

Introduction aux BD

Plan général du cours

- Introduction et modélisation BD
- SQL introduction
- SQL et calcul relationnel
- *SQL : Interrogation*
- *SQL : Mises-à-jour*
- Dépendances fonctionnelles
- Normalisation de schémas relationnels
- Transactions et tolérance aux pannes
- Contrôle de concurrence

• *Modèles de données*
• *et langages logiques :*
• *conception*
• *interrogation*
• *mise-à-jour, cohérence*

Théorie de la conception

Performances et cohérence d'exécution

administrateur
r
développeur
utilisateur

Introduction aux Bases de Données

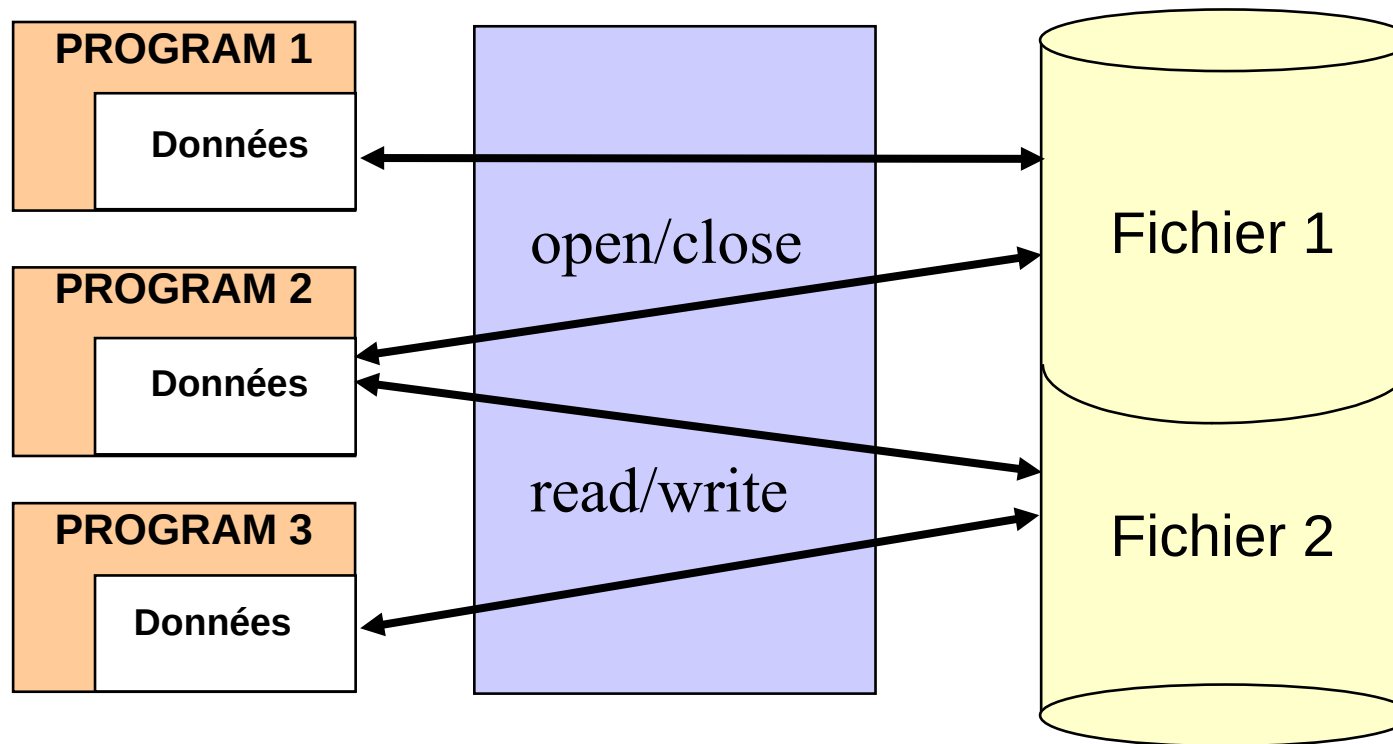
- **Fichiers et Bases de Données**
- Systèmes de Gestion de Bases de Données (SGBD)
- Langages et modèles de données
- Modèle Entité-Association
- Modèle Relationnel

Qu'est-ce qu'une « Base de Données (BD) » ?

- Une **base de données** (BD) est une *collection de données structurées* sur des entités (objets, individus) et des relations dans un contexte (applicatif) particulier.
- Un **système de gestion de base de données** (SGBD) est *un (ensemble de) logiciel(s)* qui facilite la création et l'utilisation de bases de données.
- Les données sont définies, administrées et gérées en utilisant des langages fondés sur des modèles de données.

Fichier $\neq$ Bases de Données

- Fichiers :
 - ♦ opérations simples : ouvrir/fermer, lire/écrire
 - ♦ différentes méthodes d'accès (séquentiel, indexé, haché, etc.)
 - ♦ utilisation par plusieurs programmes difficile (format ?)

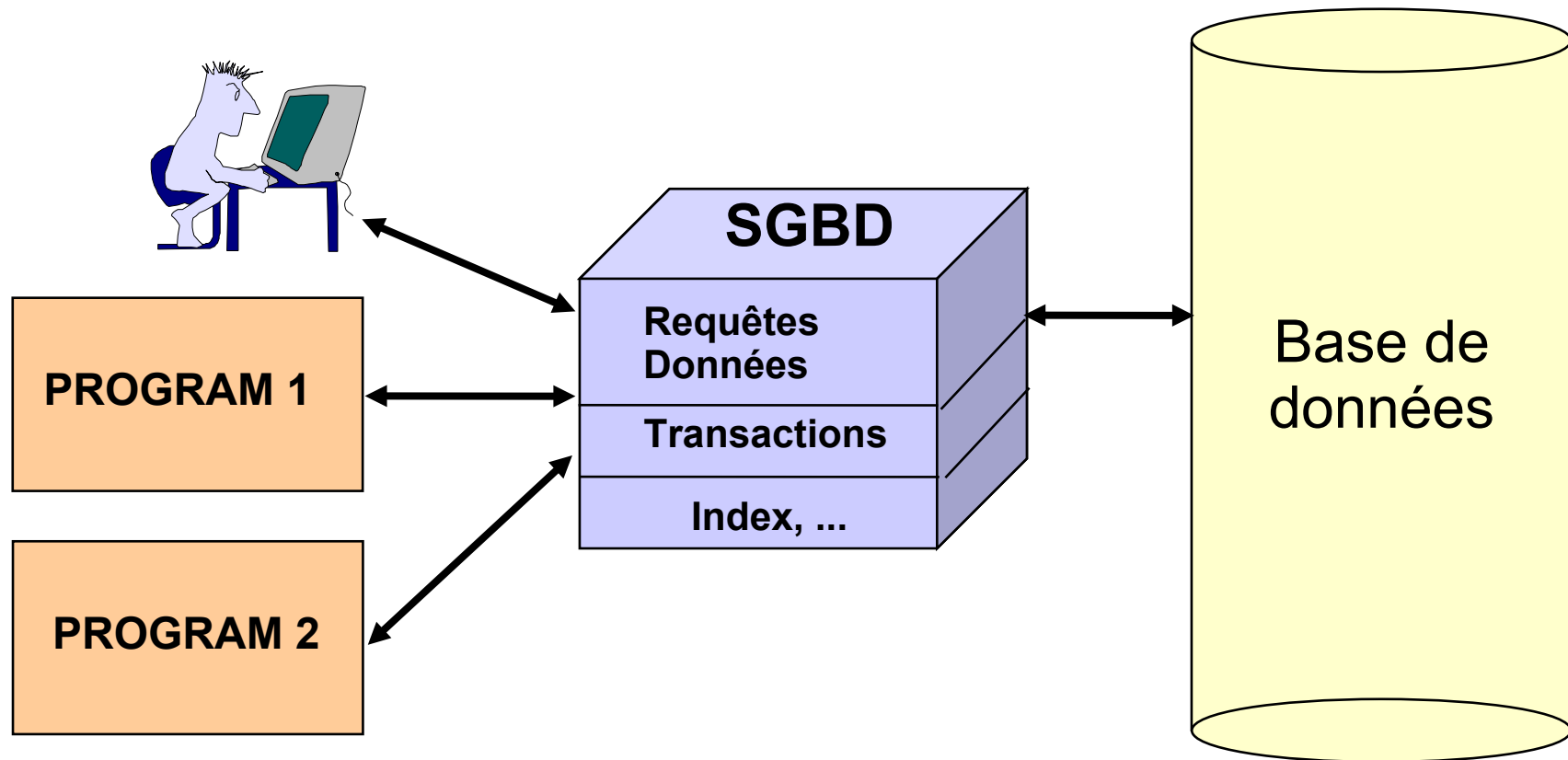


Propriétés des fichiers

- Faible structuration des données
 - ♦ données « plates »
- Dépendance entre programmes et fichiers
 - ♦ modification du *format de stockage* implique modification des programmes
- Redondance des données
 - ♦ partage de données au niveau du fichier
- Absence de contrôle de cohérence globale des données

- La gestion de données structurées dans des fichiers représente une partie importante du coût de développement et de maintenance d'applications.

Approche « Base de Données »



Problèmes avec les fichiers résolus avec une base de données

- **Fichier :**

- Faible structuration des données
- Dépendance entre programmes et fichiers
- Redondance des données
- Absence de contrôle de cohérence globale des données



- **Base de Données :**

- Structuration des données à travers un schéma de données
- Indépendance entre programmes et données
- Données partagées
- Contrôle de la cohérence logique et physique (schémas, transactions)

Introduction aux Bases de Données

- Fichiers et Bases de Données
- Systèmes de Gestion de Bases de Données (SGBD)
- Langages et modèles de données
- Modèle Entité-Association
- Modèle Relationnel

Qu'est-ce qu'un « SGBD » ?

- Une **base de données** (BD) est une *collection de données structurées* sur des entités (objets, individus) et des relations dans un contexte (applicatif) particulier.
- Un **système de gestion de base de données** (SGBD) est *un (ensemble de) logiciel(s)* qui facilite la création et l'utilisation de bases de données.
- Les données sont définies, administrées et gérées en utilisant des langages fondés sur des modèles de données.

Système de Gestion de Bases de Données (SGBD)

- Objectif : faciliter le partage de grands volumes de données entre différents utilisateurs / applications
- Un SGBD doit garantir
 - la **cohérence** et l'**intégrité** des données en cas d'erreurs de programmation, d'accès concurrents, de pannes, d'accès non-autorisés, ...
 - des **performances** d'accès sur des grands volumes de données pour des grands nombres de clients

Approche Bases de Données : Séparation en couches indépendantes

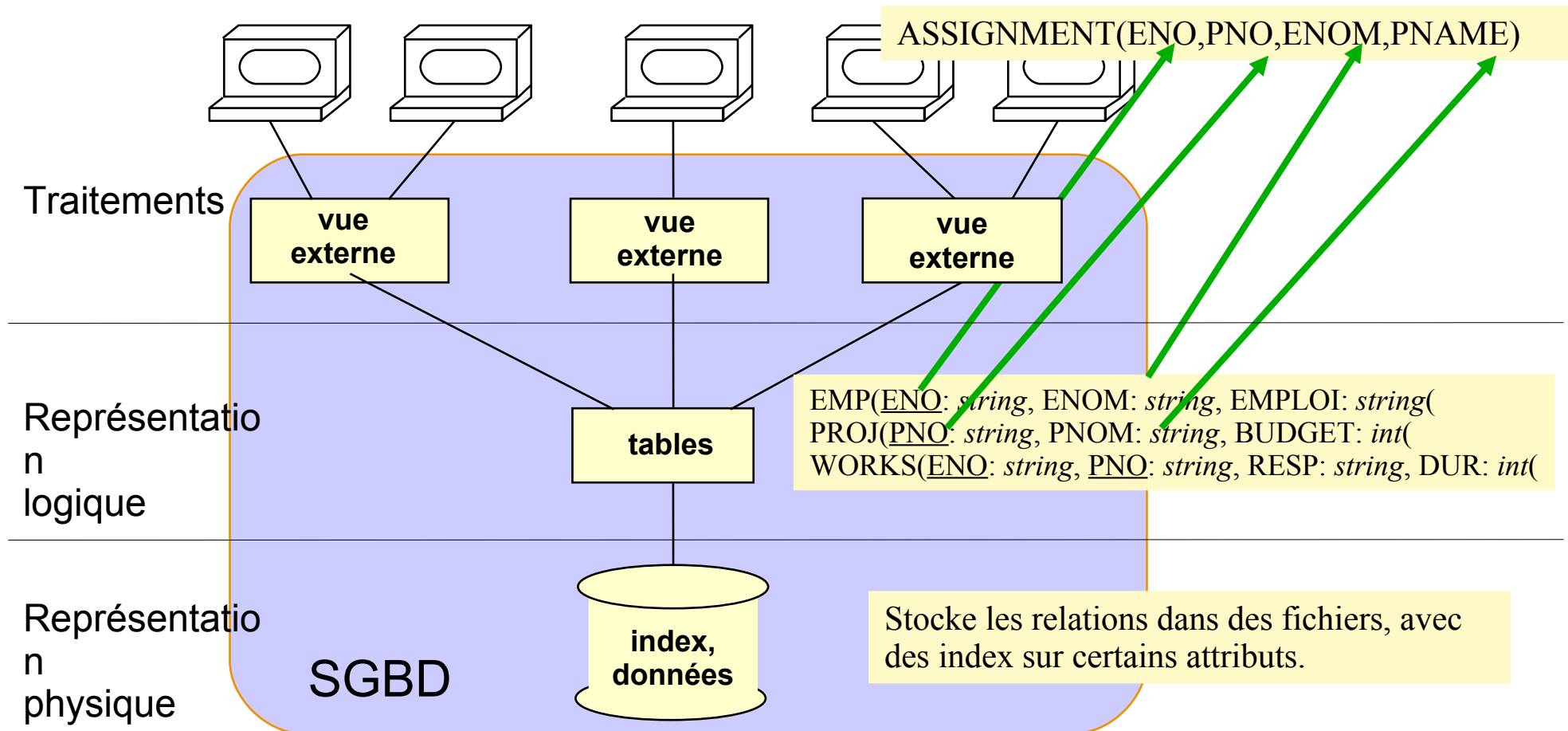
- Séparation le problème de la gestion de données en trois « couches » :

- Traitements (calcul, affichage, ...)
- **Représentation logique des données**
- **Représentation physique des données**



- Couche = ensemble de sous-problèmes bien définis :
 - ♦ Indépendance *traitements/représentation logique des données* : vues externes cachent les détails de l'organisation logique des données
 - ♦ Indépendance *représentation logique/représentation physique* : schéma logique cache les détails du stockage physique des données (organisation sur disque, index, ...)

Architecture ANSI/SPARC



Fonctions d'un SGBD

- Représentation et structuration de l'information :
 - ◆ Description de la *structure des données* : Employés, salaires, noms,
 - ◆ Description de *contraintes logiques sur les données* : $0 < \text{Age} < 150$, ...
 - ◆ *Vue* : *réorganisation (virtuelle)* de données pour des besoins spécifiques
- Gestion de l'intégrité des données :
 - ◆ Vérification des contraintes spécifiées dans le schéma
 - ◆ Exécution *transactionnelle* des requêtes (mises-à-jours)
 - ◆ Gestion de la concurrence multi-utilisateur et des pannes
- Traitement et optimisation de requêtes :
 - ◆ La performance est un problème géré par l'administrateur du SGBD et non pas par le développeur d'application (indépendance physique)

Introduction aux Bases de Données

- Fichiers et Bases de Données
- Systèmes de Gestion de Bases de Données (SGBD)
- **Langages et modèles de données**
- Modèle Entité-Association
- Modèle Relationnel

Qu'est-ce qu'un «Modèle de Données » ?

- Une **base de données** (BD) est une *collection de données structurées* sur des entités (objets, individus) et des relations dans un contexte (applicatif) particulier.
- Un **système de gestion de base de données** (SGBD) est *un (ensemble de) logiciel(s)* qui facilite la création et l'utilisation de bases de données.
- *Les données* sont définies, administrées et gérées en utilisant des **langages** fondés sur des **modèles de données**.

Utilisateurs d'un SGBD

- Utilisateur final (end user) :

- ◆ accède la BD par des formes d'écran, des interfaces applicatives ou, pour les plus experts, des requêtes ad hoc (SQL)

- Développeur d'applications :

- ◆ construit (avec l'utilisateur) le schéma conceptuel

- ◆ définit et gère le schéma logique et les vues

- ◆ conçoit et implémente des applications qui accèdent la BD

- Administrateur BD (DBA)

- ◆ gère le schéma physique et règle les performances (tuning)

- ◆ charge et organise la BD

- ◆ gère la sécurité et la fiabilité

Langages et interfaces d'un SGBD

- Langages de *conception* : E/A, UML
 - ◆ Utilisation : conception *haut-niveau* d'applications (données et traitements)
- Langage base de données : SQL
 - ◆ langage déclaratif : l'utilisateur spécifie *quoi* (et non *comment*)
 - ◆ puissance d'expression limitée (par rapport à un langage de programmation comme C ou Java)
 - ◆ utilisation : *définition schémas, interrogation et mises-à-jour, administration*
- Langages de *programmation* : PL/SQL, Java, PHP, ...
 - ◆ langages impératifs avec une interface SQL
 - ◆ langage complet (au sens d'Alan Turing)
 - ◆ utilisation : *programmation d'applications complètes*

Langages BD (SQL)

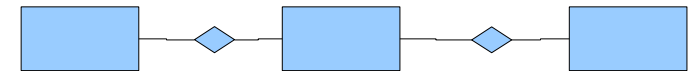
- Langage de Définition de Données (LDD)
 - ♦ pour définir les schémas externes (vues), logiques et physiques
 - ♦ les définitions sont stockées dans le répertoire système (dictionnaire)
- Langage de manipulation de données (LMD)
 - ♦ langage déclaratif pour interroger (langage de requêtes) et mettre à jour les données
 - ♦ peut être autonome (par ex. SQL seul) ou intégré dans un langage de programmation

Modèles de données

- Modèle de données = langage + **sémantique** pour représenter et manipuler des données



- Modèle conceptuel : *conception*



- *structuration haut-niveau (conceptuelle)* de l'information (pas d'opérations)
- modèle *entité-association (E/A)*, UML, Merise, ...

- Modèle logique : *conception et développement*

- *définition et utilisation* des données dans le SGBD
- modèle hiérarchique, réseau, **relationnel**, objet

```
R(A,B);  
select A from R where B=2;
```

- Modèle physique : *administration*

- *organisation physique des données et implantation des opérations*
- modèles de stockage sur disque, indexes, algorithmes ...

```
use index RI;  
read record r;
```

Introduction aux Bases de Données

- Fichiers versus Bases de Données
- Système de Gestion de Bases de Données (SGBD)
- Langages et modèles de données
- Modèle Entité-Association
- Modèle Relationnel

Le modèle entité-association (E/A)

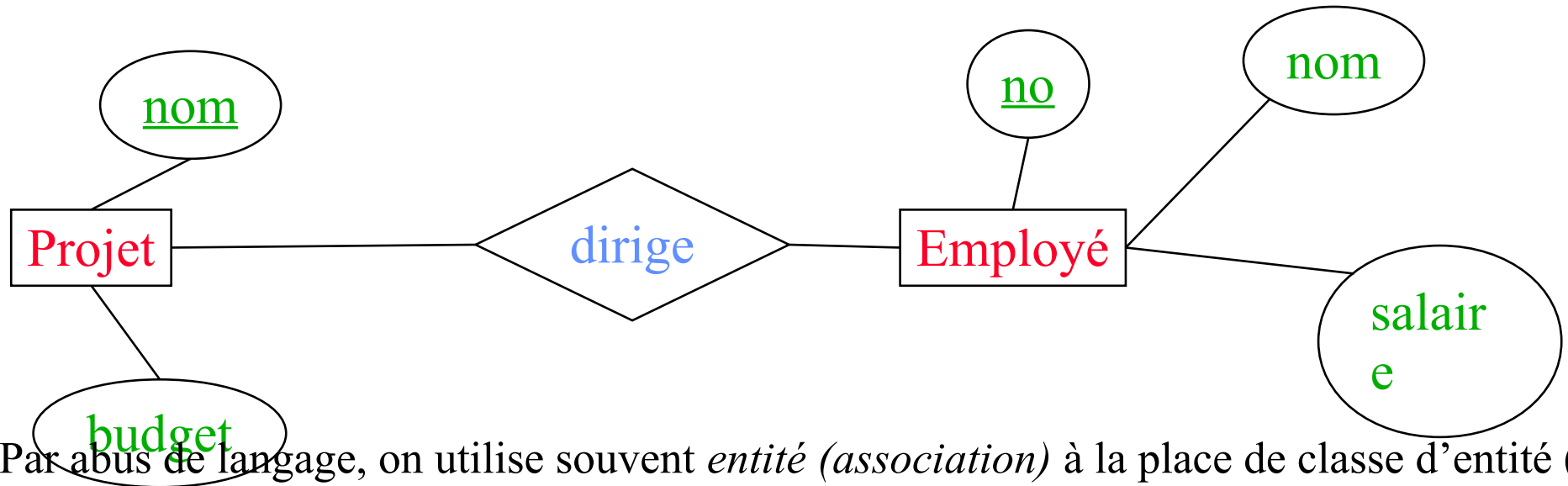
- « E/R (entity-relationship) model » en anglais
-
- Modèle de *conception* / schémas conceptuels :
 - Modélisation *graphique* des entités, de leurs attributs et des associations entre entités
 - *Détection d'erreurs de conception* avant le développement !
 - Traduction automatique dans un modèle logique (relationnel)
- Supporté par les outils pour BD
- ~ partie “modélisation de données” dans UML

Les concepts

- **Entité** : un objet qui *existe* dans le monde réel et possède
 - ♦ une *identité* et
 - ♦ des *propriétés*
- Exemples :
 - ♦ l'employé Martin, le projet Compilateur
- **Association** : une « relation » entre deux ou plusieurs entités
- Exemples :
 - ♦ l'employé Smith *dirige* le projet Compilateur
 - ♦ l'employé Martin *travaille dans* le projet Compilateur
- **Attribut** : propriété d'une **entité** ou d'une **association**
 - ♦ prend ses valeurs dans un domaine (string, [1..10], etc.)
- Exemples :
 - ♦ le No de l'employé Smith est 10
 - ♦ la durée de l'affectation de Martin dans le projet Compilateur est 6

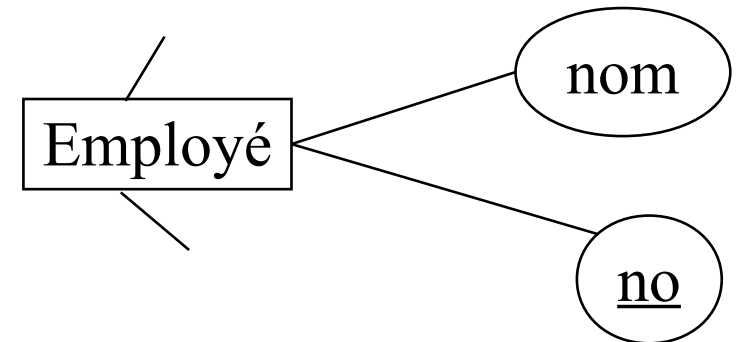
Classe d'entités et d'associations

- Une **classe d'entités** est un ensemble d'entités similaires, ayant les mêmes **attributs**
- Une **classe d'associations** est un ensemble d'associations entre les entités d'une ou de plusieurs classes.



Identificateurs

- Identificateur d'entité
 - ♦ *un ou plusieurs attributs permettant d'identifier une entité dans une classe d'entités*
 - ♦ Exemple :
 - Employé No d'employé

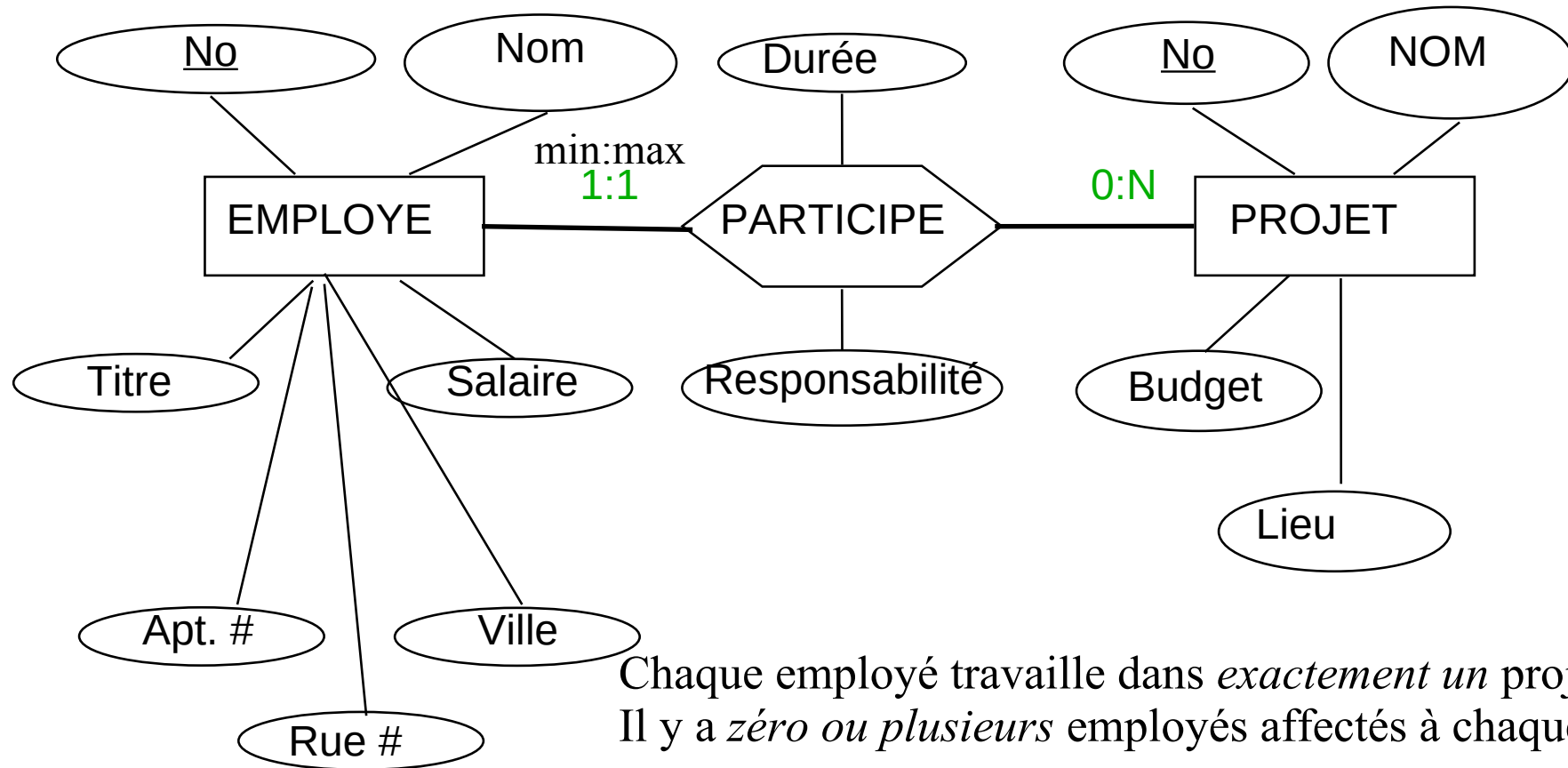


- Identificateur d'association
 - ♦ *un identificateur composé de tous les identificateurs d'entités reliées par l'association*
 - ♦ exemple
 - Travaille No d'employé, Nom de projet

Cardinalités d'une classe d'associations

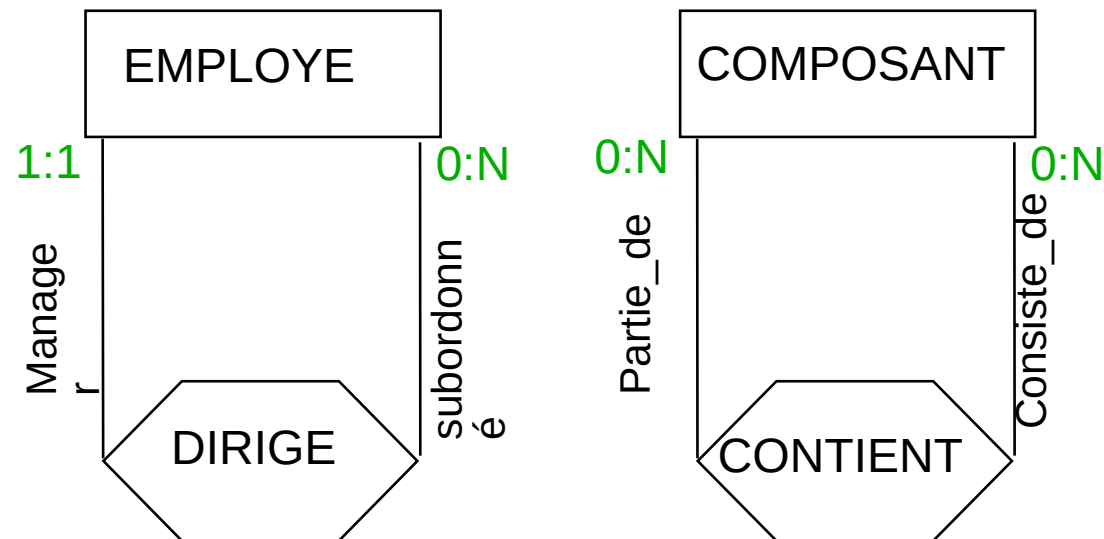
- Un intervalle **[min:max]** indique pour une classe d'entités C et une classe d'associations A, le nombre d'associations de *type* A qu'une *entité* de C peut avoir avec d'autres entités.
-
- Exemples : **[min:max]**
- un employé peut travailler dans un ou plusieurs projets [1:N]
- un projet peut avoir zéro ou plusieurs employés [0:N]
- un employé peut travailler dans zéro ou un projet [0:1]
- un projet peut avoir zéro ou plusieurs employés [0:N]
- un employé doit travailler dans *exactement* un projet [1:1]
- un employé doit travailler dans au moins deux projets [2:N]
-

Schéma entité-association

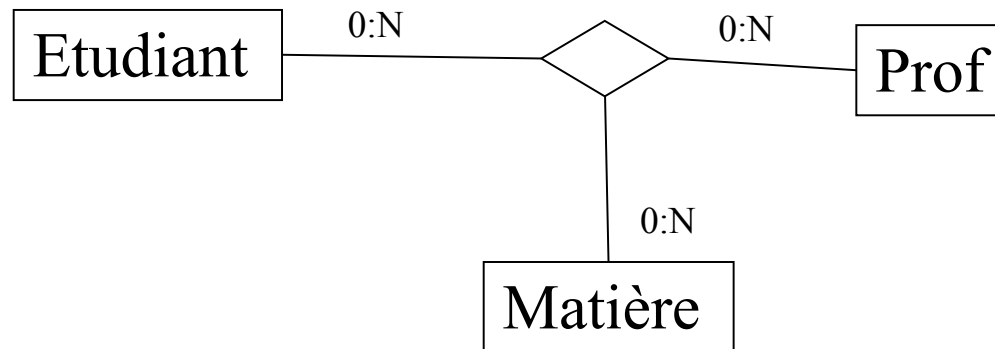


Association réflexive

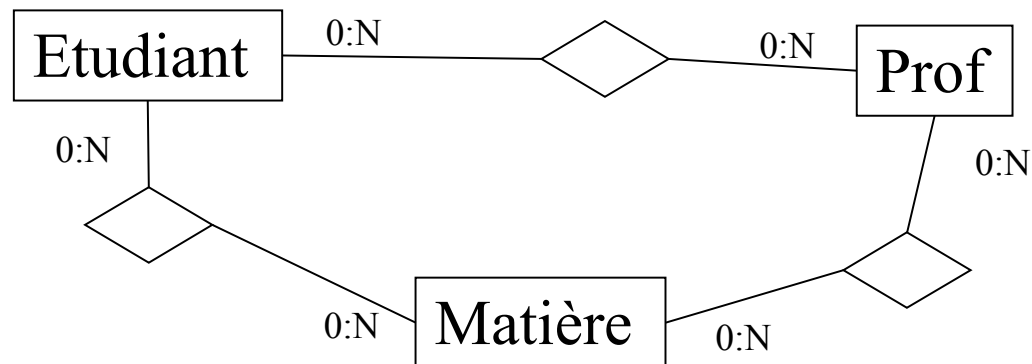
- Association réflexive : une entité d'une classe C est associé à une ou plusieurs entités de la *même* classe C.



Association ternaire



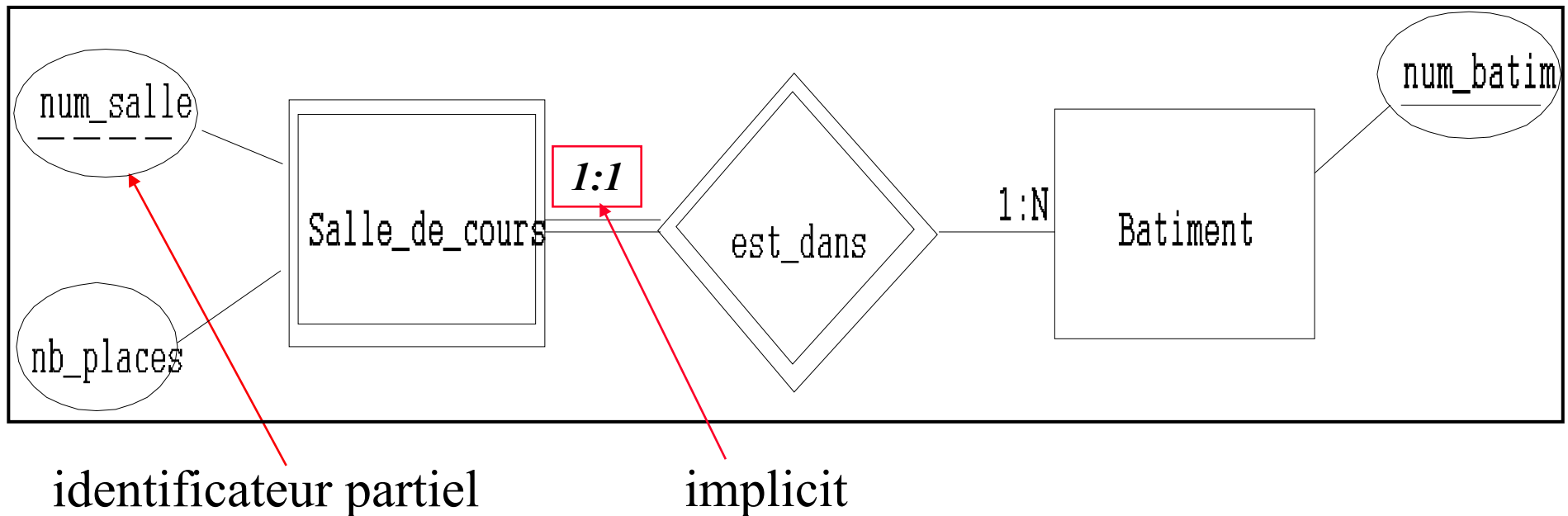
est différent de



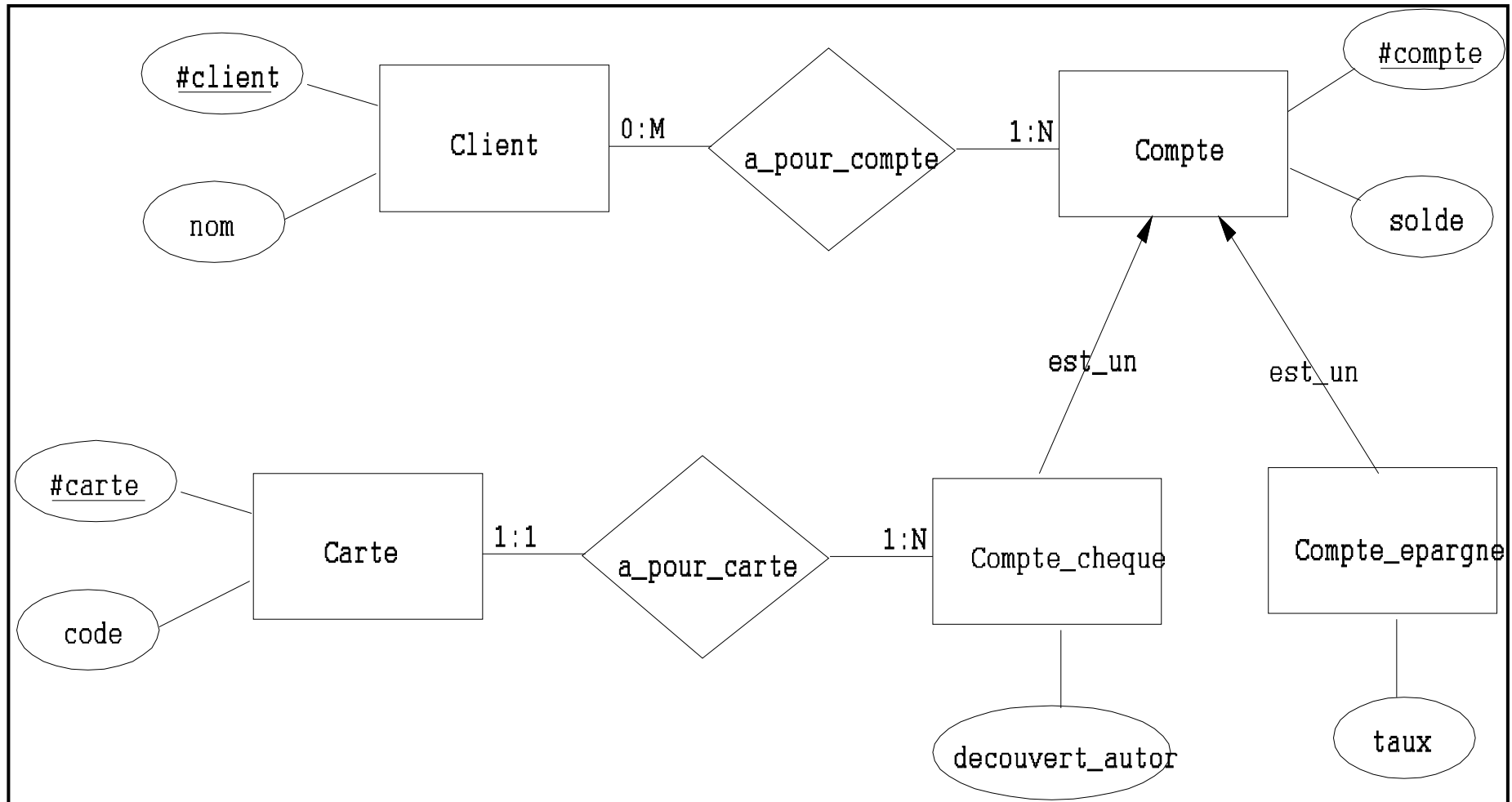
Pourquoi ?

Entités fortes et faibles

- **Entité forte**: entièrement identifiable par ses attributs
- **Entité faible**: ne peut être identifiée que par rapport à une autre entité, dite dominante, à laquelle elle se réfère. Son identificateur est :
 - identificateur partiel + identificateur de l'entité dominante.



Spécialisation



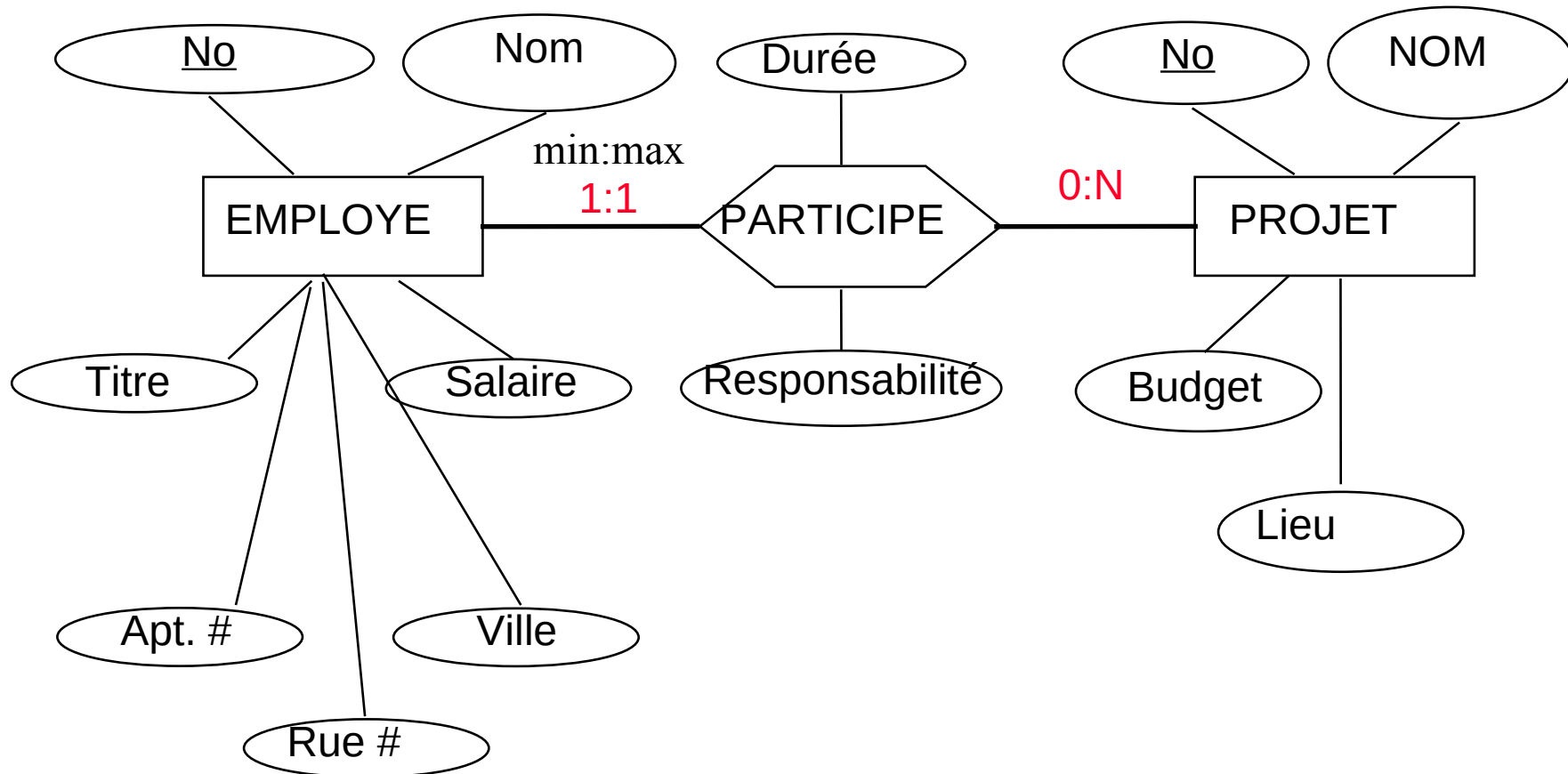
Introduction aux Bases de Données

- Fichiers versus Bases de Données
- Système de Gestion de Bases de Données (SGBD)
- Langages et modèles de données
- Modèle Entité-Association
- Modèle Relationnel

Modèle Relationnel

- Un modèle de données **logique** permet de décrire
 - la structure des données : *schéma*
 - les données : *instances*
 - les *opérations* sur le schéma et les données
-
- Exemple : Modèle relationnel
 - *schéma* = ensemble de *noms de tables* ou relations avec des attributs
 - *instance* = ensemble de *n-uplets* (tuples, lignes) stockés dans les tables
 - *opération* = expression SQL

Schéma entité-association



Une base de données relationnelle

schéma	EMP			PARTICIPE			
	ENO	ENOM	EMPLOI	ENO	PNO	RESP	DUR
instance	E1	J. Doe	Elect. Eng.	E1	P1	Manager	12
	E2	M. Smith	Syst. Anal.	E2	P1	Analyst	24
	E3	A. Lee	Mech. Eng.	E2	P2	Analyst	6
	E4	J. Miller	Programmer	E3	P3	Consultant	10
	E5	B. Casey	Syst. Anal.	E3	P4	Engineer	48
	E6	L. Chu	Elect. Eng.	E4	P2	Programmer	18
	E7	R. Davis	Mech. Eng.	E5	P2	Manager	24
	E8	J. Jones	Syst. Anal.	E6	P4	Manager	48
PROJ							
n-uplet	PNO	PNOM	BUDGET				
	P1	Instrumentation	150000				
	P2	Database Develop.	135000				
	P3	CAD/CAM	250000				
	P4	Maintenance	310000				
	P5	CAD/CAM	500000				

Schéma relationnel

- Un **schéma de relation** est composé d'un *nom de relation* R et d'un ensemble fini d'*attributs* $A = \{A_1, A_2, \dots, A_n\}$ définis sur des *domaines* de valeurs $D = \{D_i, D_j, \dots\}$
 - Notation : $R(A_1: D_i, A_2: D_j, \dots, A_n: D_k)$
 - Ex. : $EMP(ENO:integer, ENOM:string, EMPLOI:string)$
 - Autre notation (sans domaines): $R(A_1, A_2, \dots, A_n)$
 - Ex. : $EMP(ENO, ENOM, EMPLOI)$
- Un **schéma relationnel** S est un ensemble de schémas de relations.
 - Notation : $S = \{R_1, R_2, \dots, R_m\}$

Base de données relationnelle

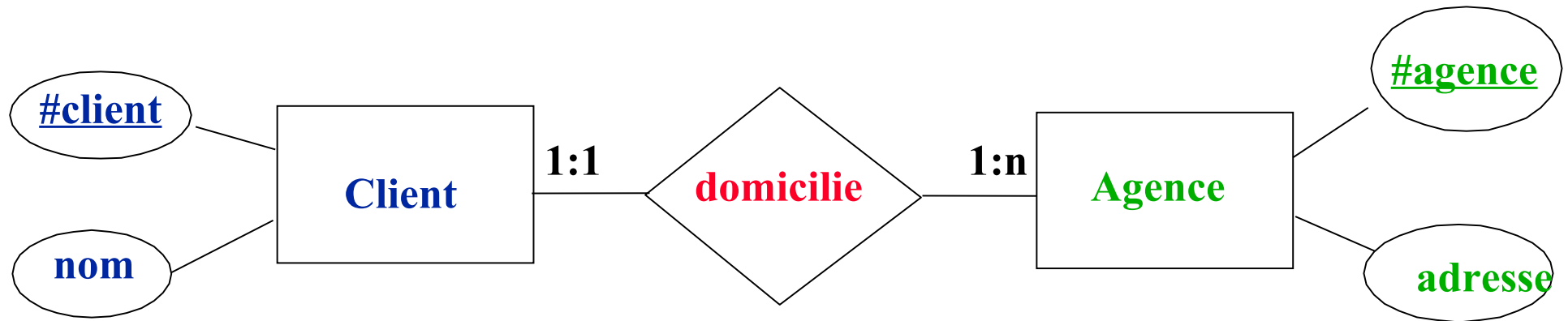
- Un **n-uplet** est une séquence de valeurs $\langle d_1, d_2, \dots, d_n \rangle$.
- Une **relation** (une **table**) **r** est un *ensemble fini* de *n-uplets*.
- Une **base de données relationnelle** est définie par un schéma relationnel $S = \{ R_1, R_2, \dots, R_n \}$ et un ensemble d'instances (tables) de $R_1, R_2, \dots, R_n$.
- Généralement on considère que pour chaque schéma de relation R_i , il existe exactement une instance qu'on note également R_i .

Conception de schémas relationnels

- Une schéma relationnel peut contenir des centaines de tables avec des milliers d'attributs.
- Problème: comment éviter des erreurs de conception?
- Deux solutions (complémentaires):
 - Génération (automatique) à partir d'un schéma E/A
 - Théorie des dépendances et normalisation (on verra plus tard)

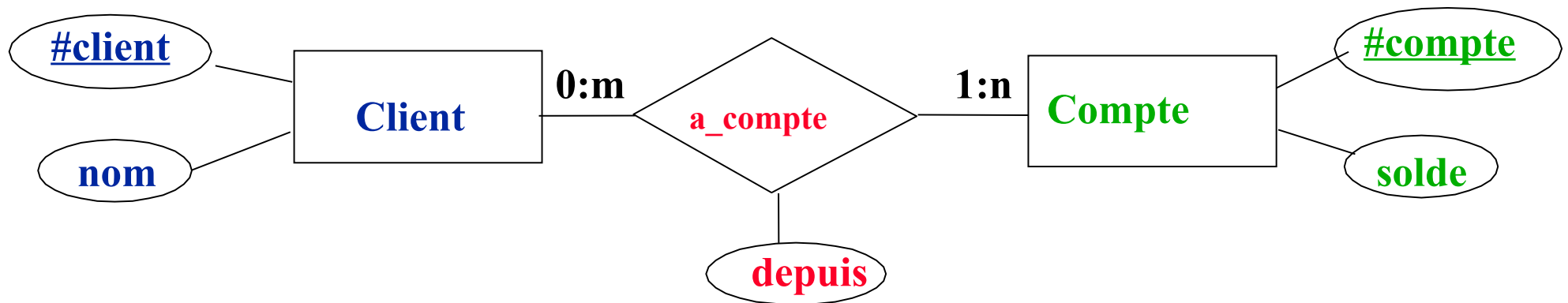
Règles de passage d'E/A vers un schéma relationnel

- Pour chaque type d'entité C avec les attributs $A_1, A_2, \dots, A_n$
 - créer un schéma de relation $R_C(A_1, A_2, \dots, A_n)$
 - clé de R_C = attributs clés de C
- Pour chaque type d'association A avec au moins un 1:1:
 - rajouter dans le schéma de relation R_C du type d'entité C *du côté du 1:1*, les clés des autres types d'entité associés ainsi que les attributs de l'association.
 - la clé de R_C ne change pas (sauf pour une entité faible).
- Pour les autres associations A (1:N, N:M, 0:1) :
 - créer un schéma de relation R_A avec les clés des types d'entité reliés et les attributs de l'association.
 - clé de R_A = union des clés des types d'entités associés



Client(#client, nom, #agence)

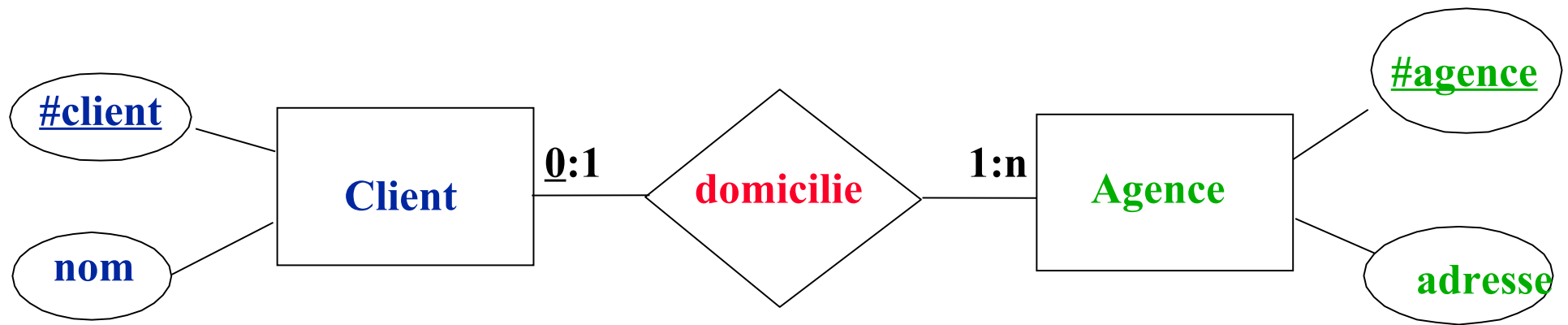
Agence(#agence, adresse)



Client(#client, nom)

Compte(#compte, solde)

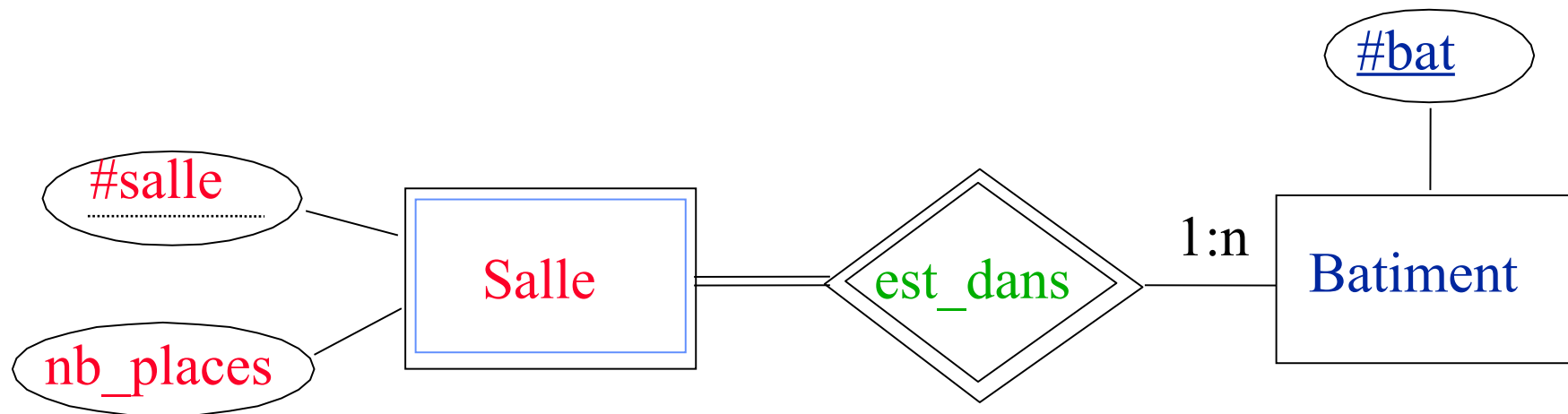
Acompte(#client, #compte, depuis)



Client(#client, nom)

Agence(#agence, adresse)

Domicile(#client, #agence(

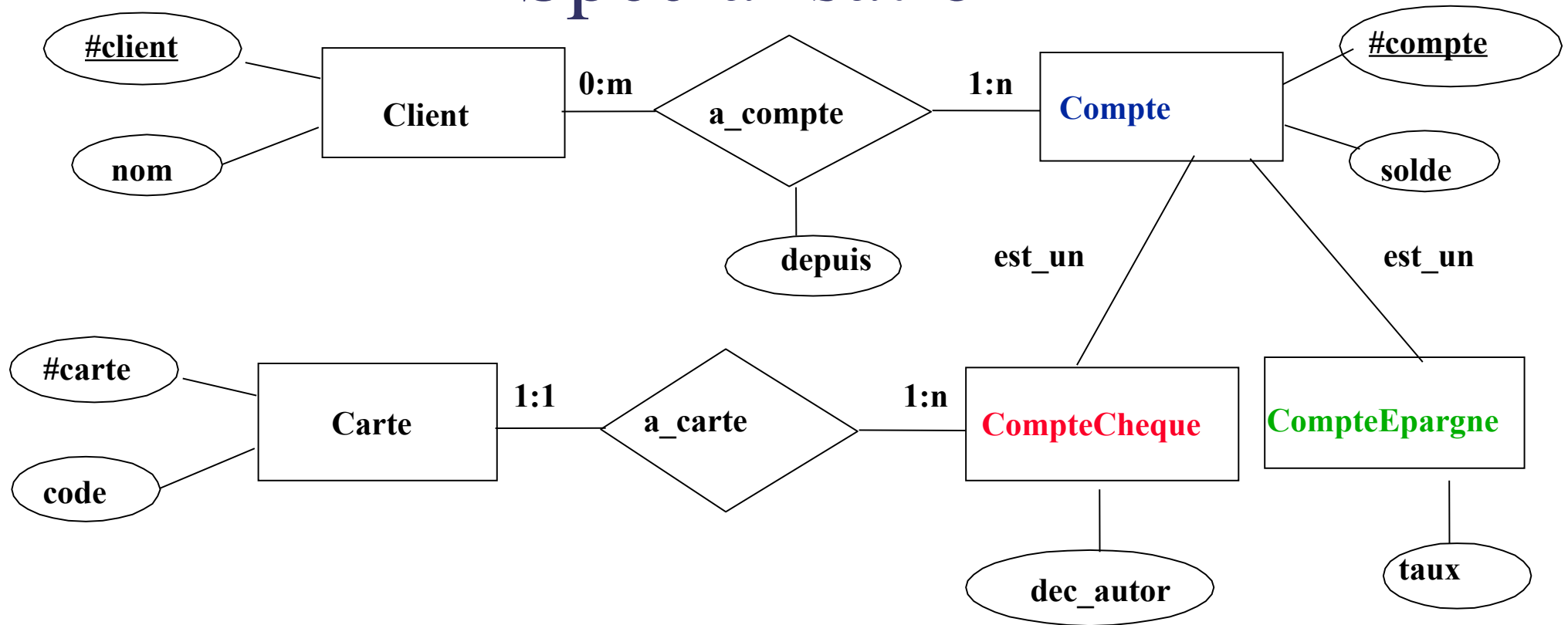


La clé de l'entité dominante fait partie de la clé de l'entité faible

Salle(#salle, #bat, nbplaces)

Batiment(#bat(... ,

Spécialisation



Compte(#compte, solde) CompteCh(#compte, dec-aut) CompteEp(#compte, taux)

Si Compte est un type d'entité « abstrait » (sans instances) :

CompteCh(solde, dec-aut, #compte) CompteEp(solde, taux, #compte)

Bases de données relationnelles

- Modèle (de données) *relationnel*
 - organise les données dans des **tables** ou **relations**
-
- Avantages :
 - « Relation » est un concept simple avec des **fondements mathématiques solides** :
 - théorie des ensembles
 - logique du premier ordre
 - On peut définir des **langages de requêtes simples, puissants et efficaces**

BD relationnelle : schéma et données

schéma

EMP		
<u>ENO</u>	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

données

WORKS			
<u>ENO</u>	<u>PNO</u>	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ		
<u>PNO</u>	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

PAY	
<u>TITLE</u>	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

Schéma relationnel

- EMP(ENO, ENAME, TITLE)
- PROJ (PNO, PNAME, BUDGET)
- WORKS(ENO,PNO, RESP, DUR)
- PAY(TITLE, SALARY)
-
- Les attributs soulignés sont les *clés* (identifiants de n-uplets)

EMP

<u>ENO</u>	ENAME	TITLE
------------	-------	-------

PROJ

<u>PNO</u>	PNAME	BUDGET
------------	-------	--------

WORKS

<u>ENO</u>	<u>PNO</u>	RESP	DUR
------------	------------	------	-----

PAY

<u>TITLE</u>	SALARY
--------------	--------

Schéma relationnel

Un **schéma de relation** est composé d'un *nom de relation* R et d'un ensemble fini d'*attributs* $A = \{A_1, A_2, \dots, A_n\}$ définis sur des *domaines* de valeurs $D = \{D_i, D_j, \dots\}$

Notation : $R(A_1: D_i, A_2: D_j, \dots, A_n: D_k)$

EMP(ENO:integer, ENOM:string, EMPLOI:string)

Autre notation sans domaines : $R(A_1, A_2, \dots, A_n)$

EMP(ENO, ENOM, EMPLOI)

Un **schéma relationnel** S est un ensemble de schémas de relations : $S = \{R_1, R_2, \dots, R_m\}$

Degré ou **arité** d'une relation : nombre d'**attributs**

Cardinalité d'une relation : nombre de **n-uplets**

Propriétés du modèle relationnel

- Fondement théorique : **théorie des ensembles**
 - **pas d'ordre entre les n-uplets !**
 - SQL : le *résultat* d'une requête peut être trié à la fin
 - pas d'ordre entre les attributs (distinction par le nom)
 - pas de doublons (deux ou plus de n-uplets identiques)
 - SQL : le résultat peut contenir des doublons (efficacité)
- Les valeurs d'attributs sont **atomiques** :
 - entiers, chaînes de caractères, date, ...
 - pas de constructeurs de types (struct, listes, set ...)
 - SQL 3 : modèle relationnel-*objet* avec des constructeurs de types

SQL : Introduction

Bases de données relationnelle

Commandes de définition de données (DDL)

Requêtes d'interrogation simples

Commandes de mise-à-jour de données (DML)

SQL

- SQL permet d'*interroger* et de *créer*, de *modifier* et de *supprimer* des bases de données.

•

SQL query : interrogation de données

pas de modification dans la BD

SQL-DDL : création, modification et suppression de
schémas

création, modification et suppression de schémas de relation

définition de clés et d'autres contraintes

SQL-DML : création, modification et suppression de
données

insertion, modification et suppression de n-uplets

Commandes DDL

- Création de schémas :

- **CREATE SCHEMA** *nom_schema* **AUTHORIZATION**
nom_utilisateur

-

- Création de tables :

- **CREATE TABLE** *nom_table*
 - (Attribute_1 <Type> [**DEFAULT** <value>],
 - Attribute_2 <Type> [**DEFAULT** <value>],
 - ...
 - Attribute_n <Type> [**DEFAULT** <value>]
 - [<Constraints>])

Exemple

- Définition de la relation **Project**
 -
 - **CREATE TABLE** Project (
 - Pno **CHAR**(3) ,
 - Pname **VARCHAR**(20) ,
 - Budget **NUMBER**(10, 2) **DEFAULT** 0.00 ,
 - City **CHAR**(9)) ;

Types de données ANSI SQL/

Numériques :

INT, SMALLINT. Possibilité de DEFAULT AUTOINCREMENT

FLOAT, DOUBLE (PRECISION), REAL

NUMBER(i,j) : i=précision (nombres de chiffres); j=échelle

NUMBER(4,3) = [0.000 ... 9.999]

FLOAT(p) : précision p (# de bits)

Chaînes de caractères :

CHAR(n) : longueur fixe = n

CHAR VARYING(n) : longueur variable avec max=n

CLOB : character large object (4 GO)

NCHAR, NCLOB : national character set (UTF8, UTF16, ...)

Séquences binaires :

RAW(n) : n <= 2Kb

LONG RAW(n) : n <= 2Mb

BLOB : binary large object (4 GO)

Types de données SQL / Oracle

Dates :

DATE : YYYY-MM-DD HH

Heure :

TIME : HH:MM:SS

Instants de temps :

- TIMESTAMP : 1999-04-15 8:00:00 US/Pacific

les champs DATE et TIME avec une précision de 10^{-6} secondes

Autres types *Oracle* : XML, spatial, media (audio/image/video)

Standard ANSI SQL et Oracle

- ANSI SQL

- CHAR(n)
- CHAR VARYING(n)
- NATIONAL CHAR(n)
- NUMERIC[(p,s)], DECIMAL[(p,s)]
- INT, SMALLINT
- FLOAT, DOUBLE PRECISION
- REAL

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

CHAR(n)
VARCHAR2(n)
NCHAR(n)
NUMBER(p,s)
NUMBER(38)
FLOAT(126)
FLOAT(63)

Type DATE

Format par défaut : YYYY-MM-DD

Fonctions :

`SYSDATE` : date / heure actuelle

`TO_DATE('98-DEC-25:17:30','YY-MON-DD:HH24:MI')`

-
- **SELECT * FROM** my_table
- **WHERE** datecol = `TO_DATE('04-OCT-2010','DD-MON-YYYY')`;

Arithmétique :

Date +/- N jours

`SYSDATE + 1` = demain

Types utilisateurs : domaines

- Pour créer un DOMAINE :

- **CREATE DOMAIN** *nom_domaine* **AS** *type*

-

- Exemple :

- **CREATE DOMAIN** Gender **AS** CHAR(1)

-

- Utile pour pouvoir contrôler l'évolution de définitions de types.

-

- : CREATE DOMAIN n'est **pas** implanté.

ORACLE®

Contraintes d'intégrité

- Une *contrainte d'intégrité* est une *condition (logique)* qui doit être satisfaite par les données stockées dans la BD.

•

- **But** : maintenir la cohérence/l'intégrité de la BD :
 - **Vérifier/valider *automatiquement*** (en dehors de l'application) les données lors des mises-à-jour (insertion, modification, effacement)
 - **Déclencher *automatiquement* des mises-à-jour** entre tables pour maintenir la cohérence globale.

Exemple

EMP

<u>ENO</u>	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

WORKS

<u>ENO</u>	<u>PNO</u>	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	NULL	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

clé

référence

PROJ

<u>PNO</u>	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	NULL
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

PAY

<u>TITLE</u>	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

Contraintes d'attributs

- **PRIMARY KEY (<attributs>)**
 - désigne un *ensemble d'attributs* comme la *clé primaire* de la table
- **FOREIGN KEY (attributs) REFERENCES <table>**
 - désigne un *ensemble d'attributs* comme la *clé étrangère* dans une contrainte référentielle (plus tard)
- **<attribut> NOT NULL**
 - spécifie qu'un attribut peut ne pas être renseigné
- **UNIQUE (<attributs>)**
 - spécifie un *ensemble d'attributs* dont les valeurs doivent être distinctes pour chaque couple de n-uplets (clé).

Exemple : Clés

- Créer la table Project(Pno, Pname, Budget, City) :
-
- **CREATE TABLE** Project
- (Pno **CHAR**(3),
- Pname **VARCHAR**(20) **UNIQUE NOT**
- **NULL**,
- Budget **DECIMAL**(10,2) **DEFAULT**
- 0.00,
- City **CHAR**(9),
- **PRIMARY KEY** (Pno) ;
-
- **Remarque**: Les attributs de la clé primaire sont toujours **UNIQUE** et **NOT NULL**.

ou Pno **CHAR**(3) **PRIMARY KEY**

Exemple : Clés et clés étrangères

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

- Définition des références inter-tables
 - **CREATE TABLE** Works
 - (Eno **CHAR**(3) ,
 - Pno **CHAR**(3) ,
 - Resp **CHAR**(15) ,
 - Dur **INT** ,
 - **PRIMARY KEY** (Eno, Pno) ,
 - **FOREIGN KEY** (Eno) **REFERENCES** Emp (Eno) ,
 - **FOREIGN KEY** (Pno) **REFERENCES** Project (Pno)) ;

Autres commandes DDL

- **DROP SCHEMA** *nom_schema* [,...] [**CASCADE** | **RESTRICT**]

- supprime les schémas indiqués
- CASCADE : toutes les tables du schéma
- RESTRICT : seulement les tables vides (par défaut)

- **DROP TABLE** *nom_table* [,...] [**CASCADE** | **RESTRICT**]

- RESTRICT : supprime la table seulement si elle n'est référencée par aucune contrainte (clé étrangère) ou vue (par défaut)
- CASCADE : supprime aussi toutes les tables qui « dépendent » de *nom_table*

- **ALTER TABLE** *nom_table* **OPERATION**

- *modifie* la définition de la table
- opérations:
 - Ajouter (ADD), effacer (DROP), changer (MODIFY) attributs et contraintes
 - changer propriétaire, ...

SQL : Introduction

Bases de données relationnelle

Commandes de définition de données (DDL)

Requêtes d'interrogation simples

Commandes de mise-à-jour de données (DML)

Requêtes d'interrogation SQL

- Structure de base d'une requête SQL simples :

• SELECT $var_i.A_{ik}, \dots$	attributs
• FROM $R_{i1} var_1, R_{i2} var_2 \dots$	tables
• WHERE P	prédicat/condition

- où:

- var_i désigne la table R_{ij}

- Les variables dans la clause SELECT et dans la clause WHERE doivent être *liées* dans la clause FROM.

-

- **Simplifications :**

- Si var_j n'est pas spécifiée, alors la variable s'appelle par défaut R_{ij} .

- Si **une seule table/variable** var possède l'attribut A , on peut écrire plus simplement A au lieu de $var.A$.

Requêtes simples

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

- Noms de tous les employés ?

- ```
SELECT t.Ename FROM Emp t
```
- ```
SELECT Ename FROM Emp
```

- Noms des projets avec leurs budgets ?

- ```
SELECT Pname, Budget FROM Project
```

- Villes où un projet existe ?

- ```
SELECT DISTINCT City FROM Project
```

DISTINCT enlève les doublons !

- Tous les employés (toutes les informations) ?

- ```
SELECT * FROM Emp
```

# Example

PROJ

| <u>PNO</u> | PNAME             | BUDGET |
|------------|-------------------|--------|
| P1         | Instrumentation   | 150000 |
| P2         | Database Develop. | 135000 |
| P3         | <i>CAD/CAM</i>    | 250000 |
| P4         | Maintenance       | 310000 |
| P5         | <i>CAD/CAM</i>    | 500000 |

SELECT PNO, BUDGET FROM PROJ :

| <u>PNO</u> | BUDGET |
|------------|--------|
| P1         | 150000 |
| P2         | 135000 |
| P3         | 250000 |
| P4         | 310000 |
| P5         | 500000 |

SELECT PNAME FROM PROJ :

| PNAME             |
|-------------------|
| Database Develop. |
| Instrumentation   |
| <i>CAD/CAM</i>    |
| Maintenance       |
| <i>CAD/CAM</i>    |

SELECT DISTINCT PNAME FROM PROJ :

| PNAME             |
|-------------------|
| Maintenance       |
| <i>CAD/CAM</i>    |
| Database Develop. |
| Instrumentation   |

# Requêtes avec prédicats

**Emp** (Eno, Ename, Title, City)    **Project**(Pno, Pname, Budget, City)  
**Pay**(Title, Salary)                    **Works**(Eno, Pno, Resp, Dur)

- Professions qui gagnent plus de 50 000 € par an ?
  - **SELECT** x.Title
  - **FROM** Pay x
  - **WHERE** x.Salary > 50000
  -
- Numéros des managers d'un projet qui dure plus de 17 mois?
  - **SELECT** Eno
  - **FROM** Works
  - **WHERE** Dur > 17 **AND** Resp='Manager'
  -

# Exemple de sélection

EMP

| ENO | ENAME     | TITLE       |
|-----|-----------|-------------|
| E1  | J. Doe    | Elect. Eng. |
| E2  | M. Smith  | Syst. Anal. |
| E3  | A. Lee    | Mech. Eng.  |
| E4  | J. Miller | Programmer  |
| E5  | B. Casey  | Syst. Anal. |
| E6  | L. Chu    | Elect. Eng. |
| E7  | R. Davis  | Mech. Eng.  |
| E8  | J. Jones  | Syst. Anal. |

```
SELECT * FROM EMP
WHERE TITLE = 'Elect. Eng.'
```

| ENO | ENAME  | TITLE       |
|-----|--------|-------------|
| E1  | J. Doe | Elect. Eng  |
| E6  | L. Chu | Elect. Eng. |

# Requêtes avec plusieurs relations : jointure

**Emp** (Eno, Ename, Title, City)   **Project**(Pno, Pname, Budget, City)  
**Pay**(Title, Salary)                      **Works**(Eno, Pno, Resp, Dur)

- Noms et titres des employés qui travaillent dans un projet pendant plus de 17 mois?

- **SELECT** Ename, Title
- **FROM**     Emp, Works
- **WHERE**    Dur > 17
- **AND** Emp.Eno = Works.Eno
- 

- Noms et titres des employés qui travaillent dans un projet à Paris ?

- **SELECT** Ename, Title
- **FROM**     Emp E, Works W, Project P
- **WHERE**    P.City = 'Paris'
- **AND**     E.Eno = W.Eno **AND** W.Pno = P.Pno

# Example

EMP

| <u>ENO</u> | ENAME     | TITLE       |
|------------|-----------|-------------|
| E1         | J. Doe    | Elect. Eng  |
| E2         | M. Smith  | Syst. Anal. |
| E3         | A. Lee    | Mech. Eng.  |
| E4         | J. Miller | Programmer  |
| E5         | B. Casey  | Syst. Anal. |
| E6         | L. Chu    | Elect. Eng. |
| E7         | R. Davis  | Mech. Eng.  |
| E8         | J. Jones  | Syst. Anal. |

```
SELECT ENO, ENAME, EMP.TITE,
SALARY
FROM EMP, PAY
WHERE EMP.TITLE=PAY.TITLE
```

| ENO | ENAME     | EMP.TITLE   | SALARY |
|-----|-----------|-------------|--------|
| E1  | J. Doe    | Elect. Eng. | 55000  |
| E2  | M. Smith  | Analyst     | 70000  |
| E3  | A. Lee    | Mech. Eng.  | 45000  |
| E4  | J. Miller | Programmer  | 60000  |
| E5  | B. Casey  | Syst. Anal. | 70000  |
| E6  | L. Chu    | Elect. Eng. | 55000  |
| E7  | R. Davis  | Mech. Eng.  | 45000  |
| E8  | J. Jones  | Syst. Anal. | 70000  |

PAY

| <u>TITLE</u> | SALARY |
|--------------|--------|
| Elect. Eng.  | 55000  |
| Syst. Anal.  | 70000  |
| Mech. Eng.   | 45000  |
| Programmer   | 60000  |

# Tri du résultat : ORDER BY

**Emp** (Eno, Ename, Title, City)   **Project**(Pno, Pname, Budget, City)  
**Pay**(Title, Salary)                      **Works**(Eno, Pno, Resp, Dur)

- Noms, budgets et villes des projets de budget supérieur à 250 000 euros, en ordonnant le résultat par ordre décroissant de budget puis par nom par ordre alphanumérique croissant ?

- - **SELECT**           Pname, Budget, City
  - **FROM**   Project
  - **WHERE** Budget > 250000
  - **ORDER BY** Budget DESC, Pname

- Par défaut, l'ordre est ascendant (ASC). L'ordre descendant peut être spécifié par le mot-clé DESC

# Requêtes et valeurs NULL

- Les valeurs d'attributs peuvent être inconnues : NULL  
une *opération* avec un attribut de valeur NULL retourne NULL  
une *comparaison* avec un attribut de valeur NULL retourne UNKNOWN

UNKNOWN introduit une logique à trois valeurs :

Vrai = 1, UNKNOWN = 0.5, Faux = 0

$x \text{ AND } y = \min(x, y)$ ,  $x \text{ OR } y = \max(x, y)$ ,  $\text{not}(x) = 1 - x$

**Attention :** NULL n'est pas une constante :

« NAME = NULL » ou « NULL + 30 » sont incorrects.

Pour vérifier si la valeur d'un attribut est inconnue, on utilise IS NULL :

```
SELECT Pname
FROM Proj WHERE City IS NULL
```



# SQL : Introduction

Bases de données relationnelle

Commandes de définition de données (DDL)

Requêtes d'interrogation simples

Commandes de mise-à-jour de données (DML)

# Insertion de tuples

- **INSERT INTO** *table* [ ( *column* [, ...] ) ]
- { **VALUES** ( { *expression* | **DEFAULT** } [, ...] ) | *query* }
- 

• en spécifiant des valeurs différentes pour tous les attributs *dans l'ordre* utilisé dans CREATE TABLE :

- **INSERT INTO R VALUES** (value( $A_1$ ), ..., value( $A_n$ ))

• en spécifiant les noms d'attributs (indépendant de l'ordre) :

- **INSERT INTO R** (  $A_i$ , ...,  $A_k$  ) **VALUES** (value( $A_i$ ), ..., value( $A_k$ ))

• insertion du résultat d'une requête (copie) :

- **INSERT INTO R** <*requête\_SQL*>

# Exemples d'insertion

**Emp** (Eno, Ename, Title, City)    **Project**(Pno, Pname, Budget, City)  
**Pay**(Title, Salary)                      **Works**(Eno, Pno, Resp, Dur)

- Insertion d'un nouvel employé :

- 
- **INSERT INTO**                      Emp (Eno, Ename, Title, City)
- **VALUES**
- ('E24', 'Martin', 'Programmeur', 'Paris')

- 
- Insertion dans Pay des n-uplets dont les titres existent dans Emp;  
avec le salaire à 0 :

- 
- **INSERT INTO** Pay
- **SELECT DISTINCT** Title, 0 **FROM** Emp

# Suppression de tuples

- **DELETE FROM** *table* [ **WHERE** *condition* ]

- Supprimer tous les employés qui ont travaillé dans le projet P3 pendant moins de 3 mois :

- 

- 

- **DELETE FROM** Emp

- 

- **WHERE** Eno **IN** ( **SELECT** Eno

- 

- **FROM** Works

- 

- **WHERE** PNO='P3' **AND** Dur < 3)

- 

- **Attention** : il faut aussi effacer les n-uplets correspondants dans la table Works (cohérence des données).

# Modification de tuples

- **UPDATE** *table*
- **SET** *column* = { *expression* | **DEFAULT** } [, ...]
- [ **WHERE** *condition* ]

- 

- - **UPDATE**  $R$
  - **SET**  $A_i = \text{value}, \dots, A_k = \text{value}$
  - **WHERE**  $P$

# Exemples de modification

**Emp** (Eno, Ename, Title, City)      **Project**(Pno, Pname, Budget, City)  
**Pay**(Title, Salary)                      **Works**(Eno, Pno, Resp, Dur)

- Plafonner les salaires à 50 000 euros :

- 
- **UPDATE** Pay **SET** Salary = 50000
- **WHERE** Salary > 50000;
- 

- Augmenter de 5% les budgets des projets situés à Nantes :

- 
- **UPDATE** Project
- **SET** Budget = Budget\*1.05
- **WHERE** City = 'Nantes'

# SQL : interrogation de BD

Requêtes simples

Requêtes complexes

SQL et le calcul relationnel

Agrégats et groupement

Couplage SQL et langage de programmation

# SQL : Rappel

- Structure de base d'une requête SQL simples :

**SELECT**  $\text{var}_i.A_{ik}, \dots$  *attributs*  
**FROM**  $R_{i1} \text{var}_1, R_{i2} \text{var}_2 \dots$  *variables (n-uplet)*  
**WHERE**  $P$  *prédicat/condition*

- où:

La variable n-uplet  $\text{var}_i$  « appartient » à la table  $R_{ij}$  :  $R_{i1}(\text{var}_1)$

Les variables dans la projection (clause SELECT) et dans la condition (clause WHERE) doivent être liées dans la clause FROM.

- **Simplifications :**

Si  $\text{var}_j$  n'est pas spécifiée, alors la variable s'appelle par défaut  $R_{ij}$ .

Si **une seule variable n-uplet** possède l'attribut A, on peut écrire plus simplement A (non-ambiguïté).



# SQL : Requêtes imbriquées

- Requête **imbriquée** dans la clause WHERE d'une requête **externe**:
  - **SELECT** ...
  - **FROM** ...
  - **WHERE** [Opérande] **Opérateur** (**SELECT** ...  
**FROM** ...  
**WHERE** ...)
- Opérateurs ensemblistes :
  - **(A<sub>1</sub> , ...A<sub>n</sub>) IN <sous-req>** : appartenance ensembliste
  - **EXISTS <sous-req>** : test d'existence
  - **(A<sub>1</sub> , ...A<sub>n</sub>) <comp> [ALL | ANY] <sous-req>** : comparaison avec quantificateur (ANY par défaut)

# Expression « IN »

```
SELECT ...
FROM ...
WHERE (A1, ..., An) [NOT] IN (SELECT B1, ..., Bn
 FROM ...
 WHERE ...)
```

- **Sémantique** : la condition est vraie si le n-uplet désigné par  $(A_1, \dots, A_n)$  de la requête *externe* appartient (n'appartient pas) au résultat de la requête *interne*.

# ALL/ANY

```
SELECT ...
FROM ...
WHERE (A1, ..., An) ! ALL/ANY (SELECT B1, ..., Bn
FROM ...
WHERE ...)
```

- 

On peut utiliser une comparaison ! ! {=, <, <=, >, >=, <>} et ALL (! ) ou ANY (! ): La condition est alors vraie si la comparaison est vraie pour tous les n-uplets /au moins un n-uplet de la requête interne.

- 

- Comment peut-on exprimer « IN » avec ALL/ANY?

# Exemple avec “IN”

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

- Noms des employés qui habitent dans des villes où il y a des [n’y a pas de] projets de budget inférieur à 50?
- 
- **SELECT** Ename
- **FROM** Emp
- **WHERE** City [**NOT**] **IN** (**SELECT** City
- **FROM** Project
- **WHERE** Budget < 50)

# Expression “EXISTS”

Q

```
SELECT ...
FROM ...
WHERE
```

(NOT) EXISTS (SELECT \*  
FROM ...  
WHERE P)

Q'

•

*Sémantique procédurale* : pour chaque n-uplet  $x$  de la requête externe  $Q$ , exécuter la requête interne  $Q'$ ; s'il existe au moins un n-uplet  $y$  dans le résultat de la requête interne, alors sélectionner  $x$ .

*Sémantique logique* :  $\{ x... \mid Q(x) \wedge (\exists y) Q'(y) \}$

Les deux requêtes sont généralement corrélées :  $P$  dans la requête interne  $Q'$  exprime une jointure entre les tables de  $Q'$  et les tables de la requête externe  $Q$ .

# Exemple avec “EXISTS”

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

- Noms des employés qui travaillent dans une ville où il y a un projet?

- 
- **SELECT** Ename **FROM** Emp
- **WHERE EXISTS** ( **SELECT** \*
- **FROM** Project
- **WHERE** Emp.City=Project.City)

- Noms des employés qui travaillent dans une ville sans projet?

- 
- **SELECT** Ename **FROM** Emp
- **WHERE NOT EXISTS** ( **SELECT** \*
- **FROM** Project
- **WHERE** Emp.City=Project.City)

# Exemple

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

- Villes où il y a un employé ?

- 

- **SELECT** City **FROM** Emp

- 

- Villes où il y a un projet ?

- 

- **SELECT** City **FROM** Project

- 

- 

- 

- 

-

# SQL : Opérateurs ensemblistes

- Trois opérations ensemblistes :

|              |        |                  |
|--------------|--------|------------------|
| union        | $\cup$ | <b>UNION</b>     |
| difference   | -      | <b>EXCEPT</b>    |
| intersection | $\cap$ | <b>INTERSECT</b> |

- 

Par défaut, les opérations ensemblistes *éliminent les doublons* (*ensemble*). Pour garder les doublons (*multi-ensemble*), il faut ajouter ALL après l'opérateur : UNION ALL, EXCEPT ALL, INTERSECT ALL

Les types des deux tables doivent être “compatibles” : même nombre d'arguments du même type.

- 



EXCEPT est remplacé par MINUS

MINUS ALL et INTERSECT ALL ne sont pas implantés

-



# Requêtes avec opérateurs ensemblistes

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

- Villes où il y a soit un employé soit un projet ?

- 
- (SELECT City FROM Emp)
- UNION
- (SELECT City FROM Project)
- 

- Villes où il y a des employés mais pas de projets ?

- 
- (SELECT City FROM Emp)
- EXCEPT
- (SELECT City FROM Project)
- 

- Comment peut-on faire la même requête sans EXCEPT avec une sous-requête ?

# Requêtes avec opérateurs ensemblistes

**Emp** (Eno, Ename, Title, City)      **Project**(Pno, Pname, Budget, City)  
**Pay**(Title, Salary)                      **Works**(Eno, Pno, Resp, Dur)

- **Villes où il y a à la fois un employé et un projet?**
- (**SELECT** City **FROM** Emp)
- **INTERSECT**
- (**SELECT** City **FROM** Project)
- 
- 
- 
- 
- **Noms des projets et des employés de Paris?**
- (**SELECT** Ename AS Nom **FROM** Emp
- **WHERE** City = 'Paris')
- **UNION**
- (**SELECT** Pname AS Nom **FROM** Project
- **WHERE** City = 'Paris')

Comment peut-on  
faire la même requête sans  
INTERSECT avec une  
sous-requête ?

# SQL : interrogation de BD

Requêtes d'interrogation simples

Requêtes complexes

**Agrégats et groupement**

SQL et le calcul relationnel

Couplage SQL et langage de programmation

# SQL : Fonctions d'agrégation

- Pour calculer une valeur numérique à partir d'une relation
- fonctions d'agrégation appliquée à un attribut :
  - COUNT (nombre de valeurs),
  - SUM (somme des valeurs),
  - MAX (valeur maximale),
  - MIN (valeur minimale)
  - AVG (moyenne des valeurs)

```
SELECT AggFunc (A_i) , ... , AggFunc (A_j)
FROM R_1 , . . . , R_m
WHERE P
```

# Exemples d'agrégation

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

Budgets totaux des projets de Paris?

```
SELECT SUM(Budget)
FROM Project
WHERE City = 'Paris'
```

Nombre de villes où il y a un projet avec l'employé E4?

```
SELECT COUNT(DISTINCT City)
FROM Project, Works
WHERE Project.Pno = Works.Pno
AND Works.Eno = 'E4'
```

# Fonctions d'agrégation

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

Noms des projets dont le budget est supérieur au budget moyen?

```
SELECT Pname
FROM Project
WHERE Budget > ANY
 (SELECT AVG (Budget)
 FROM Project)
```

# Requêtes de groupement : GROUP BY

- Pour *partitionner* les n-uplets résultats en fonction des valeurs de certains attributs :

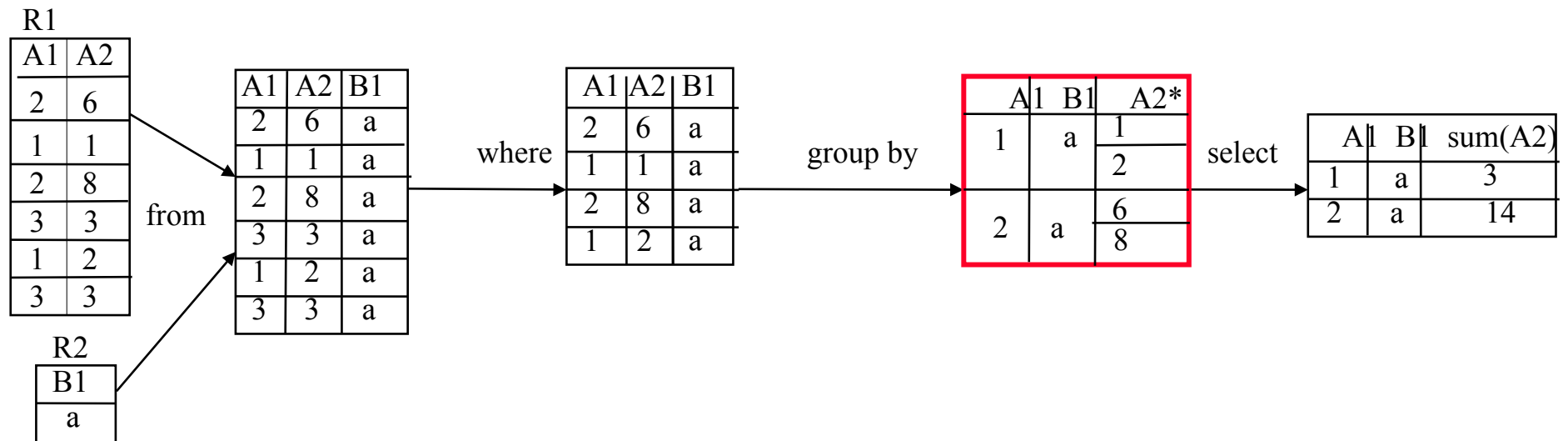
```
SELECT $A_i, \dots, A_n, aggr1, aggr2, \dots$
FROM $R_1, \dots, R_m$
WHERE P
GROUP BY $A_j \dots, A_k$
```

- Règle:

tous les attributs projetés ( $A_i, \dots, A_n$ ) dans la clause SELECT *ne sont pas impliqués dans une opération d'agrégation* et doivent être inclus dans l'ensemble des attributs ( $A_j \dots, A_k$ ) de la clause GROUP BY (qui peut avoir d'autres attributs en plus)

# GROUP BY

**SELECT**       $A1, B1, \text{sum}(A2)$   
**FROM**     $R_1, R_2$   
**WHERE**  $A1 < 3$   
**GROUP BY**  $A1, B1$





# Exemples de groupement

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

- Numéros des projets avec le nombre d'employés impliqués **par projet ?**
  - **SELECT** Pno, Count (Eno)
  - **FROM** Works
  - **GROUP BY** Pno
  -
- Noms des projets avec la durée moyenne et la durée maximale de participation d'un employé **par projet ?**
  - **SELECT** Pname, **AVG** (Dur) , **MAX** (Dur)
  - **FROM** Works, Project
  - **WHERE** Works.Pno=Project.Pno
  - **GROUP BY** Pno, Pname

# Predicats sur des groupes

- Pour *garder les groupes (partitions)* qui satisfont une certaine condition :

```
SELECT $A_i, \dots, A_n$
FROM $R_1, \dots, R_m$
WHERE P
GROUP BY $A_j \dots, A_k$
HAVING Q
```

- 
- Règle : La condition  $Q$  porte généralement sur des valeurs atomiques retournées par un opérateur d'agrégation sur les attributs qui n'apparaissent pas dans le GROUP BY

# Exemples de groupement

**Emp** (Eno, Ename, Title, City)  
**Pay**(Title, Salary)

**Project**(Pno, Pname, Budget, City)  
**Works**(Eno, Pno, Resp, Dur)

- Villes dans lesquelles habitent plus de 2 employés?
- **SELECT** City
- **FROM** Emp
- **GROUP BY** City
- **HAVING COUNT**(ENO) > 2
- 
- Projets dans lesquels plus de 2 employés partagent une responsabilité?
- **SELECT DISTINCT** Pno
- **FROM** Works
- **GROUP BY** Pno, Resp
- **HAVING COUNT**(**DISTINCT** ENO) > 2