

## Partie 2 : Modèle relationnel et SQL

M. T. Pham, A. Ridard, P. Berg

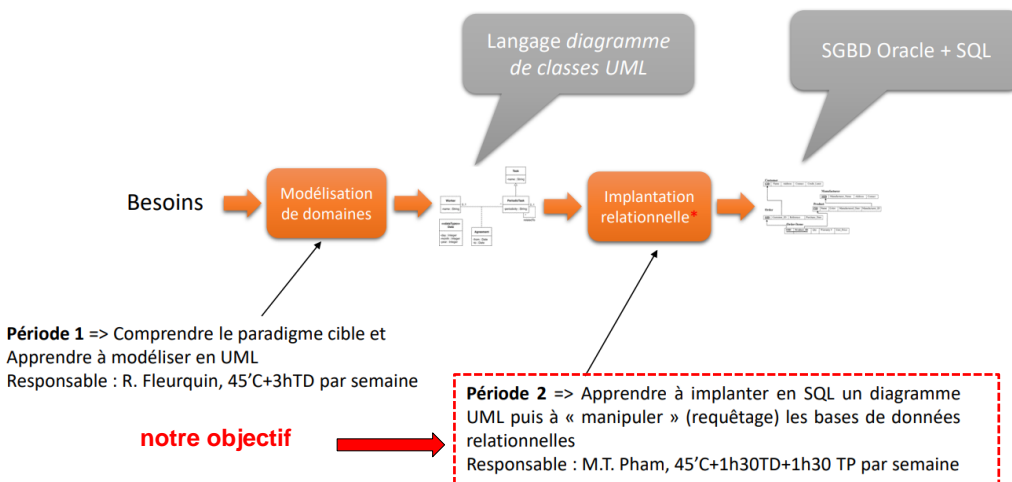
### Cours 1

## SGBD Relationnelle

## Traduction UML vers Schéma Relationnel

### Rappel sur la démarche de la R1.05

(cf. cours 0 R1.05 en période 1)



## Déroulement de R1.05-P2

- 7 semaines: S45 → S50 (2022) et S1(2023) avec 0.5C+1TD+1TP par semaine

### Planning

- S1: SGBD Relationnel et Traduction UML vers Schéma Relationnel *Cours 1*
  - S2: Création des tables
  - S3: Insertion, Mis à jour et Suppression
  - S4: Récapitulatifs
  - S5+S6+S7: Algèbre Relationnelle & Requêtes SQL *Cours 3*
- Cours 2* (S2, S3, S4)

### Evaluation:

- Présence obligatoire en Cours, TDs et TP
- Contrôle continu** : notes des rendus des TP + QCMs possibles
- Contrôle final** : écrit (Calculatrice + A4 recto-verso autorisé)

## Système de gestion de base de données (SGDB)

- **Base de données** : une collection partagée de données en relation logique et une description des données, conçues pour satisfaire les besoins d'information d'une organisation
- **SGDB** : le système logiciel qui permet à des utilisateurs de définir, créer, mettre à jours et supprimer une base de données et d'en contrôler l'accès
  - Langage de définition de données (LDD)
  - Langage de manipulation de données (LMD)
  - Langage de contrôle de données (LCD)

- Très grande quantité de données à gérer
- Besoin d'interroger, mettre à jour souvent de manière rapide et efficace des données
- Contrôle la redondance d'information
- Contrôle le partage de données
- Sécurité, administrer des données
- Offrir des interfaces d'accès multiples
- Assurer la reprise en cas de panne

## SGDB Relationnel

- **SGBDR** : permet aux utilisateurs de construire, mettre à jour, gérer et interagir avec une base de données relationnelle, permettant de stocker des données sous forme de tableau
- Premier système dans les années 1980 : *Oracle avec langage SQL*
- Actuellement : (<https://db-engines.com/en/ranking>)

397 systems in ranking, October 2022

Rank			DBMS	Database Model	Score		
Oct 2022	Sep 2022	Oct 2021			Oct 2022	Sep 2022	Oct 2021
1.	1.	1.	Oracle	Relational, Multi-model	1236.37	-1.88	-33.98
2.	2.	2.	MySQL	Relational, Multi-model	1205.38	-7.09	-14.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	924.68	-1.62	-45.93
4.	4.	4.	PostgreSQL	Relational, Multi-model	622.72	+2.26	+35.75
5.	5.	5.	MongoDB	Document, Multi-model	486.23	-3.40	-7.32
6.	6.	6.	Redis	Key-value, Multi-model	183.38	+1.91	+12.03
7.	7.	8.	Elasticsearch	Search engine, Multi-model	151.07	-0.37	-7.19
8.	8.	7.	IBM Db2	Relational, Multi-model	149.66	-1.73	-16.30
9.	11.	11.	Microsoft Access	Relational	138.17	-1.87	+21.79
10.	10.	9.	SQLite	Relational	137.80	-1.02	+8.43

## Règles d'écriture et de codage

- Le programmeur passant beaucoup moins de temps à écrire du code nouveau qu'à le relire et le modifier, il faut que **le code soit facile à lire**.
- **Les règles suivantes sont impératives (pour les TDs/TPs de R1.05 à IUT Vannes)**
  - ✓ Tables (relations) en **minuscules avec la 1<sup>ère</sup> lettre en majuscule**
  - ✓ Attributs (titres des colonnes) en **minuscules**
  - ✓ **(1)** pour la **clé primaire**
  - ✓ Pour la **clé étrangère** : différence entre la période 1 et la période 2

```

Departement ( nomD(1), nbGroupes )
Etudiant ( noEtu(1), nom, prenom, @leDept REF Departement(nomD) )

```



```

Departement ( nomD(1), nbGroupes)
Etudiant ( noEtu(1), nom, prenom, leDept = @Departement.nomD )

```

# Notion de Diagramme relationnel (non UML)

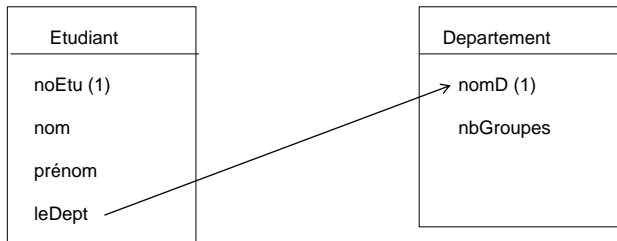
- C'est le dessin associé au '**schéma relationnel**'
- Il met en évidence les deux niveaux de relations du modèle :
  - relation-table
  - association entre tables par FK

Schéma relationnel :

Departement ( nomD(1), nbGroupes)

Etudiant ( noEtu(1), nom, prenom, leDept = @Departement.nomD )

Diagramme  
relationnel



## Traduction

## UML vers Schéma Relationnel

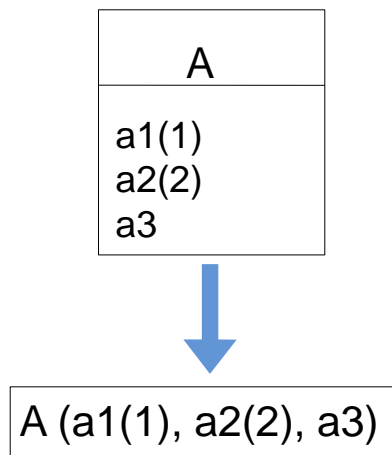
## UML vers SR

- On applique des « **règles de traduction** »
- Le **schéma relationnel** doit préciser les contraintes :
  - De clés (primaires, secondaires, étrangères)
  - D'existence (présence obligatoire) → NOT NULL (NN)
  - D'unicité (sauf en cas de multi-attributs) → UNIQUE (UQ)
- Comme pour le diagramme de classes UML, les contraintes qui ne sont pas exprimées dans le schéma relationnel doivent apparaître dans les « **contraintes textuelles** »

## Règle 1 : classes

- Une classe devient une relation (une table dans la BD)
- Ses attributs deviennent les attributs de la relation (les colonnes de la table dans la BD)
- Ses identifiants deviennent les clés de la relation
- Ses objets deviennent les t-uples de la relation (les lignes de la table dans la BD)

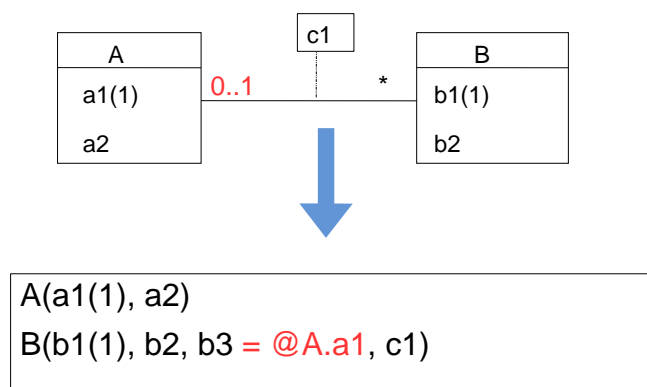
## Règle 1 : classes



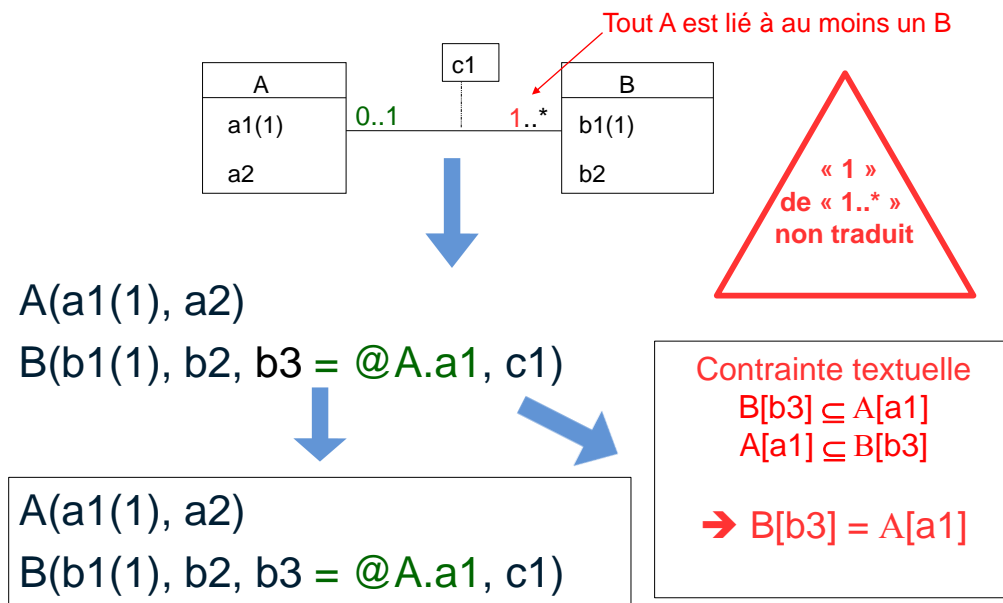
## Règle 2 : associations 1-\*

- Le « 1-\* » correspond aux multiplicités maximales apparaissant aux extrémités de l'association
- Les associations « \*-1 » se traitent de la même manière
- Ce type d'association devient **une clé étrangère** « à la source » c'est à dire **dans la relation du côté « \* »**
- Les éventuels **attributs portés par cette association** deviennent **des attributs de la relation contenant la clé étrangère**

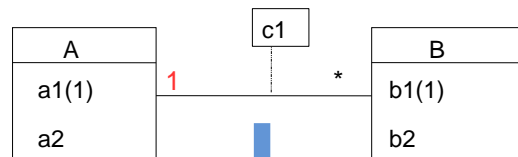
## Règle 2 : associations 1-\*



## Règle 2 : associations 1-\*



## Règle 2 : associations 1-\*



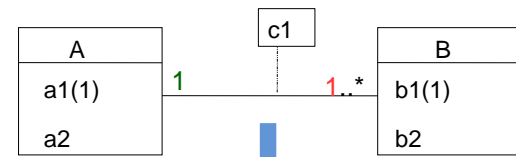
A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1, c1)

0 interdit  
non traduit

A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1 (NN), c1)

17

## Règle 2 : associations 1-\*



A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1 (NN), c1)

« 1 »  
de « 1..\* »  
non traduit

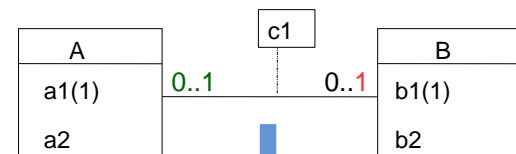
A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1 (NN), c1)

Contrainte textuelle  
B[b3] = A[a1]

18

## Règle 3 : associations 1-1

- Le « 1-1 » correspond aux multiplicités maximales apparaissant aux extrémités de l'association
- Ce type d'association devient **une clé étrangère dans la relation de notre choix** (privilégier la table avec le moins de lignes sauf si le 0 est autorisé d'un côté mais pas de l'autre)
- Les éventuels attributs portés par cette association deviennent des attributs de la relation contenant la clé étrangère



A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1, c1)

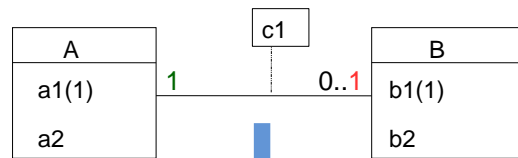
« 1 »  
de « 0..1 »  
non traduit

A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1 (UQ), c1)

19

## Règle 3 : associations 1-1

## Règle 3 : associations 1-1



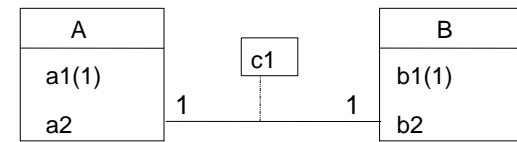
A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1 (NN), c1)



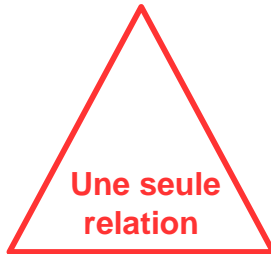
A(a1(1), a2)  
B(b1(1), b2, b3 = @A.a1 (NN) (UQ), c1)

21

## Règle 3 : associations 1-1



R(a1(1), b1(2), a2, b2, c1)

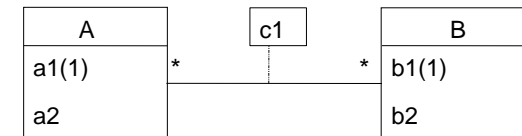


22

## Règle 4 : associations \*-\*

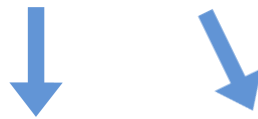
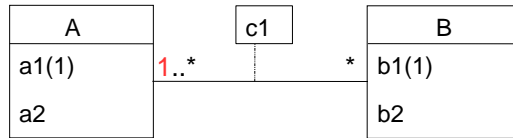
- Le « \*-\* » correspond aux multiplicités maximales apparaissant aux extrémités de l'association
- Ce type d'association devient **une nouvelle relation** (dite « table-association » au niveau de la base de données)
- La **clé primaire** de cette nouvelle relation est **composée des deux clés primaires** des relations associées
- Chaque composante de cette clé primaire** multiple (la plupart du temps double) est **une clé étrangère**
- Les éventuels attributs portés par cette association deviennent des attributs de cette nouvelle relation

## Règle 4 : associations \*-\*



A(a1(1), a2)  
B(b1(1), b2)  
R([a = @A.a1, b = @B.b1](1), c1)

## Règle 4 : associations \*-\*

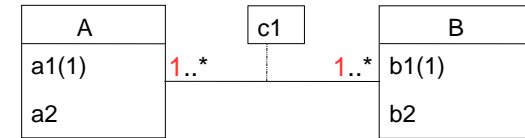


A(a1(1), a2)  
B(b1(1), b2)  
R([a=@A.a1, b=@B.b1](1), c1)

Contrainte textuelle  
 $B[b1] = R[b]$

25

## Règle 4 : associations \*-\*



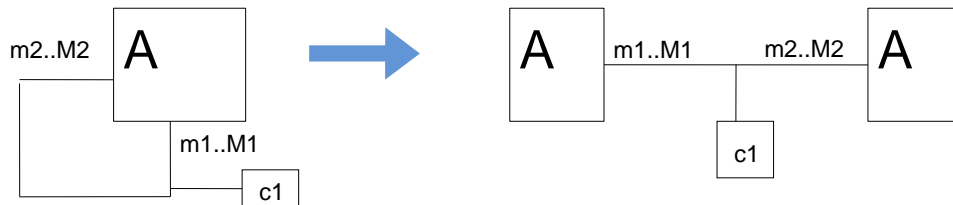
A(a1(1), a2)  
B(b1(1), b2)  
R([a=@A.a1, b=@B.b1](1), c1)

Contrainte textuelle  
 $B[b1] = R[b]$   
 $A[a1] = R[a]$

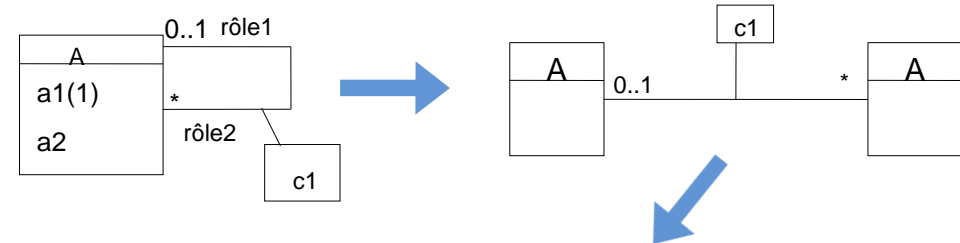
26

## Règle 5 : associations réflexives

Il faut « voir » les associations réflexives de la manière suivante et appliquer les règles précédentes !



## Règle 5 : associations réflexives

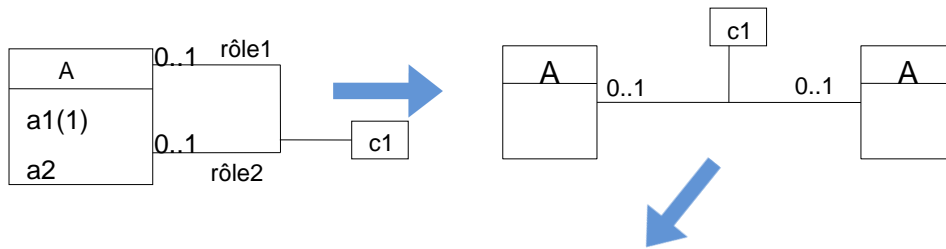


A(a1(1), a2, a3 = @A.a1, c1)

27

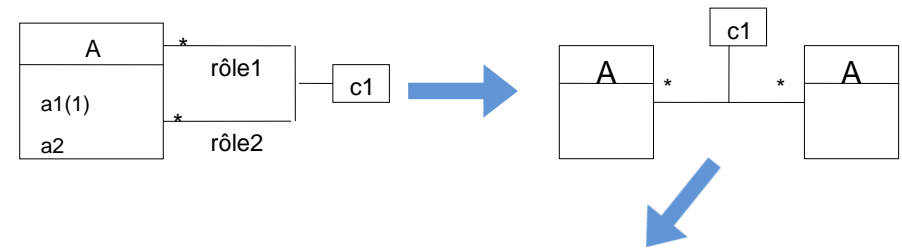
28

## Règle 5 : associations réflexives



$A(a1(1), a2, a3 = @A.a1 \text{ (UQ)}, c1)$

## Règle 5 : associations réflexives



$A(a1(1), a2)$   
 $R([r1=@A.a1, r2=@A.a2](1), c1)$

29

30

## Résumé (1/3) – A remplir

Type d'association	Cardinalités	Implantation
1-*	<b>A 0..1 ----- 0..* B</b> <b>Implantation de base</b>	A (idA(1), ...) B (idB(1), monA=@A.idA, ...)
1-*	<b>A 0..1 ----- 1..* B</b> <b>Spécialisation 1</b>	A (idA(1), ...) B (idB(1), monA=@A.idA, ...) Contrainte textuelle: B[monA] = A[idA]
1-*	<b>A 1..1 ----- 1..* B</b> <b>Spécialisation 2</b>	A (idA(1), ...) B (idB(1), monA=@A.idA(NN), ...) Contrainte textuelle: B[monA] = A[idA]
1-1	<b>A 0..1 ----- 0..1 B</b> <b>Implantation de base</b>	A (idA(1), ...)
1-1	<b>A 1..1 ----- 0..1 B</b> <b>Spécialisation 1</b>	A (idA(1), ...)
1-1	<b>A 1..1 ----- 1..1 B</b> <b>Spécialisation 2</b>	A (idA(1), ...)

31

## Résumé (2/3) – A remplir

Type d'association	Cardinalités	Implantation
*-*	<b>A 0..* ----- 0..* B</b> <b>Implantation de base</b>	
*-*	<b>A 1..* ----- 0..* B</b> <b>Spécialisation 1</b>	
*-*	<b>A 1..* ----- 1..* B</b> <b>Spécialisation 2</b>	
Réflexive 1-*	<b>A 0..1 ----- 0..* B</b> <b>Implantation de base</b>	
Réflexive 1-1	<b>A 0..1 ----- 0..1 B</b> <b>Implantation de base</b>	
Réflexive *-*	<b>A 0..* ----- 0..* B</b> <b>Implantation de base</b>	

32



### A retenir:

1. Toute classe devient une relation.
2. Une association est implantée soit sous la forme d'une nouvelle relation (cas **\*-\***) soit d'une clé étrangère (tous les autres cas).
3. Pour une association, ce sont les cardinalités maximums des deux côtés qui déterminent le *type d'association* et donc l'*implantation de base* retenue. a.
  - a) Les associations de type 1-1 et 1-\* relèvent de base d'un déport de clé d'une relation vers l'autre.
  - b) Les associations **\*-\*** imposent de base la création d'une troisième relation qui a pour clé la concaténation des clés étrangères des deux autres relations.
4. Les cardinalités minimales permettent, elles, d'affiner l'implantation de base précédente en lui ajoutant des contraintes : selon les cas (NN), (UQ), contrainte textuelle A[idA]=B[monA], etc.).
5. Les associations réflexives se traitent comme les autres associations. La relation joue simplement les deux rôles. On imagine que la relation est « dupliquée ».
6. Les attributs portés par les associations **\*-\*** sont placés dans la troisième relation, celle qui implémente l'association **\*-\***.

## Principales instructions SQL

### Partie 2 : Modèle relationnel et SQL

M. T. Pham, A. Ridard, B. Berg

#### Cours 2

### SQL – Langage de définition et manipulation des données

Création de tables, Insertion, Mis à jour et Suppression

- **Langage de Définition des Données (LDD)**  
CREATE, DROP, ALTER
- **Langage de Manipulation des Données (LMD)**
  - Insertion, Mis à jour, Suppression: INSERT, UPDATE, DELETE
  - Interrogations: SELECT
- **Langage de Contrôle des Données (LCD)**
  - Contrôle d'accès aux données  
GRANT, REVOKE
  - Gestion de transactions  
COMMIT, ROLLBACK



R1.05 – Période 2

## Règles de codage

- Le programmeur passant beaucoup moins de temps à écrire du code nouveau qu'à le relire et le modifier, il faut que **le code soit facile à lire**.
- **Les règles suivantes sont impératives (pour les TDs/TPs dans R1.05) :**
  - ✓ Tables (relations) en **minuscules avec la 1<sup>ère</sup> lettre en majuscule**
  - ✓ Attributs (titres des colonnes) en **minuscules**
  - ✓ **(1)** pour la **clé primaire**
  - ✓ Mots du SGBD en **majuscules**
  - ✓ **alignement des parenthèses et indentation**
  - ✓ **décomposition** systématique des instructions
  - ✓ **commentaires** des passages délicats

## Règles de codage

### ➤ Exemple des relations

Employé (id(1), nom, prénom, dpt)  
Produit (libellePro(1), prix, quantité)  
Entreprise (noEnt(1), nom, chiffreAffaire)

### ➤ Exemple des codes SQL (à pratiquer dans les TDs/TPs)

```
SELECT  DISTINCT    noEnt
FROM    Entreprise
WHERE   NOT EXISTS (SELECT  *
                        FROM  Employe
                        WHERE  lEntreprise = noEnt
                        AND    salaire ≥ 1300
                        ) ;
```

## Principales instructions SQL

- **Langage de Définition des Données (LDD)**  
**CREATE, DROP, ALTER**
- **Langage de Manipulation des Données (LMD)**
  - **Insertion, Mis à jour, Suppression:** INSERT, UPDATE, DELETE

## Exemple de référence

Schéma relationnel :

-----  
Client(idClient(1), nomClient(NN), prenomClient, telClient(UQ))

Produit(idProduit(1), libelleProduit, prix, prixPromo)

Commande((idP=@Produit.idProduit, idC=@Client.idClient)(1), quantite)

Contraintes textuelles :

- 
- Le prix d'un produit est strictement positif
  - Unicité du couple (nomClient, prenomClient)
  - PrixPromo < Prix

## Création de tables

- La création de tables s'effectue via la commande :

### CREATE TABLE

- Syntaxe de la commande :

```
CREATE TABLE Table_name (  
    [attribut type [contrainte_attribut], ...]  
    [contrainte_table, ...]  
);
```

- Une **contrainte d'attribut** porte sur un seul attribut
- Une **contrainte de table** porte sur plusieurs attributs

## Création de tables

- L'ordre de création est important

```
>_ CREATE TABLE Client ( ... );  
  
CREATE TABLE Produit ( ... );  
  
CREATE TABLE Commande (  
    idP NUMBER  
        CONSTRAINT fk_Commande_Produit REFERENCES Produit(idProduit),  
    idC NUMBER  
        CONSTRAINT fk_Commande_Client REFERENCES Client(idClient),  
    quantite NUMBER,  
    -- contrainte de table  
    CONSTRAINT pk_Commande PRIMARY KEY (idP, idC)  
);
```

## Destruction de tables

- La destruction de tables s'effectue via la commande :

### DROP TABLE

- Syntaxe de la commande :

```
DROP TABLE Table_name ;
```

- L'ordre de destruction est important

```
>_ DROP TABLE Commande ;  
  
DROP TABLE Produit ;  
  
DROP TABLE Client ;
```

## Altération de tables

- L'altération de tables s'effectue via la commande

### ALTER TABLE

- Il existe trois modes de spécification des altérations
  - ADD** : ajout de colonnes et de contraintes
  - MODIFY** : modification des types des colonnes
  - DROP** : suppression de colonnes et de contraintes

## Altération de tables

- Ajout, suppression ou modification de type de colonnes



```
ALTER TABLE Table_name ADD column_name type ;  
  
ALTER TABLE Table_name DROP COLUMN column_name ;  
  
ALTER TABLE Table_name MODIFY COLUMN column_name type;
```

- Ajout ou suppression de contraintes

```
ALTER TABLE Table_name ADD CONSTRAINT ...;  
  
ALTER TABLE Table_name DROP CONSTRAINT constraint_name ;
```

## Contraintes d'intégrité (à la création)

- Les contraintes gérées dans le script de création de tables :
  - Clé primaire
  - Clé étrangère
  - Existence (présence obligatoire)
  - Unicité
  - Autres vérifications
- La contrainte d'existence est toujours d'attribut
- Les autres peuvent être d'attribut ou de table

## Contrainte d'attribut : clé primaire

- Syntaxe :



```
CONSTRAINT < nom de la contrainte > PRIMARY KEY
```

- Convention de notation :
  - pk\_<nom de la table>
- Exemple :



```
idClient NUMBER  
  
CONSTRAINT pk_Client PRIMARY KEY
```

```
Client(idClient(1), nomClient(NN), prenomClient, telClient(UQ))
```

## Contrainte d'attribut : clé étrangère

- Syntaxe :



```
CONSTRAINT < nom de la contrainte >  
  
REFERENCES < table référencée > ( < attribut référencé > )
```

- Convention de notation :
  - fk\_<nom de la table>\_<nom de la table référencée>
- Exemple :



```
idP NUMBER  
  
CONSTRAINT fk_Commande_Produit REFERENCES Produit(idProduit)
```

```
Commande((idP=@Produit.idProduit, idC=@Client.idClient)(1), quantite)
```

## Contrainte d'attribut : existence

- Syntaxe :



```
CONSTRAINT < nom de la contrainte > NOT NULL
```

- Convention de notation :

- `nn_<nom de la colonne>`

- Exemple :



```
nomClient VARCHAR2(20)
        CONSTRAINT nn_nomClient NOT NULL
```

`Client(idClient(1), nomClient(NN), prenomClient, telClient(UQ))`

## Contrainte d'attribut : unicité

- Syntaxe :



```
CONSTRAINT < nom de la contrainte > UNIQUE
```

- Convention de notation :

- `uq_<nom de la colonne>`

- Exemple :



```
telClient VARCHAR2(10)
        CONSTRAINT uq_telClient UNIQUE
```

## Contrainte d'attribut : autres vérifications

- Syntaxe :



```
CONSTRAINT < nom de la contrainte > CHECK ( < condition à vérifier > )
```

- Convention de notation :

- `ck_<expression de la contrainte>`

- Exemple :



```
prix NUMBER
        CONSTRAINT ck_prixPositif CHECK ( prix > 0 )
```

Contraintes textuelles :

- Le prix d'un produit est strictement positif

## Contrainte de table : clé primaire

- Syntaxe :



```
CONSTRAINT < nom de la contrainte >
        PRIMARY KEY ( < liste des attributs composant la clé > )
```

- Convention de notation :

- `pk_<nom de la table>`

- Exemple :



```
CONSTRAINT pk_Commande PRIMARY KEY (idC, idP)
```

`Commande((idP=@Produit.idProduit, idC=@Client.idClient)(1), quantite)`

## Contrainte de table : clé étrangère

- Syntaxe :

```
CONSTRAINT < nom de la contrainte >  
  
FOREIGN KEY ( < liste des attributs référencants > )  
  
REFERENCES < table référencée > ( < liste des attributs référencés > )
```

- Convention de notation :
  - fk\_<nom de la table>\_<nom de la table référencée>
- Exemple : **si on considère [nomClient, prenomClient](1)**

```
>_ CONSTRAINT fk_Commande_Client  
  
FOREIGN KEY (leNomC, lePrenomC)  
  
REFERENCES Client(nomClient, prenomClient)
```

- Syntaxe :

```
CONSTRAINT < nom de la contrainte >  
  
UNIQUE ( < liste des attributs non duplicable > )
```

- Convention de notation :
  - uq\_<nom des colonnes>
- Exemple :

```
>_ CONSTRAINT uq_nomClient_PrenomClient UNIQUE (nomClient,  
prenomClient)
```

Contraintes textuelles :  
- Unicité du couple (nomClient, prenomClient)

## Contrainte de table : autres vérifications

- Syntaxe :

```
CONSTRAINT < nom de la contrainte >  
  
CHECK ( < condition multi-attributs à vérifier dans la table > )
```

- Convention de notation :
  - ck\_<expression de la contrainte>
- Exemple :

```
>_ CONSTRAINT ck_prixInfPrixPromo CHECK (prixPromo < prix)
```

Contraintes textuelles :  
- PrixPromo < Prix

## Contrainte de table : unicité

## Principales instructions SQL

- Langage de Définition des Données (LDD)  
CREATE, DROP, ALTER
- Langage de Manipulation des Données (LMD)
  - Insertion, Mis à jour, Suppression: INSERT, UPDATE, DELETE

## Exemple de référence

Schéma relationnel :

-----  
Client(idClient(1), nomClient(NN), prenomClient, telClient(UQ))

Produit(idProduit(1), libelleProduit, prix, prixPromo)

Commande((idP=@Produit.idProduit, idC=@Client.idClient)(1), quantite)

Contraintes textuelles :

- - Le prix d'un produit est strictement positif  
- Unicité du couple (nomClient, prenomClient)  
- PrixPromo < Prix

## Insertion des données

- On utilise l'instruction **INSERT**
- Syntaxe de la commande



```
INSERT INTO table_name [ (attribute, ...) ]  
  
VALUES (value, ...) ;
```

- Exemple



```
INSERT INTO Client (idClient, nomClient)  
VALUES (1, 'RIDARD');  
  
INSERT INTO Produit  
VALUES (1, 'libelle 1', 100, 80);  
  
INSERT INTO Produit  
VALUES (2, 'libelle 2', 100, NULL);
```

## Modification des données

- On utilise l'instruction **UPDATE**
- Syntaxe de la commande



```
UPDATE table_name  
  
SET column1 = value1, column2 = value2, ...  
  
[ WHERE condition ]
```

- Exemple



```
UPDATE Client  
SET prenomClient = 'Anthony'  
WHERE idClient = 1 ;  
  
UPDATE Produit  
SET prix = 80, prixPromo = 60  
WHERE prix = 100 ;
```

## Suppression des données

- On utilise l'instruction **DELETE**
- Syntaxe de la commande



```
DELETE FROM table_name  
  
[ WHERE condition ]
```

- Exemple



```
DELETE FROM Client ;  
  
DELETE FROM Produit  
WHERE prixPromo / prix > 0.7 ;
```

## Partie 2 : Modèle relationnel et SQL

M. T. Pham, A. Ridard, P. Berg

### Cours 3

## Algèbre Relationnelle et Requêtes SQL

L'**algèbre relationnelle** définit les opérations standards qui permettent de manipuler les relations pour en extraire l'information recherchée

## Opérateurs relationnels

- Unaires
- Binaires de même schéma
- Binaires de schémas quelconques

## Opérateurs relationnels

- Unaires
- Binaires de même schéma
- Binaires de schémas quelconques



- Projection
- Restriction (sélection)
- Tri

- Projection
- Restriction (sélection)
- Tri

### Projection

- Soient **R** une relation, **a** un attribut de **R**.
- **R[a]**, **projection de R sur a**, est la relation qui n'a qu'un attribut : **a** et dont les tuples représentent toutes les valeurs différentes de **a** dans **R**
- **R[a,b]**, **projection de R sur a et b**, est la relation qui a les deux attributs : **a** et **b**, et dont les tuples représentent toutes les valeurs différentes des couples **(a,b)** dans **R**.

### Exemple de projection

Voiture

imatriculation	marque	puissance	date1ereImm
24ET7898	RENAULT	7	23/07/2010
76YU9087	PEUGEOT	6	12/04/1999
75GY6435	AUDI	8	09/02/2008
67HR4321	PEUGEOT	7	17/11/2011
46FC5687	RENAULT	7	22/06/2007

- **Voiture[marque,puissance]**

## Projection élargie

- Soient **R** une relation, **a** un attribut de **R**, **f** une fonction dont l'ensemble de départ est le domaine de l'attribut **a**, et l'ensemble d'arrivée le domaine de l'attribut **b** (présent ou non dans **R**).
- **R[f(a)]** est la relation qui n'a qu'un attribut **b** et dont les tuples représentent toutes les valeurs différentes de **f(a)** dans **R**.

## Exemple de projection élargie

Voiture

imatriculation	marque	puissance	date1erelmm
24ET7898	RENAULT	7	23/07/2010
76YU9087	PEUGEOT	6	12/04/1999
75GY6435	AUDI	8	09/02/2008
67HR4321	PEUGEOT	7	17/11/2011
46FC5687	RENAULT	7	22/06/2007

- **Voiture[puissance+2]**

## Traduction d'une projection

- AR : **R[a, 3\*b]**

- SQL :

```
>_
SELECT DISTINCT a, 3*b
FROM R;
```

- Attention : **DISTINCT** !!!

## Projection + Renommage des colonnes (AS)

- AR : **R[a, 3\*b AS new\_name]**

- SQL :

```
>_
SELECT DISTINCT a, 3*b AS new_name
FROM R;
```

- Attention : **AS** est facultatif

## ➤ Projection

## ➤ Restriction (sélection)

## ➤ Tri

- Soient **R** une relation,  $v$  une valeur du domaine de l'attribut **a** de **R**
- **R{a = v}**, **restriction de R à a=v**, est la relation ayant tous les attributs de **R**, mais uniquement les tuples où l'attribut **a** prend la valeur  $v$
- Restriction suivant une condition quelconque

## Exemple de restriction

### Voiture

imatriculation	marque	puissance	date1ereImm
24ET7898	RENAULT	7	23/07/2010
76YU9087	PEUGEOT	6	12/04/1999
75GY6435	AUDI	8	09/02/2008
67HR4321	PEUGEOT	7	17/11/2011
46FC5687	RENAULT	7	22/06/2007

- `Voiture{marque = 'RENAULT'}`
- `Voiture{marque = 'RENAULT'}[puissance]` → ça donne ?

## Traduction d'une restriction

- AR : `R{a > 2}`

- SQL :

```
>_ SELECT *  
    FROM R  
   WHERE a>2;
```

## Restriction + Projection

➤ AR :  $R\{a > 2\}[b]$

➤ SQL :

```
>_
SELECT DISTINCT b
FROM R
WHERE a>2;
```

Attention : Il ne faut pas écrire  $R[b]\{a > 2\}$

## Restriction + Projection

**Produit** (nomProduit(1), prix, ...)

Quels sont les noms des produits dont le prix est plus cher ou égal à 18 euros ?

➤ AR :  $\text{Produit}\{\text{prix} \geq 18\}[\text{nomProduit}]$

```
>_
SELECT DISTINCT nomProduit
FROM Produit
WHERE prix >= 18;
```

➤ Attention : le **DISTINCT** dans ce cas n'est pas obligatoire (pourquoi ?)

## Recherche de motif (LIKE)

**Employe** (nom(1), fonction, salaire, adresse, ...)

Quels sont les noms des employés dont l'adresse contient 'Vannes' ?

➤ AR :  $R\{a > 2\}[b]$

```
>_
SELECT DISTINCT nom
FROM Employe
WHERE UPPER(adresse) LIKE 'VANNES';
```

➤ Attention : **UPPER** !!!

➤ On peut aussi utiliser « **NOT LIKE** »

## Données manquantes (IS NULL)

**Employe** (nom(1), prenom, fonction, salaire, adresse, ...)

Quels sont les employés dont on n'a pas enregistré le prénom ?

➤ AR :  $R\{\text{prenom} = ^\}$

```
>_
SELECT *
FROM Employe
WHERE prenom IS NULL ;
```

➤ Attention : (  $\text{prenom} = \text{NULL}$  ne fonctionne pas → faut utiliser **IS** )

## Données non manquantes

**Employe** (nom(1), prenom, fonction, salaire, adresse, ...)

Quels sont les employés dont on a enregistré le prénom ?

➤ AR : **R{prenom !=^}**



```
SELECT  *
FROM    Employe
WHERE   prenom IS NOT NULL ;
```

## Opérateurs relationnels unaires

➤ Projection

➤ Restriction (sélection)

➤ Tri

## Tri des T-uples

**Produit** (nomPro(1), prix, ...)

Quels sont les noms des produits ordonnés par ordre alphabétique ?



```
SELECT  DISTINCT nomPro
FROM    Produit
ORDER BY nomPro;
```

➤ AR : **Produit[nomPro] (nomPro>)**

➤ Attention : ordre croissant (ASC) par défaut (sinon il faut préciser DESC)

## Tri multi-attributs

**Produit** (nomPro(1), prix, ...)



```
SELECT  DISTINCT nomPro, prix
FROM    Produit
ORDER BY nomPro, prix DESC;
```

➤ AR : **Produit[nomPro,prix] (nomPro>) (prix<)**

## Comptage interne

- La 'pseudo-colonne' **ROWNUM** attribue à chaque tuple résultat de mapping un numéro de sortie.
- Ce numéro change à chaque mapping (à chaque instruction **SELECT**) et est donné avant le tri (dans le cas où l'utilisateur demande un tri par **ORDER BY**)

## Limitation du nombre des T-uples

**Produit** (nomPro(1), prix, ...)

Quels sont les noms des dix premiers produits ?

```
>_
SELECT  DISTINCT nomPro
FROM    Produit
WHERE   ROWNUM <= 10;
```

➤ AR : **Produit{ROWNUM<=10} [nomPro]**

## Tri + Limitation

**Produit** (nomPro(1), prix, ...)

Quels sont les dix prix de produits les plus chers ?

```
>_
SELECT  DISTINCT prix
FROM    Produit
WHERE   ROWNUM <= 10
ORDER BY prix DESC;
```

→ ne fonctionne pas !!! ( car la numérotation se fait avant le tri )

## Tri + Limitation

**Produit** (nomPro(1), prix, ...)

Quels sont les dix prix de produits les plus chers ?

```
>_
SELECT  *
FROM (   SELECT DISTINCT prix
        FROM    Produit
        ORDER BY prix DESC
      )
WHERE   ROWNUM <= 10 ;
```

→ Solution correcte !!!

- Unaires
- Binaires de même schéma
- Binaires de schémas quelconques


- Union
- Intersection
- Différence ensembliste

## Union, Intersection, Différence ensembliste

- Soient **R1** et **R2** deux relations **de même schéma**
- La réunion ou **union**  $R1 \cup R2$  est la relation de même schéma comprenant tous les tuples différents de **R1** ou **R2**
- L'**intersection**  $R1 \cap R2$  est la relation de même schéma comprenant tous les tuples de **R1** qui se retrouvent dans **R2**
- La **différence ensembliste**  $R1 \setminus R2$  est la relation de même schéma comprenant tous les tuples de **R1** qui ne se retrouvent pas dans **R2**

## Union, Intersection, Différence ensembliste

- Une requête s'exprime par un '**mapping**' contenant au moins un **SELECT**. Un mapping peut être simple (un seul **SELECT**), ou contenir des **sous-mappings**.



```
SELECT    DISTINCT ...  
FROM      ...  
WHERE     ...  
  
UNION/INTERSECT/MINUS  
  
SELECT    DISTINCT ...  
FROM      ...  
WHERE     ... ;
```

## Exemple d'union

R1

->

A1	A2	A3
a1	a2	a3
b1	b2	b3
c1	c2	c3
d1	d2	d3

->

R2

->

A1	A2	A3
a1	a2	a3
e1	e2	e3
d1	d2	d3

->

R1 U R2

A1	A2	A3
a1	a2	a3
b1	b2	b3
c1	c2	c3
d1	d2	d3
e1	e2	e3

## Exemple d'union

Voiture

immat	marque	puissance	date1erelmm
24ET7898	RENAULT	7	23/07/2010
76YU9087	PEUGEOT	6	12/04/1999
75GY6435	AUDI	8	09/02/2008
67HR4321	PEUGEOT	7	17/11/2011
46FC5687	RENAULT	7	22/06/2007

Moto

Immat	marque	puiss	date1erelmm
34E87	Yamaha	12	17/06/2004
87Y54	Yamaha	9	08/05/2010
98I09	Honda	8	24/07/2009

Voiture[marque,puissance] U Moto[marque,puiss] → à vous

## Exemple d'intersection

R1

->

A1	A2	A3
a1	a2	a3
b1	b2	b3
c1	c2	c3
d1	d2	d3

->

R2

->

A1	A2	A3
a1	a2	a3
e1	e2	e3
d1	d2	d3

->

R1 ∩ R2

A1	A2	A3
a1	a2	a3
d1	d2	d3

## Exemple d'intersection

Voiture

immat	marque	puissance	date1erelmm
24ET7898	RENAULT	7	23/07/2010
76YU9087	PEUGEOT	6	12/04/1999
75GY6435	AUDI	8	09/02/2008
67HR4321	PEUGEOT	7	17/11/2011
46FC5687	RENAULT	7	22/06/2007

Moto

Immat	marque	puiss	date1erelmm
34E87	Yamaha	12	17/06/2004
87Y54	Yamaha	9	08/05/2010
98I09	Honda	8	24/07/2009

Quelles sont les puissances communes aux voitures et motos ?

Voiture[puissance] ∩ Moto[puiss] → à vous



## Exemple de différence

**R1**

	A1	A2	A3
->	a1	a2	a3
	b1	b2	b3
	c1	c2	c3
->	d1	d2	d3

**R2**

	A1	A2	A3
->	a1	a2	a3
	e1	e2	e3
->	d1	d2	d3

**R1 - R2**

	A1	A2	A3
	b1	b2	b3
	c1	c2	c3

## Exemple d'une différence

Voiture

immat	marque	puissance	date1erelmm
24ET7898	RENAULT	7	23/07/2010
76YU9087	PEUGEOT	6	12/04/1999
75GY6435	AUDI	8	09/02/2008
67HR4321	PEUGEOT	7	17/11/2011
46FC5687	RENAULT	7	22/06/2007

Moto

Immat	marque	puiss	date1erelmm
34E87	Yamaha	12	17/06/2004
87Y54	Yamaha	9	08/05/2010
98I09	Honda	8	24/07/2009

Quelles sont les puissances des voitures qui ne sont pas la puissance d'une moto ?

Voiture[puissance] - Moto[puiss] → à vous

## Opérateurs relationnels

- Unaires
- Binaires de même schéma
- Binaires de schémas quelconques

## Opérateurs relationnels binaires de schémas quelconques

### Domaines quelconques :

- Produit (cartésien)

### Domaines non disjoints :

- Jointure (jointure naturelle, équi-jointure, théta-jointure)
- Division

## Produit cartésien

- Soient **R1** et **R2** deux relations de schémas quelconques
- Le **produit (cartésien)** **R1** **x** **R2** est la relation ayant tous les attributs de **R1**, tous les attributs de **R2**, et dont les tuples sont toutes les combinaisons possibles obtenues en juxtaposant un tuple de **R1** et un tuple de **R2**

## Exemple de produit

**R1**

A	B	C
a1	a2	a3
b1	b2	b3

**R2**

X	Y
x1	y1
x2	y2

**R1 x R2**

A	B	C	X	Y
a1	a2	a3	x1	y1
b1	b2	b3	x1	y1
a1	a2	a3	x2	y2
b1	b2	b3	x2	y2

## Exemple de produit

**Personne**

nom	prénom
Fourt	Lisa
Juny	Carole

**Cadeau**

article	prix
livre	15
montre	100
jeu	20

**Personne** **x** **Cadeau** ➔ **à vous**

## Traduction d'un produit

➤ AR : **R1** **x** **R2**

➤ SQL :

```
>_ SELECT *  
    FROM R1,R2  
    ;
```

## ➤ Jointure naturelle

## ➤ Equi-jointure

## ➤ Théta-jointure

## ➤ Jointure externe

- Soient **R1** et **R2** deux relations dont les schémas comportent **le même attribut A**, tous les autres attributs étant différents
- La **jointure naturelle** **R1 \* R2** est la relation ayant tous les attributs de **R1**, tous les attributs de **R2**, et dont les tuples sont toutes les combinaisons obtenues en juxtaposant un tuple de **R1** et un tuple de **R2** qui prennent **la même valeur pour l'attribut a**

## Jointure naturelle avec plusieurs attributs

- Si l'intersection des schémas de **R1** et **R2** comporte **plusieurs attributs a, b...**, la jointure naturelle se fera sur tous les attributs communs
- **R1 \* R2** est alors la relation ayant tous les attributs de **R1**, tous les attributs de **R2**, et dont les tuples sont toutes les combinaisons obtenues en juxtaposant un tuple de **R1** et un tuple de **R2** qui prennent **les mêmes valeurs pour a, b ...**

## Exemple de jointure naturelle

Personne

nom	prénom	âge
Fourt	Lisa	16
Juny	Carole	20
Fan	June	16

Cadeau

âge	article	prix
16	livre	15
20	montre	100
5	jeu	20

Personne \* Cadeau

nom	prénom	âge	article	prix
Fourt	Lisa	16	livre	15
Juny	Carole	20	montre	100
Fan	June	16	livre	15

## Equi-jointure

- Soient **R1** et **R2** deux relations dont les schémas comportent des attributs **a** (de R1) et **b** (de R2) de même domaine
- L'**équi-jointure** **R1 [ [a=b] ] R2** est la relation ayant tous les attributs de **R1**, tous les attributs de **R2**, et dont les tuples sont toutes les combinaisons obtenues en juxtaposant un tuple de **R1** et un tuple de **R2** qui prennent la même valeur pour les attributs **a** et **b**

## Equi-jointure et jointure naturelle

- Il est toujours possible de remplacer la jointure naturelle par une équi-jointure
- Si l'intersection des schémas de **R1** et **R2** est l'attribut **a**, alors :

$$R1 * R2 \Leftrightarrow R1 [ [ R1.a = R2.a ] ] R2$$

## Théta-jointure

- Soient **R1** et **R2** deux relations dont les schémas peuvent être liés par une condition **Cond**
- La **théta-jointure** **R1 [ [Cond] ] R2** est la relation ayant tous les attributs de **R1**, tous les attributs de **R2**, et dont les tuples sont toutes les combinaisons obtenues en juxtaposant un tuple de **R1** et un tuple de **R2** qui vérifient la condition **Cond**
- Une jointure naturelle, une équi-jointure sont des cas particuliers de théta-jointure

## Jointure externe

- **Jointure externe (gauche)** : les tuples de **R1** qui n'ont pas de valeur correspondante dans **R2** parmi les attributs communs de **R1** et **R2**, sont inclus dans la relation résultante. Les valeurs manquantes dans la seconde relation sont mises à **NULL**.
- **Jointure externe droit** : voir exemple
- **Jointure externe complète** : voir exemple

## Exemple de jointure externe

R1

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4

R1

C	D
c1	d1
c1	d2
c2	d3
c5	d5

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c2	d3

Jointure naturelle

B	C	D
b1	c1	d1
b1	c1	d2
b2	c2	d3
b3	c3	NULL
b4	c4	NULL

Jointure externe gauche

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c2	d3
NULL	NULL	c5	d5

Jointure externe droit

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c2	d3
a3	b3	c3	NULL
a4	b4	c4	NULL
NULL	NULL	c5	d5

Jointure externe complète

## Traduction d'une jointure

- AR :  $R1 * R2 \{ \text{cond} \}$  ou  $R1 [ [R1.a = R2.a] ] \{ \text{cond} \}$
- SQL : Une jointure naturelle se traduit toujours comme une théta-jointure (qui se traduit comme un **produit suivi d'une restriction**) :

>\_

```
SELECT *
FROM   R1, R2
WHERE  R1.a = R2.a -- condition de jointure
AND    cond ;      -- condition de restriction
```

- Attention : On peut utiliser un mapping imbriqué ou l'opérateur **JOIN** dans SQL → voir en S2

## Exemple de jointure

**Bateau** (noBateau(1), longBateau, .....)

**Emplacement** (noEmplacement(1), longPlace, unBateau = @Bateau[noBateau])

Donner toute information sur les bateaux de moins de 10 m qui sont sur des places de 10m ou plus ?

>\_

```
SELECT *
FROM   Emplacement, Bateau
WHERE  unBateau=noBateau
AND    longPlace >=10
AND    longBateau < 10 ;
```

- AR :  $\text{Emplacement} * \text{Bateau} \{ \text{longPlace} \geq 10 \ \& \ \text{longBateau} < 10 \}$

## Exemple d'auto jointure

**Personne** (nom(1), salaire, .....)

Qui gagne plus que moi ?

>\_

```
SELECT DISTINCT P2.nom
FROM   Personne P1, Personne P2
WHERE  P1.nom = 'moi'
AND    P2.nom != 'moi'
AND    P1.salaire < P2.salaire ;
```

- AR :  $\text{Personne } P1 * \text{Personne } P2 \{ P1.\text{nom} = \text{'moi'} \ \& \ P2.\text{nom} \neq \text{'moi'} \ \& \ P1.\text{salaire} < P2.\text{salaire} \}$

## Division

- Soient **R1** la relation définie sur l'ensemble d'attributs **Ea1** et **R2** la relation définie sur l'ensemble d'attributs **Ea2**, telle que:

$$Ea2 \subseteq Ea1$$

- Soit **Ea** = **Ea1** - **Ea2**
- La **division** **R1/R2** définit une relation sur les attributs **Ea**, constituée de l'ensemble des tuples de **R1** qui correspondent à la combinaison de **tous les** tuples de **R2**

## Exemple de division

<b>R1</b>	<b>A</b>	<b>X</b>	<b>R2</b>	<b>X</b>
	a1	x1		x1
	a2	x2		x2
	a3	x1		
	a1	x2		
	a2	x1		
<b>R1/R2</b>	<b>A</b>			
	a1			
	a2			

Plus des détails sur la division → Semestre 2

## Références

- <https://fr.wikipedia.org/>
- <https://fr.wikibooks.org/>
- Christian Soutou, *Modélisation des bases de données : UML et les modèles entité-association*, 3<sup>ème</sup> édition, Groupe Eyrolles.
- Christian Soutou, *SQL pour Oracle : Optimisation des requêtes et schémas*, 5<sup>ème</sup> édition, Groupe Eyrolles.
- Elisabetta De Mria, *Cours L2 Informatique*, UFR Sciences, Université Côte d'Azur, <https://www.i3s.unice.fr/%18edemaria/>
- SQL Tutorials, <https://www.w3schools.com/sql/default.asp>
- Introduction to SQL, Oracle Live SQL, <https://livesql.oracle.com/>