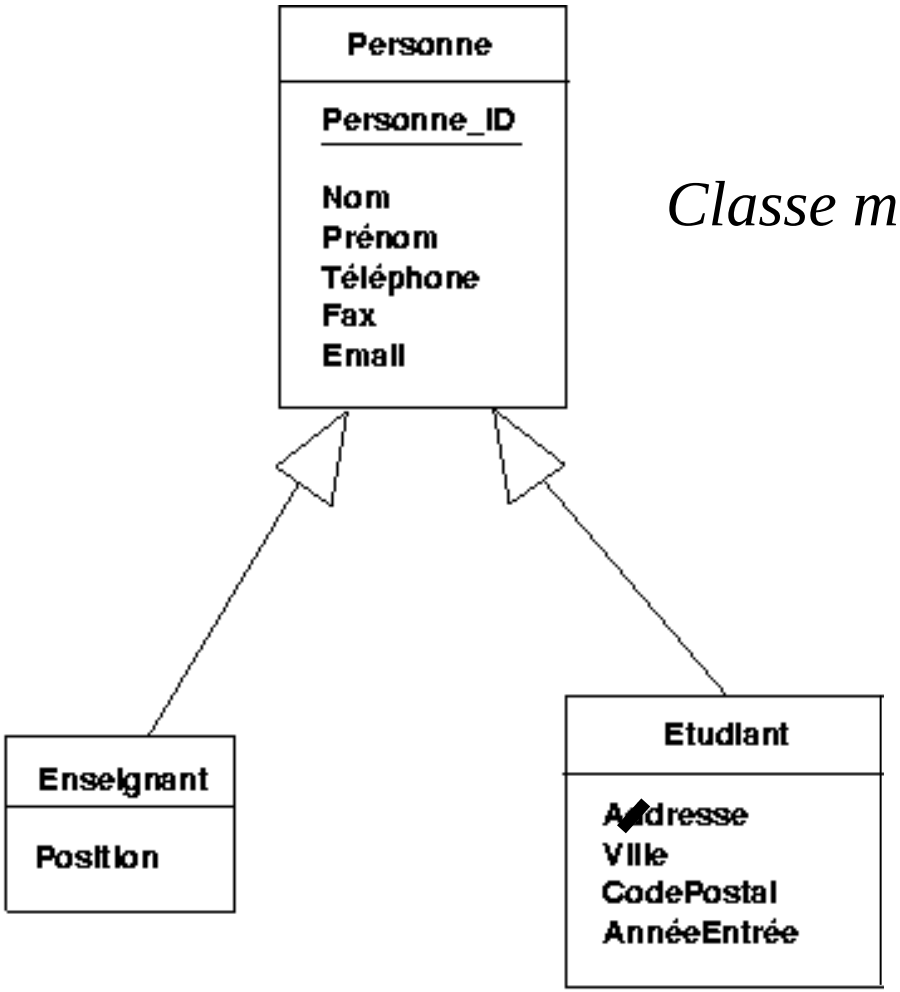
Généralisation/Spécialisation (E/A - Merise)



Ensemble d'entités généralisantes

Ensemble d'entités spécialisées

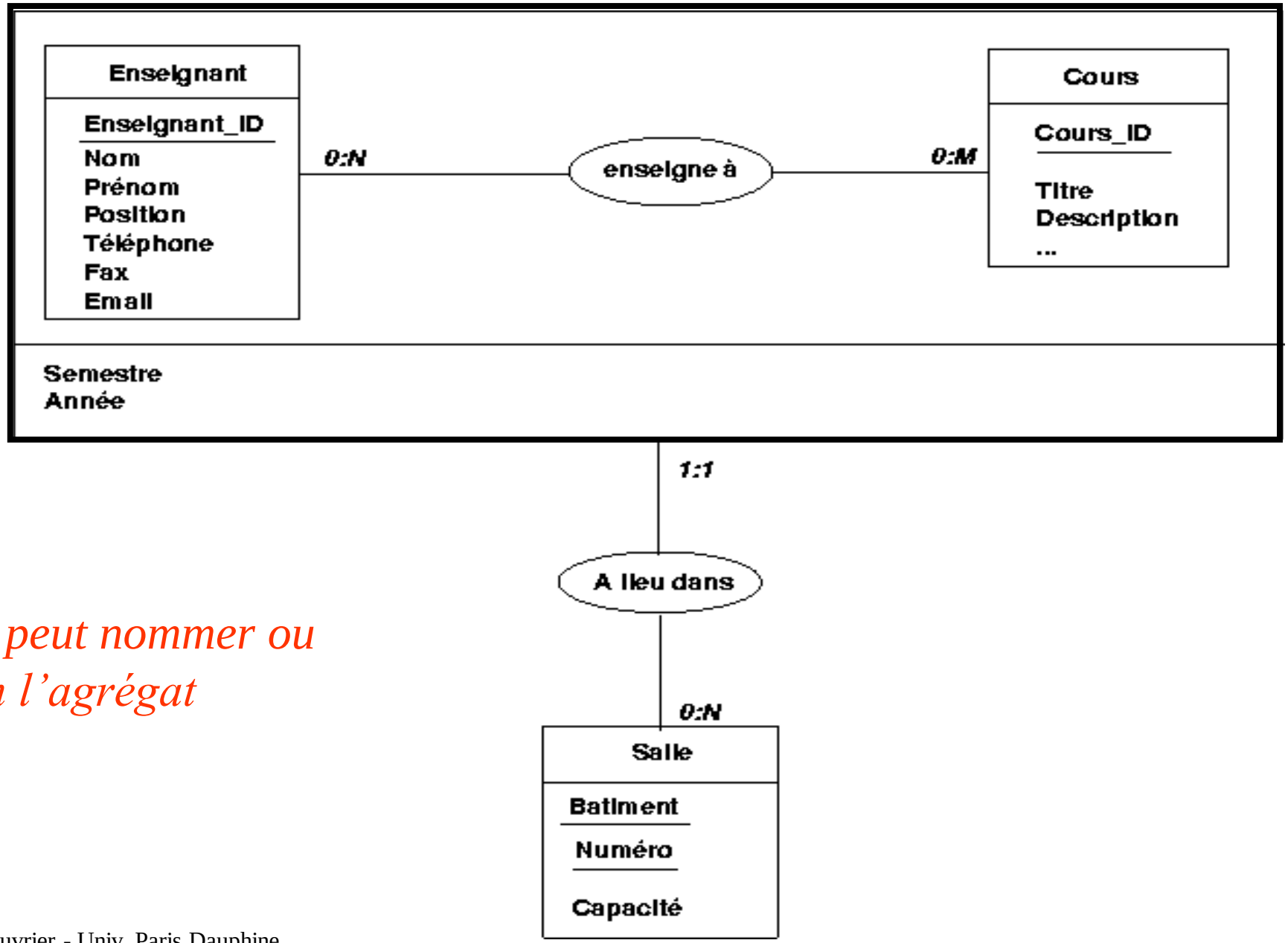
Héritage (UML)



Classe mère / Sur-classe

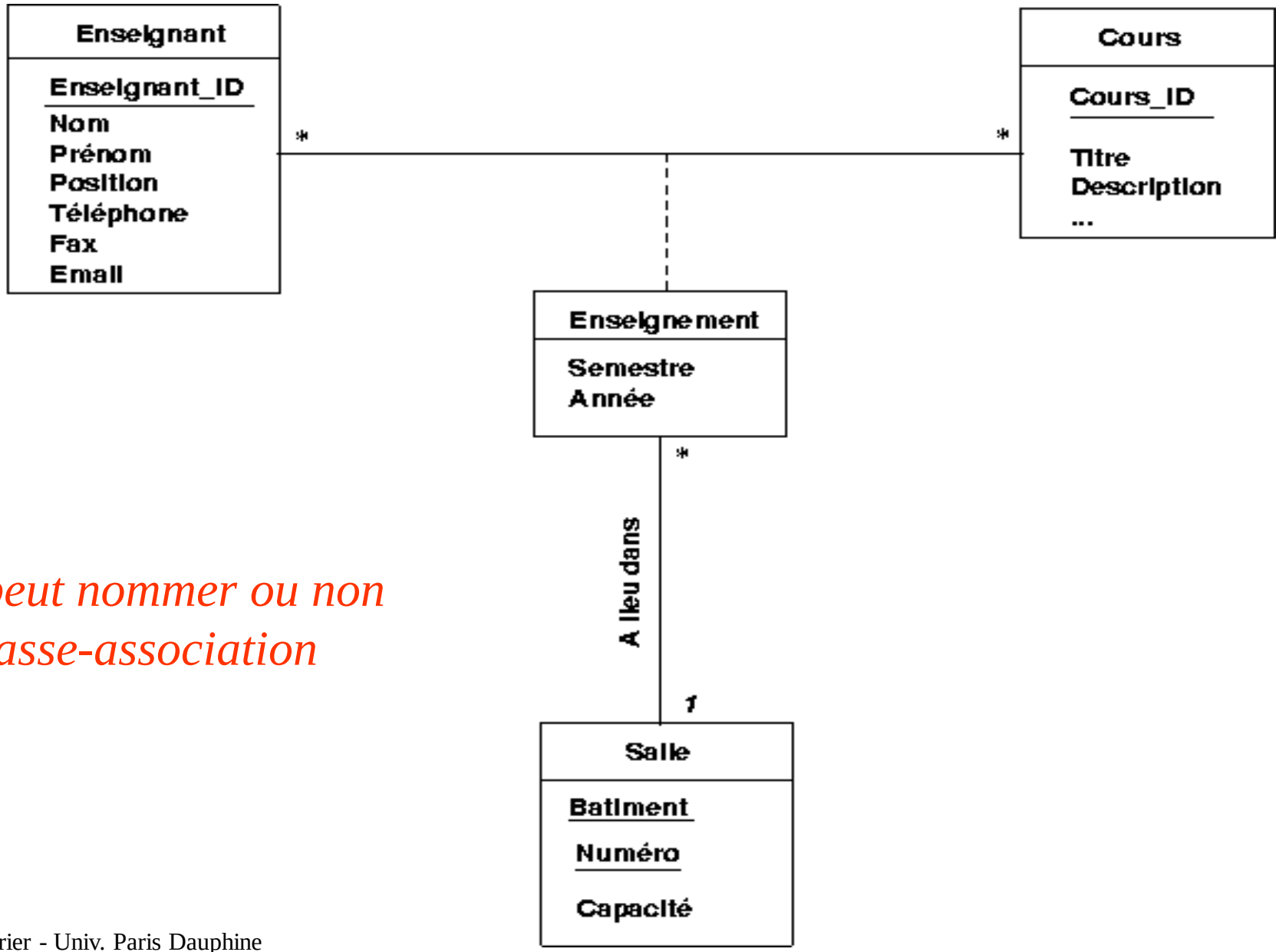
Classes dérivées ou filles / sous-classes

Agrégat (E/A - Merise)



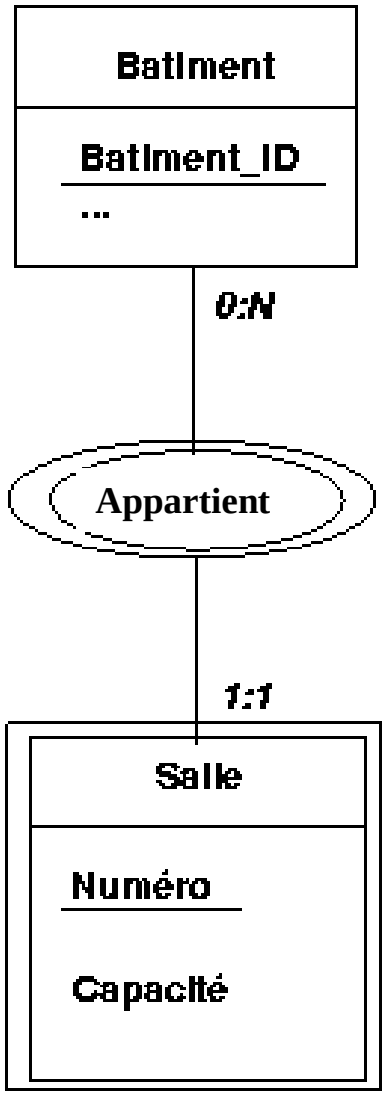
On peut nommer ou non l'agrégat

Classe-Association (UML)



On peut nommer ou non la classe-association

Entité Faible (E/A - Merise)

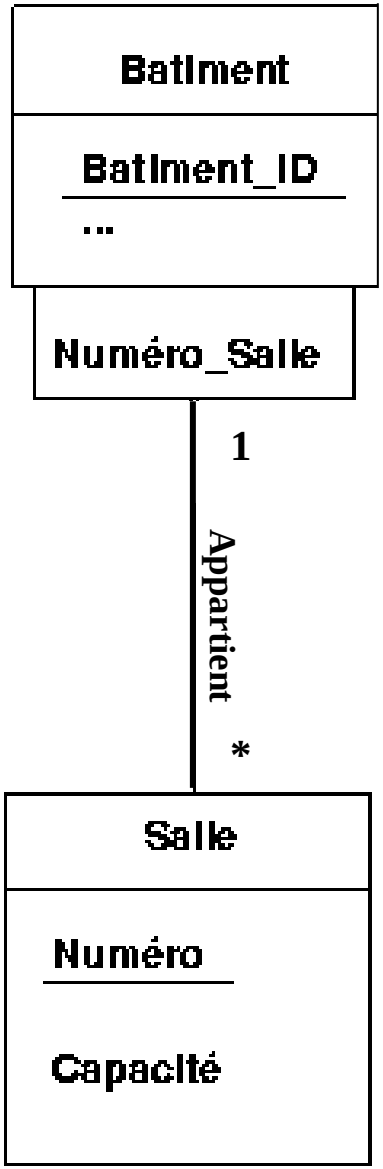


Chaque salle a un numéro unique dans un bâtiment donné

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

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

Association qualifiée (UML)

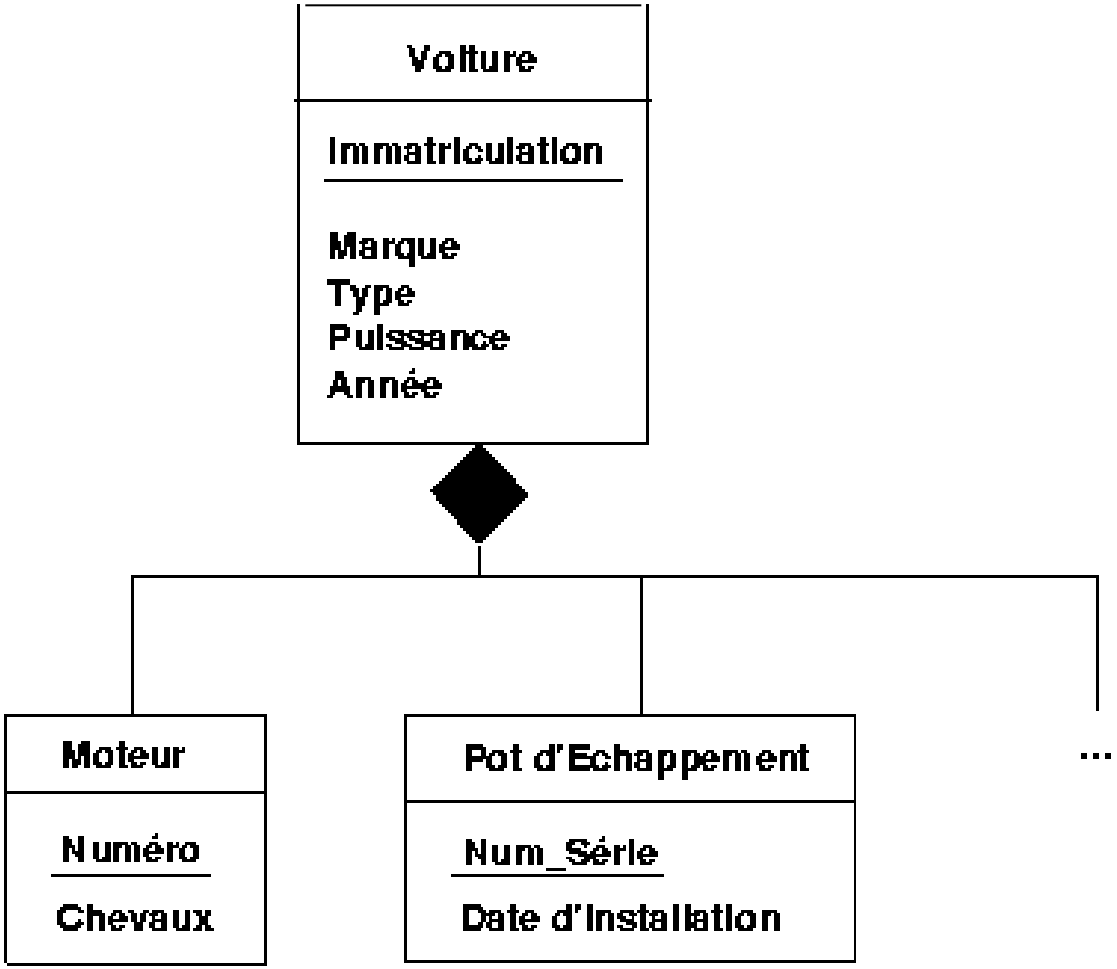


Chaque salle a un numéro unique dans un bâtiment donné

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

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

Composition (UML)



Passage au relationnel

2 étapes :

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

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

2. **Transformations des associations :**

   a) **Suppression/Fusion de relations créées à l'étape 1**

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

   c) **Ajout de nouvelles relations**

   En fonction des cardinalités/multiplicités

Passage au relationnel

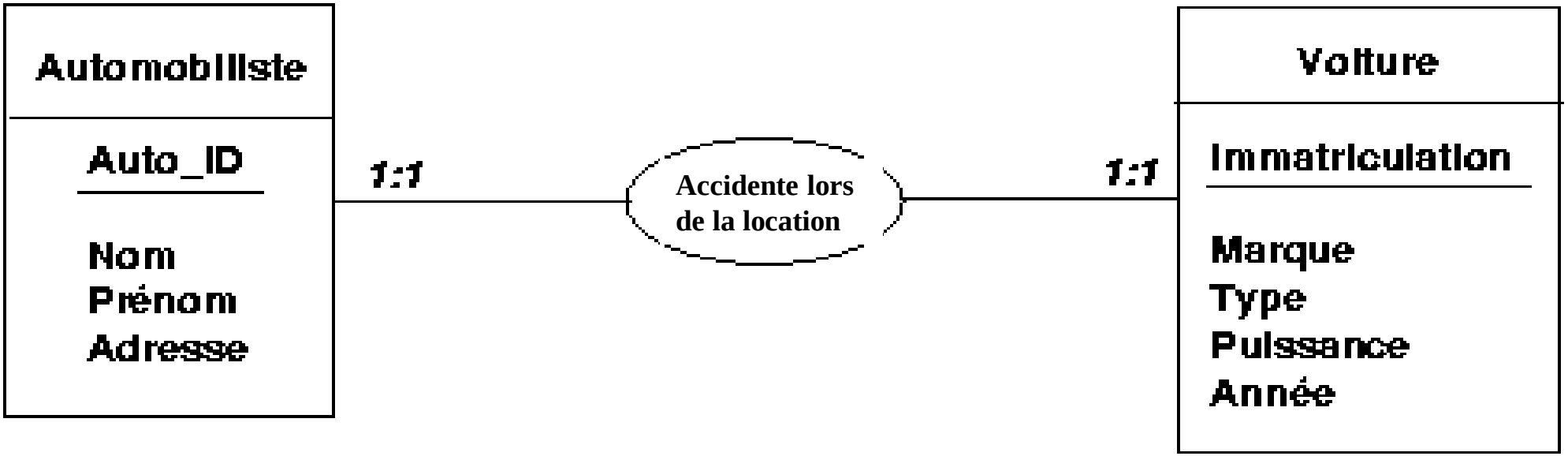
Transformation des ensembles d'entités :

- **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/association qualifiée *E* ⇒**

  une relation R qui comprend tous les attributs de E +

  l'identificateur de l'ensemble d'entités fortes/classe associé(e)
- **généralisation-spécialisation/héritage ⇒**
  - l'ensemble d'entités généralisante/classe mère *E* ⇒ une relation *R*
  - chaque ensemble d'entités E_i spécialisé/classe fille
    ⇒ une relation R_i dans laquelle l'identifiant est de même domaine que l'identifiant de E et est une clé étrangère faisant référence à la clé primaire de E

Transformation des ensembles d'associations 1:1 en E/A

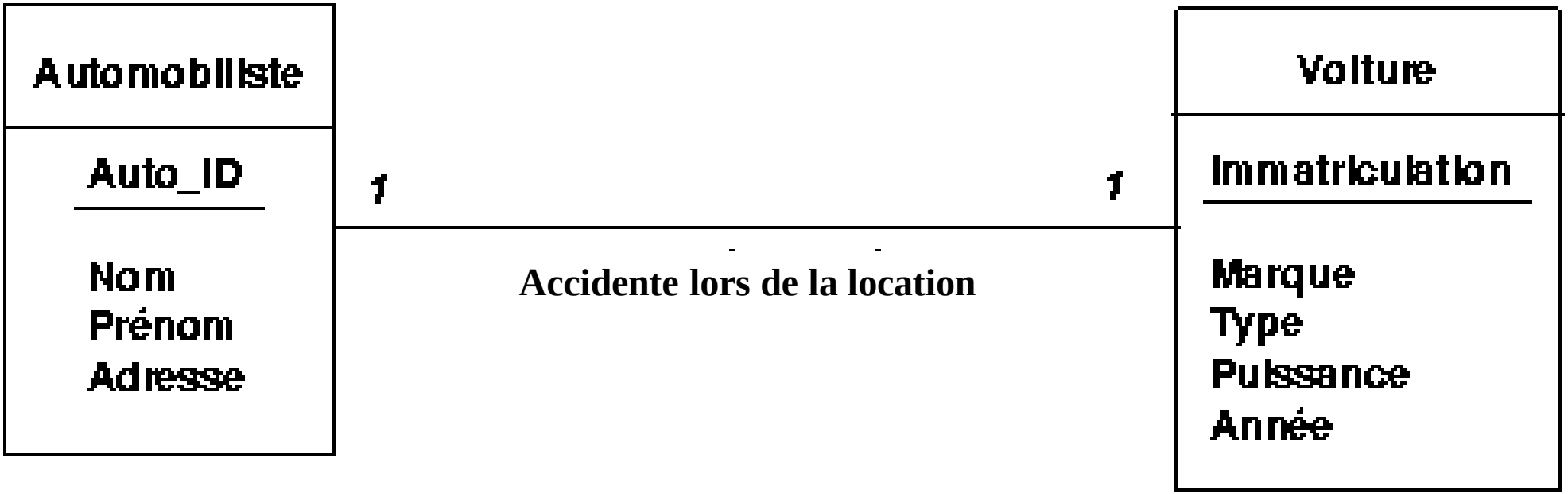


Automobiliste (Auto_ID, Nom, Prénom, Adresse)

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

Comment faire le lien ?

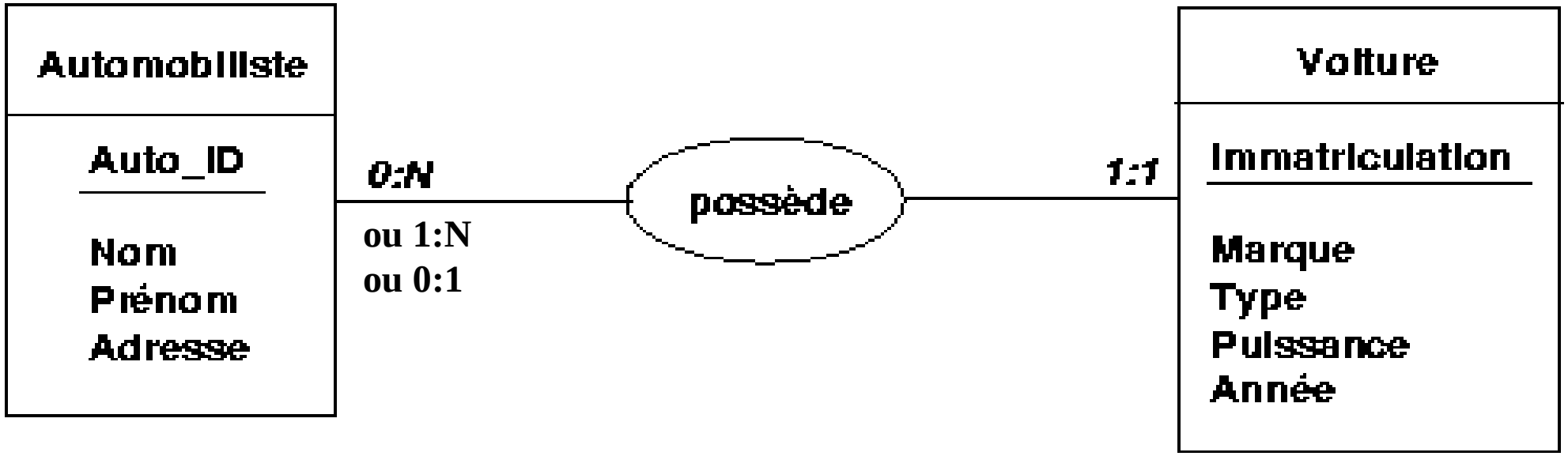
Transformation des ensembles d'associations 1..1 en UML



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

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

Transformation des ensembles d'associations 1:N en E/A

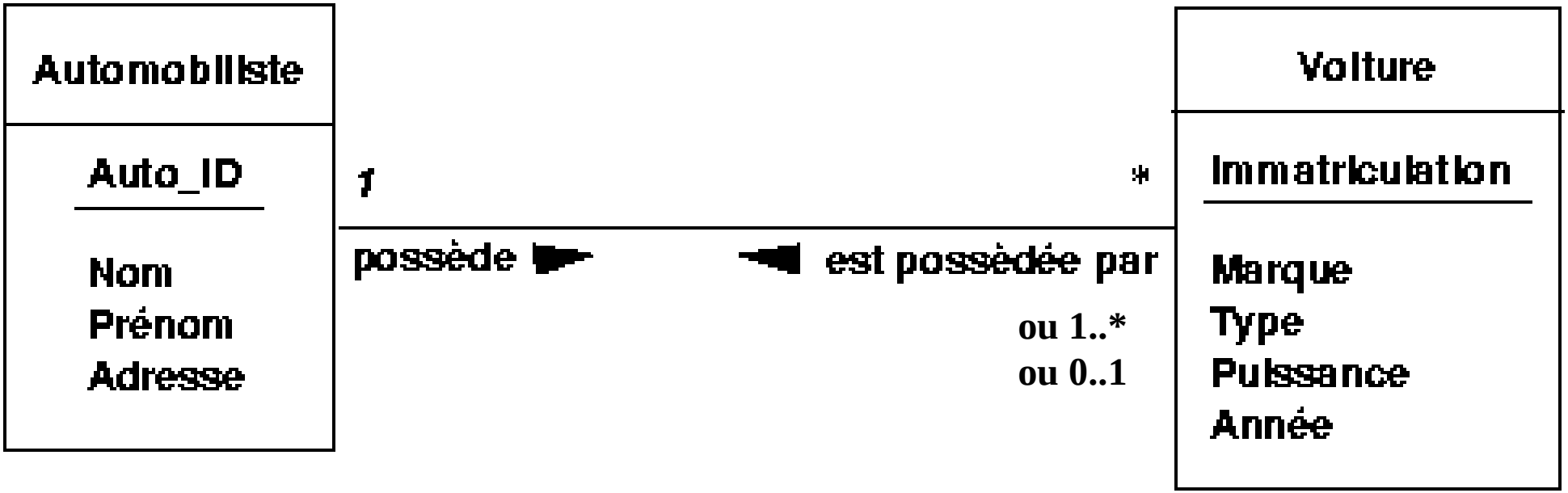


Automobiliste (Auto_ID, Nom, Prénom, Adresse)

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

NB : *#Auto_ID fait référence à Auto_ID de Automobiliste*

Transformation des ensembles d'associations 1..* UML

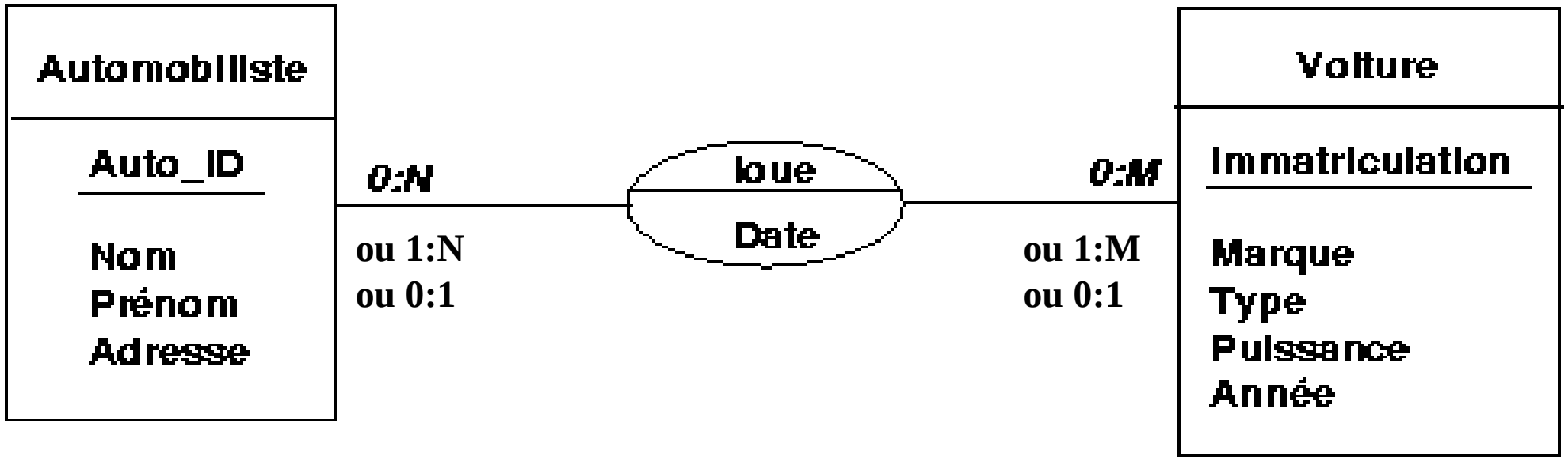


Automobiliste (Auto_ID, Nom, Prénom, Adresse)

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

NB : *#Auto_ID fait référence à Auto_ID de Automobiliste*

Transformation des ensembles d'associations N:M en E/A



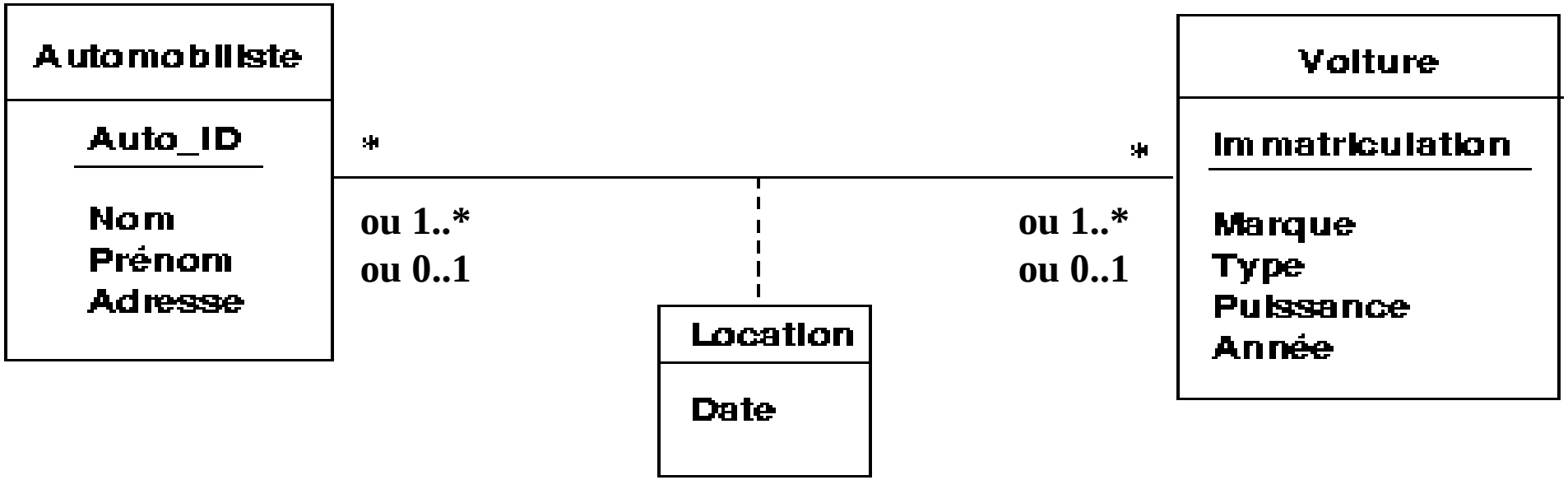
Automobiliste (Auto_ID, Nom, Prénom, Adresse)

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

Location (#Auto_ID, #Immatriculation, Date) ou

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

Transformation des ensembles d'associations *:* en UML



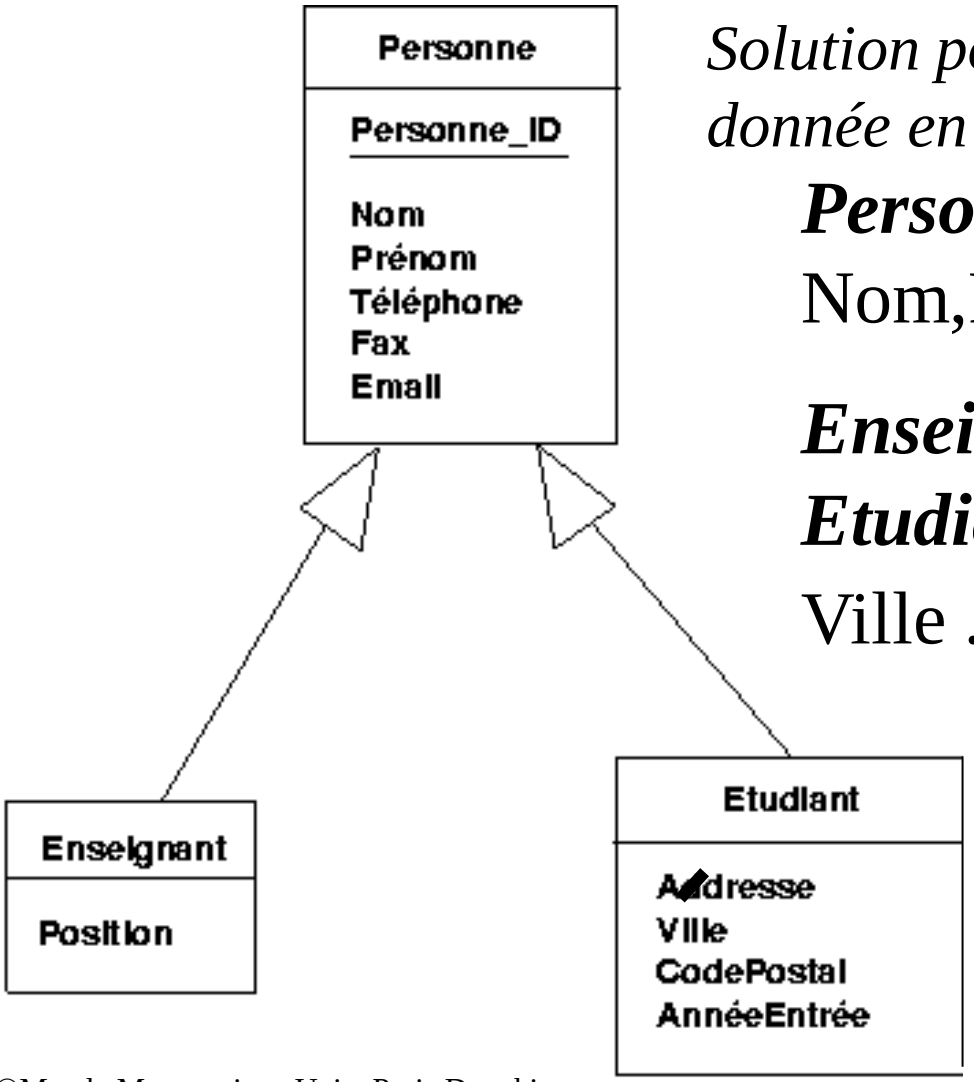
Automobiliste (Auto_ID, Nom, Prénom, Adresse)

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

Location (#Auto_ID, #Immatriculation, Date) ou

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

Transformation des concepts Généralisation-Spécialisation / Héritage



Solution possible (une autre sera donnée en cours) :

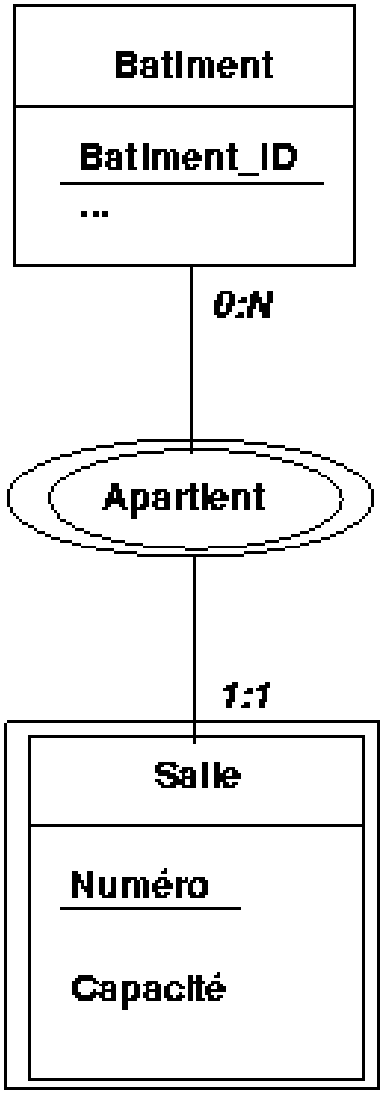
Personne (Personne_ID, Nom,Prénom, Téléphone ...)

Enseignant (#Personne_ID, Position)

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

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

Transformation des entités faibles E/A



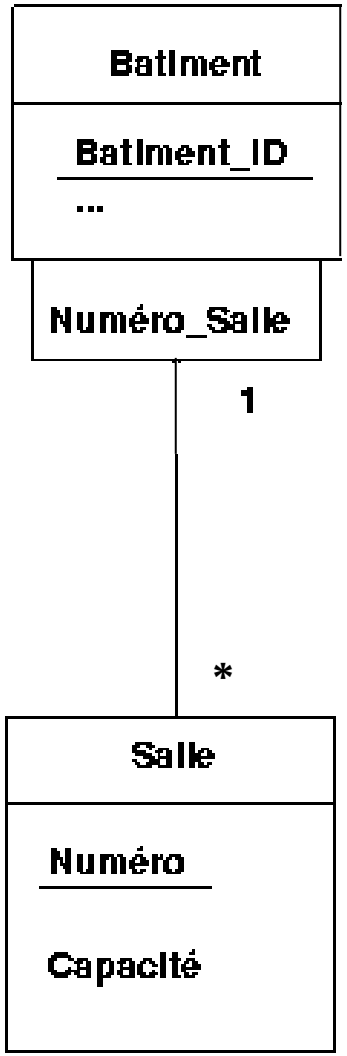
Bâtiment (Bâtiment_ID, ...)

Salle (Numéro, #Bâtiment_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



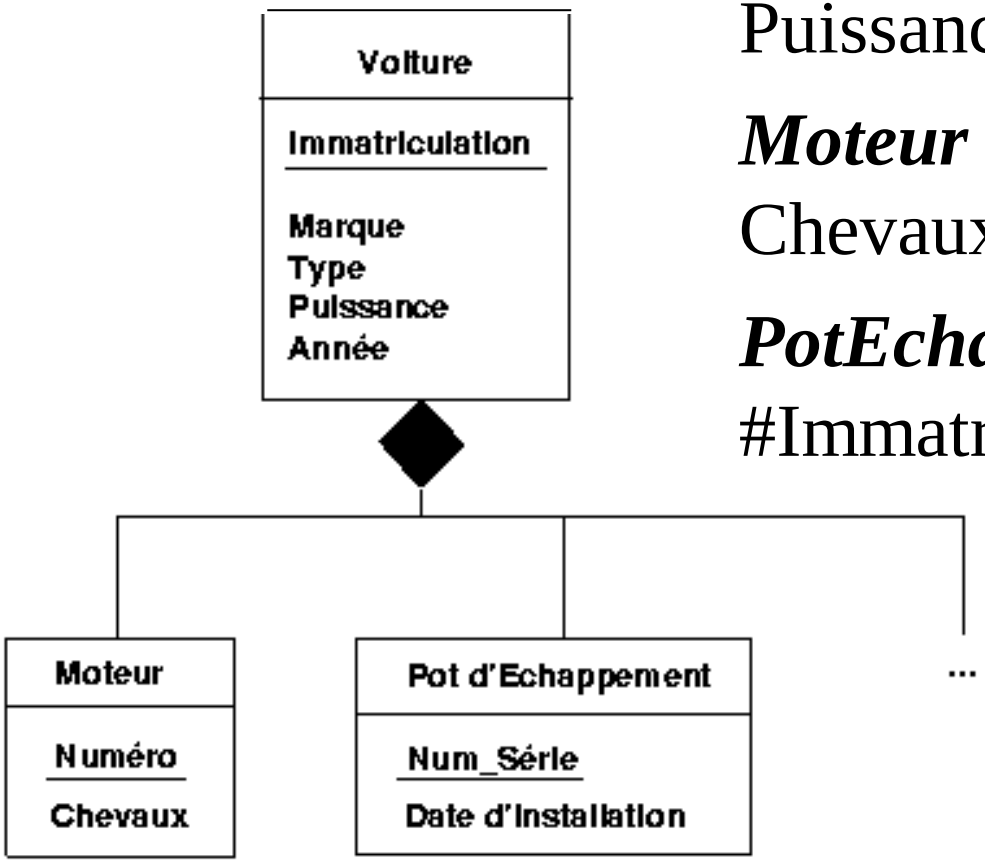
Bâtiment (Bâtiment_ID, ...)

Salle (Numéro, #Bâtiment_ID, Capacité)

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

#Bâtiment_ID fait référence à Bâtiment_ID de Bâtiment

Transformation de la composition UML



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

Moteur (Numéro, #Immatriculation, Chevaux)

PotEchappement (Num_Série, #Immatriculation, DateInstallation)