# Systèmes de gestion de bases de données

NFP 107

http://deptinfo.cnam.fr/new/spip.php?article685

C. Crochepeyre, M. Ferecatu, M. Crucianu, P. Rigaux, V. Thion, N. Travers
(prenom.nom)[at]cnam.fr

Équipe Vertigo
Laboratoire Cédric
Conservatoire national des arts & métiers, Paris, France
http://cedric.cnam.fr/vertigo



## Plan du cours





## Plan du cours







## Plan du cours

## Objectif du cours

COMPRENDRE et MAÎTRISER
la technologie des
BASES DE DONNÉES RELATIONNELLES





## Bibliographie

**Ouvrages en français**


1. Carrez C., *Des Structures aux Bases de Données*, Masson
2. Gardarin G., *Maîtriser les Bases de Données: modèles et langages*, Eyrolles
3. Date C.J, *Introduction aux Bases de Données*, Vuibert, 970 Pages, Janvier 2001
4. Akoka J. et Comyn-Wattiau I., *Conception des Bases de Données Relationnelles*, Vuibert Informatique
5. Rigaux P., *Cours de Bases de Données*, *http://dept25.cnam.fr/BDA/DOC/cbd.pdf*






## Bibliographie

**Ouvrages en anglais**


1. R. Ramakrishnan et J. Gehrke, *DATABASE MANAGEMENT SYSTEMS*, MacGraw Hill
2. R. Elmasri, S.B. Navathe, *Fundamentals of database systems*, 3e édition, 1007 pages, 2000, Addison Wesley
3. Ullman J.D. and Widom J. *A First Course in Database Systems*, Prentice Hall, 1997
4. H. Garcia - Molina, J.D. Ullman, J. Widom, *Database Systems : The Complete Book*, Hardcover, 2002
5. Garcia-Molina H., Ullman J. and Widom J., *Implementation of Database Systems*, Prentice Hall, 1999
6. Ullman J.D., *Principles of Database and Knowledge-Base Systems*, 2 volumes, Computer Science Press
7. Abiteboul S., Hull R., Vianu V., *Foundations of Databases*,Addison-Wesley






## Bibliographie

**Le standard SQL**


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


**Quelques systèmes**

1. BD2 (IBM),
2. Oracle (actuellement 11g),
3. SQL Server (Microsoft),
4. PostgreSQL,
5. MySQL.

## Bibliographie

**SQL "à la maison"**

1. MySQL, `http://www.mysql.org` (MS Windows, Linux)
   - Installation et interface Web via EasyPhp, `http://www.easyphp.org/`
   - Administration via MySQL Workbench, `http://dev.mysql.com/doc/workbench/en/`
2. PostgreSQL, `http://www.postgresql.org` (MS Windows, Linux)
   - Interface Web via PhpPgAdmin, `http://phppgadmin.sourceforge.net/`
   - Administration via PgAdmin, `http://www.pgadmin.org/`





## Applications des bases de données

1. **Applications "classiques" :**
   - Gestion de données : salaires, stocks, ...
   - Transactionnel : comptes bancaires, centrales d'achat, réservations
2. **Applications "modernes" :**
   - Documents électroniques : bibliothèques, journaux
   - Web : commerce électronique, serveurs Web
   - Génie logiciel : gestion de programmes, manuels, ...
   - Documentation technique : plans, dessins, ...
   - Bases de données spatiales : cartes routières, systèmes de guidage GPS, ...





## Problème central : comment stocker et manipuler les données?

Une base de données est
- un grand ensemble de données
- structurées et
- mémorisées sur un support permanent

Un système de gestion de bases de données (SGBD) est
- un logiciel de haut niveau d'abstraction qui permet de manipuler ces informations



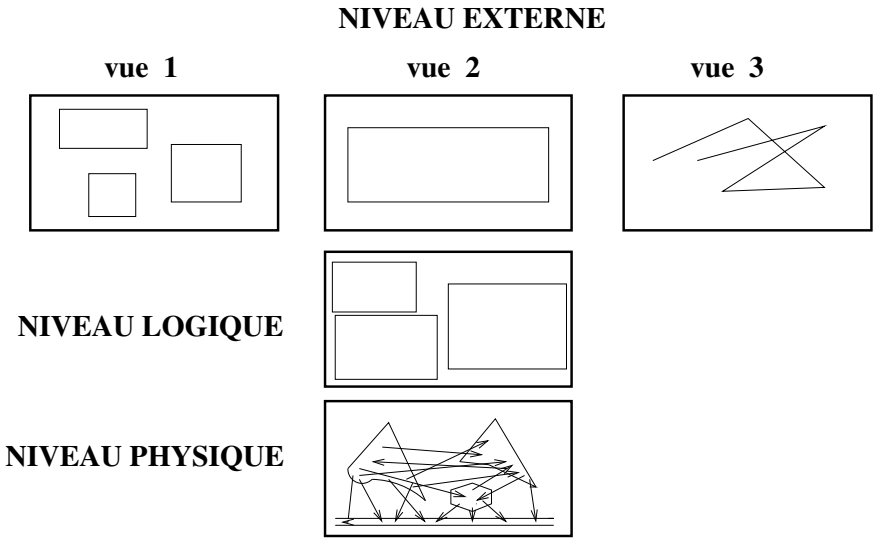


## Diversité → Complexité

**Diversité des utilisateurs, des interfaces et des architectures :**

1. diversité des utilisateurs : administrateurs, programmeurs, non informaticiens, ...
2. diversité des interfaces : langages BD, ETL, menus, saisies, rapports, ...
3. diversité des architectures : client-serveur centralisé/distribué
   *Aujourd'hui* : accès à plusieurs bases hétérogènes accessibles par réseau

## Architecture d'un SGBD : ANSI-SPARC (1975)








## Architecture d'un SGBD

Chaque niveau du SGBD réalise un certain nombre de fonctions :
**NIVEAU PHYSIQUE**

- Accès aux données, gestion sur mémoire secondaire (fichiers) des données, des index
- Partage de données et gestion de la concurrence d'accès
- Reprise sur pannes (fiabilité)
- Distribution des données et interopérabilité (accès aux réseaux)





## Architecture d'un SGBD

**NIVEAU LOGIQUE**

- Définition de la structure des données :
  **Langage de Description de Données (LDD)**
- Consultation et mise à jour des données :
  **Langages de Requêtes (LR)** et
  **Langage de Manipulation de Données (LMD)**





## Architecture d'un SGBD

**NIVEAU EXTERNE : Vues utilisateurs**
Exemple : base(s) de données du CNAM :

1. **Vue de la planification des salles :** pour chaque cours
   - Noms des enseignants
   - Horaires et salles
2. **Vue de la paye :** pour chaque enseignant
   - Nom, prénom, adresse, indice, nombre d'heures
3. **Vue du service de scolarité :** pour chaque élève
   - Nom, prénom, adresse, no d'immatriculation, inscriptions aux cours, résultats

## Intégration de ces vues

1. On laisse chaque usager avec sa vision du monde
2. Passage du niveau externe au niveau logique

On "intègre" l'ensemble de ces vues en une description **unique** :

**SCHÉMA LOGIQUE**





## Interfaces d'un SGBD

- Outils d'aide à la conception de schémas
- Outils de saisie et d'impression
- Outils ETL
- Interfaces d'interrogation (alphanumérique/graphique)
- Environnement de programmation : intégration des langages SGBD (LDD, LR, LMD) avec un langage de programmation (C++, Java, Php, Cobol, ...)
- API standards : ODBC, JDBC
- Importation/exportation de données (ex. documents XML)
- Débogueurs
- Passerelles (réseaux) vers d'autres SGBD





## En résumé

On veut gérer un grand volume de données

- structurées (ou semi-structurées),
- persistantes (stockées sur disque),
- cohérentes,
- fiables (protégées contres les pannes) et
- partagées entre utilisateurs et applications
- indépendamment de leur organisation physique





## Modèles de données

**Un modèle de données est caractérisé par :**

- Une STRUCTURATION des objets
- Des OPÉRATIONS sur ces objets

**Dans un SGBD, il existe plusieurs représentations plus ou moins abstraites de la même information :**

- le modèle conceptuel → description conceptuelle des données
- le modèle logique → programmation
- le modèle physique → stockage





## Exemple d'un modèle conceptuel: Le modèle Entités-Associations (*entity-relationship model*, P. Chen, 1976)

- Modèle *très abstrait*, utilisé pour :
  - l'analyse du monde réel et
  - la communication entre les différents acteurs (utilisateurs, administrateurs, programmeurs ...) pendant
  - la conception de la base de données
- **Mais n'est pas associé à un langage concret**.

DONC UNE STRUCTURE
MAIS PAS
D'OPÉRATIONS





## Modèle logique

1. **Langage de définition de données (LDD)** pour décrire la structure des données
2. **Langage de manipulation de données (LMD)** pour appliquer des opérations aux données

Ces langages font abstraction du niveau physique du SGBD :

1. Le LDD est indépendant de la représentation physique des données
2. Le LMD est indépendant de l'implantation des opérations





## Les avantages de l'abstraction

1. **SIMPLICITÉ**
   Les structures et les langages sont plus simples ("logiques", déclaratifs), donc plus faciles pour l'usager non expert
2. **INDÉPENDANCE PHYSIQUE**
   On peut modifier l'implantation/la représentation physique sans modifier les programmes/l'application
3. **INDÉPENDANCE LOGIQUE**
   On peut modifier les programmes/l'application sans toucher à la représentation physique des données

## Historique des modèles SGBD

**À chaque génération correspond un modèle logique**

| | | |
|---|---|---|
| $< 60$ | S.G.F. (e.g. COBOL) | |
| mi-60 | HIÉRARCHIQUE IMS (IBM) | navigationnel |
| | RÉSEAU (CODASYL) | navigationnel |
| 73-80 | RELATIONNEL | déclaratif |
| mi-80 | RELATIONNEL | explosion sur micro |
| Fin 80 | ORIENTÉ-OBJET | navig. + déclaratif |
| | RELATIONNEL ETENDU | nouvelles applications |
| | DATALOG (SGBD déductifs) | pas encore de marché |
| Fin 90 | XML | navig. + déclaratif |





## Exemples d'opérations

- *Insérer des informations concernant un employé nommé Jean*
- *Augmenter le salaire de Jean de 10%*
- *Détruire l'information concernant Jean*
- *Chercher les employés cadres*
- *Chercher les employés du département comptabilité*
- *Salaire moyen des employés comptables, avec deux enfants, nés avant 1960 et travaillant à Paris*





## Quels types d'opérations ?

4 types d'opérations :

1. **création** (ou **insertion**)
2. **modification** (ou **mise-à-jour**)
3. **destruction**
4. **recherche** (requêtes)

Ces opérations correspondent à des commandes du LMD et du LR. La plus complexe est la **recherche** (LR) en raison de la variété des critères





## Traitement d'une requête

- Analyse syntaxique
- Optimisation
  Génération (par le SGBD) d'un programme optimisé à partir de la connaissance de la structure des données, de l'existence d'index, de statistiques sur les données
- Exécution pour obtenir le résultat
  NB : on doit tenir compte du fait que d'autres utilisateurs sont peut-être en train de modifier les données qu'on interroge (concurrence d'accès)

## Concurrence d'accès

Plusieurs utilisateurs doivent pouvoir accéder en même temps aux mêmes données. Le SGBD doit savoir :

- Gérer les conflits si les utilisateurs font des mises-à-jour sur les mêmes données
- Donner une image cohérente des données si un utilisateur effectue des requêtes et un autre des mises-à-jour
- Offrir un mécanisme de retour en arrière si on décide d'annuler des modifications en cours

But : éviter les blocages, tout en empêchant des modifications anarchiques





## La gestion du SGBD

- **Le concepteur**
  - évalue les besoins de l'application
  - conçoit le schéma logique de la base
- **L'administrateur du SGBD**
  - installe le système et créé la base
  - conçoit le schéma physique
  - fait des réglages fins (*tuning*)
  - gère avec le concepteur l'évolution de la base (nouveaux besoins, utilisateurs)
- **L'éditeur du SGBD**
  - fournit le système et les outils





## Plan du cours







## Exemple de relation

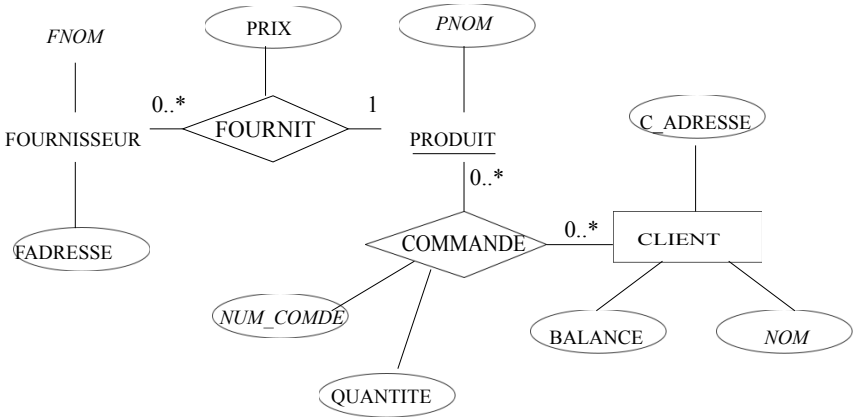
*Nom de la Relation* — *Nom d'Attribut*

**VEHICULE**

| Proprietaire | Type | Annee |
|---|---|---|
| Loic | Espace | 1988 |
| Nadia | Espace | 1989 |
| Loic | R5 | 1978 |
| Julien | R25 | 1989 |
| Marie | ZX | 1993 |

*n−uplet*

## Exemple de schéma conceptuel








## Traduction en schéma relationnel

- Le schéma conceptuel entités-associations est traduit en une ou plusieurs tables relationnelles
- Voir le cours du cycle préparatoire CNAM (`http://dept25.cnam.fr/BDA/DOC/cbd.pdf`) pour les méthodes de traductions





**FOURNISSEURS**

| FNOM | FADRESSE |
|---|---|
| Abounayan | 92190 Meudon |
| Cima | 75010 Paris |
| Preblocs | 92230 Gennevilliers |
| Sarnaco | 75116 Paris |

**FOURNITURES**

| FNOM | PNOM | PRIX |
|---|---|---|
| Abounayan | sable | 300 |
| Abounayan | briques | 1500 |
| Preblocs | parpaing | 1200 |
| Sarnaco | parpaing | 1150 |
| Sarnaco | ciment | 125 |





**CLIENTS**

| NOM | CADRESSE | BALANCE |
|---|---|---|
| Jean | 75006 Paris | -12000 |
| Paul | 75003 Paris | 0 |
| Vincent | 94200 Ivry | 3000 |
| Pierre | 92400 Courbevoie | 7000 |

**COMMANDES**

| NUM_COMDE | NOM | PNOM | QUANTITE |
|---|---|---|---|
| 1 | Jean | briques | 5 |
| 2 | Jean | ciment | 10 |
| 3 | Paul | briques | 3 |
| 4 | Paul | parpaing | 9 |
| 5 | Vincent | parpaing | 7 |

## Domaines, $n$-uplets et relations

- Un domaine est un *ensemble de valeurs*.
  Exemples : $\{0,1\}$, $\mathbb{N}$, l'ensemble des chaînes de caractères, l'ensemble des chaînes de caractères de longueur 10.
- Un $n$-uplet est une *liste de valeurs* $[v_1, \ldots, v_n]$ où chaque valeur $v_i$ est la valeur d'un domaine $D_i$ : $v_i \in D_i$
- Le produit cartésien $D_1 \times \ldots \times D_n$ entre des domaines $D_1, \ldots, D_n$ est *l'ensemble de* **tous** *les n-uplets* $[v_1, \ldots, v_n]$ où $v_i \in D_i$.
- Une relation $R$ est *un sous-ensemble fini* d'un produit cartésien $D_1 \times \ldots \times D_n$ : $R$ est un ensemble de $n$-uplets.
- Une base de données est un *ensemble de relations*.





## Attributs

Une relation $R \subset D_1 \times \ldots \times D_n$ est représentée sous forme d'une table où chaque ligne correspond à un élément de l'ensemble $R$ ($n$-uplet) :

- L'ordre des lignes n'a pas d'importance (ensemble).
- Les colonnes sont distinguées par leur ordre ou par un nom d'attribut.
  Soit $A_i$ le $i$-ème attribut de $R$ :
  - $n$ est appelé l'arité de la relation $R$.
  - $D_i$ est appelé le domaine de $A_i$.
  - Tous les attributs d'une relation ont un nom différent.
  - Un même nom d'attribut peut apparaître dans différentes relations.
  - Plusieurs attributs de la même relation peuvent avoir le même domaine.





## Schéma d'une base de données

- Le schéma d'une relation $R$ est défini par le nom de la relation et la liste des attributs avec pour chaque attribut son domaine :

  $$R(A_1 : D_1, \ldots, A_n : D_n)$$

  ou, plus simplement :

  $$R(A_1, \ldots, A_n)$$

  Exemple : VEHICULE(NOM:CHAR(20), TYPE:CHAR(10), ANNEE:ENTIER)
- Le schéma d'une base de données est l'ensemble des schémas de ses relations.





## Exemple de base de données

**SCHÉMA :**

- FOURNISSEURS(FNOM:CHAR(20), FADRESSE:CHAR(30))
- FOURNITURES(FNOM:CHAR(20), PNOM:CHAR(10), PRIX:ENTIER))
- COMMANDES(NUM_COMDE:ENTIER, NOM:CHAR(20), PNOM:CHAR(10), QUANTITE:ENTIER))
- CLIENTS(NOM: CHAR(20), CADRESSE:CHAR(30), BALANCE:RELATIF)

## Opérations sur une base de données relationnelle

- Langage de définition des données (définition et MAJ du schéma) :
  - création et destruction d'une relation ou d'une base
  - ajout, suppression d'un attribut
  - définition des contraintes (clés, références, ...)
- Langage de manipulation des données
  - saisie des $n$-uplets d'une relation
  - affichage d'une relation
  - modification d'une relation : insertion, suppression et maj des $n$-uplets
  - requêtes : consultation d'une ou de plusieures relations
- Gestion des transactions
- Gestion des vues





## Langages de requêtes relationnels

Pouvoir d'expression : Qu'est-ce qu'on peut calculer ? Quelles opérations peut-on faire ?
Fondements théoriques :

- calcul relationnel
  - logique du premier ordre, très étudiée (théorèmes)
  - langage déclaratif : on indique les propriétés que doivent vérifier les réponses à la requête
  - on n'indique pas comment les trouver
  - facile pour un utilisateur (logicien . . . )
- algèbre relationnelle
  - langage procédural, évaluateur facile à programmer
  - on indique comment trouver le résultat
  - difficile pour un utilisateur
- Théorème : ces deux langages ont le même pouvoir d'expression





## Langages de requêtes relationnels

En pratique, langage SQL :

- Langage déclaratif
- Plus naturel que logique du premier ordre
  - facile pour tout utilisateur
- Traduction automatique en algèbre relationnelle
- Évaluation de la requête à partir de l'algèbre
  - évaluation facile à programmer





## Plan du cours

## Algèbre relationnelle

Opérations "relationnelles" (ensemblistes) :

- une opération prend en entrée une ou deux relations (ensembles de $n$-uplets) de la base de données
- le résultat est toujours une relation (un ensemble)

5 opérations de base (pour exprimer toutes les requêtes) :

- opérations unaires : sélection, projection
- opérations binaires : union, différence, produit cartésien

Autres opérations qui s'expriment en fonction des 5 opérations de base : jointure, intersection et division





## Projection

Projection sur une partie (un sous-ensemble) des attributs d'une relation $R$. Notation :

$$\pi_{A_1,A_2,\ldots,A_k}(R)$$

$A_1, A_2, \ldots, A_k$ sont des attributs (du schéma) de la relation $R$. La projection "élimine" tous les autres attributs (colonnes) de $R$.





## Projection: Exemples

a) On élimine la colonne $C$ dans la relation $R$ :

| R | A | B | C |
|---|---|---|---|
| → | a | b | c |
| | d | a | b |
| | c | b | d |
| → | a | b | e |
| | e | e | a |

$\Rightarrow \pi_{A,B}(R)$

| A | B |
|---|---|
| a | b |
| d | a |
| c | b |
| e | e |

Le résultat est une relation (un ensemble) : le $n$-uplet $(a, b)$ n'apparaît qu'**une seule** fois dans la relation $\pi_{A,B}(R)$, bien qu'il existe **deux** $n$-uplets $(a, b, c)$ et $(a, b, e)$ dans $R$.





## Projection: Exemples

b) On élimine la colonne $B$ dans la relation $R$ (on garde $A$ et $C$) :

| R | A | B | C |
|---|---|---|---|
| | a | b | c |
| | d | a | b |
| | c | b | d |
| | a | b | e |
| | e | e | a |

$\Rightarrow \pi_{A,C}(R)$

| A | C |
|---|---|
| a | c |
| d | b |
| c | d |
| a | e |
| e | a |

## Sélection

Sélection avec une condition $\mathcal{C}$ sur les attributs d'une relation $R$: on garde les $n$-uplets de $R$ dont les attributs satisfont $\mathcal{C}$.
NOTATION :

$$\sigma_C(R)$$





## Sélection : exemples

a) On sélectionne les $n$-uplets dans la relation $R$ tels que l'attribut $B$ vaut "b" :

R

| A | B | C |
|---|---|---|
| a | b | 1 |
| d | a | 2 |
| c | b | 3 |
| a | b | 4 |
| e | e | 5 |

$\Rightarrow$ $\sigma_{B=''b''}(R)$

| A | B | C |
|---|---|---|
| a | b | 1 |
| c | b | 3 |
| a | b | 4 |





## Sélection : exemples

b) On sélectionne les $n$-uplets tels que

$$(A ='' a'' \vee B ='' a'') \wedge C \leq 3 :$$

R

| A | B | C |
|---|---|---|
| a | b | 1 |
| d | a | 2 |
| c | b | 3 |
| a | b | 4 |
| e | e | 5 |

$\Rightarrow$ $\sigma_{(A=''a''\vee B=''a'')\wedge C\leq 3}(R)$

| A | B | C |
|---|---|---|
| a | b | 1 |
| d | a | 2 |





## Sélection : exemples

c) On sélectionne les $n$-uplets tels que la 1re et la 2e colonne sont identiques :

R

| A | B | C |
|---|---|---|
| a | b | 1 |
| d | a | 2 |
| c | b | 3 |
| a | b | 4 |
| e | e | 5 |

$\Rightarrow$ $\sigma_{A=B}(R)$

| A | B | C |
|---|---|---|
| e | e | 5 |

## Condition de sélection

La condition $\mathcal{C}$ d'une sélection $\sigma_C(R)$ est une **formule logique** qui relie des termes de la forme $A_i \theta A_j$ ou $A_i \theta a$ avec les connecteurs logiques **et** ($\wedge$) et **ou** ($\vee$) où

- $A_i$ et $A_j$ sont des attributs de la relation $R$,
- $a$ est un élément (une valeur) du domaine de $A_i$,
- $\theta$ est un prédicat de comparaison ($=, <, \leq, >, \geq, \neq$).





## Expressions (requêtes) de l'algèbre relationnelle

Fermeture :

- Le résultat d'une opération est à nouveau une **relation**
- Sur cette relation, on peut faire une **autre opération** de l'algèbre

$\Rightarrow$ *Les opérations peuvent être* composées *pour former des expressions plus complexes de l'algèbre relationnelle.*





## Expressions de l'algèbre relationnelle

Exemple : $COMMANDES(NOM, PNOM, NUM, QTE)$

$$R'' = \pi_{PNOM}(\overbrace{\sigma_{NOM=''Jean''}(COMMANDES)}^{R'})$$

La relation $R'(NOM, PNOM, NUM, QTE)$ contient les $n$-uplets dont l'attribut $NOM$ a la valeur $''Jean''$. La relation $R''(PNOM)$ contient tous les produits commandés par Jean.





## Produit cartésien

- NOTATION : $R \times S$
- ARGUMENTS : 2 relations quelconques :

$$R(A_1, A_2, \ldots, A_n) \quad S(B_1, B_2, \ldots, B_k)$$

- SCHÉMA DE $T = R \times S$ : $T(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_k)$.
  On introduit les règles de renommage suivantes pour lever les éventuelles ambiguités sur le schéma de $T$:
  Si le produit cartésien est le produit d'une relation avec elle-même alors le nom de la relation est numéroté pour indentifier les deux rôles (par 1 et 2).
  Si les relations ont des attributs en commum, les noms des attributs en commun sont prefixés par le nom de la relation d'origine.
- VALEUR DE $T = R \times S$ : ensemble de tous les $n$-uplets ayant $n + k$ composants (attributs)
  - dont les $n$ premiers composants forment un $n$-uplet de $R$
  - et les $k$ derniers composants forment un $n$-uplet de $S$

## Exemple de produit cartésien

**R**

| A | B |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 3 | 4 |

$|R|$

**S**

| C | D | E |
|---|---|---|
| a | b | a |
| a | b | c |
| b | a | a |

$|S|$

$\Rightarrow$

**R × S**

$|R| \times |S|$

| A | B | C | D | E |
|---|---|---|---|---|
| 1 | 1 | a | b | a |
| 1 | 1 | a | b | c |
| 1 | 1 | b | a | a |
| 1 | 2 | a | b | a |
| 1 | 2 | a | b | c |
| 1 | 2 | b | a | a |
| 3 | 4 | a | b | a |
| 3 | 4 | a | b | c |
| 3 | 4 | b | a | a |





## Jointure naturelle

- NOTATION : $R \bowtie S$
- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) \quad S(B_1, \ldots, B_n, X_1, \ldots, X_k)$$

où $X_1, \ldots, X_k$ sont les attributs en commun.

- SCHÉMA DE $T = R \bowtie S$ : $T(A_1, \ldots, A_m, B_1, \ldots, B_n, X_1, \ldots, X_k)$
- VALEUR DE $T = R \bowtie S$ : ensemble de tous les $n$-uplets ayant $m + n + k$ attributs dont les $m$ premiers et $k$ derniers composants forment un $n$-uplet de $R$ et les $n + k$ derniers composants forment un $n$-uplet de $S$.





## Jointure naturelle: exemple

**R**

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

**S**

| B | C | D |
|---|---|---|
| b | c | d |
| b | c | e |
| a | d | b |
| a | c | c |

$\Rightarrow$

**R ⋈ S**

| A | B | C | D |
|---|---|---|---|
| a | b | c | d |
| a | b | c | e |
| d | b | c | d |
| d | b | c | e |
| c | a | d | b |





## Jointure naturelle

Soit $U = \{A_1, \ldots, A_m, B_1, \ldots, B_n, X_1, \ldots, X_k\}$ l'ensemble des attributs des 2 relations et $V = \{X_1, \ldots, X_k\}$ l'ensemble des attributs en commun.

$$R \bowtie S = \pi_U(\sigma_{\forall X \in V : R.X = S.X}(R \times S))$$

NOTATION : $R.X$ signifie "l'attribut $X$ de la relation $R$".

## Jointure naturelle : exemple

**R**

| A | B |
|---|---|
| 1 | a |
| 1 | b |
| 4 | a |

**S**

| A | B | D |
|---|---|---|
| 1 | a | b |
| 2 | c | b |
| 4 | a | a |

$\Rightarrow$

**R × S**

| R.A | R.B | S.A | S.B | D |
|---|---|---|---|---|
| 1 | a | 1 | a | b |
| **1** | **a** | **2** | **c** | b |
| **1** | **a** | **4** | **a** | a |
| **1** | **b** | **1** | **a** | b |
| **1** | **b** | **2** | **c** | b |
| **1** | **b** | **4** | **a** | a |
| **4** | **a** | **1** | **a** | b |
| **4** | **a** | **2** | **c** | b |
| 4 | a | 4 | a | a |

$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$
$R.A \neq S.A \rightarrow$
$R.B \neq S.B \rightarrow$
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$
$R.A \neq S.A \rightarrow$
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$

$\Downarrow$





## Jointure naturelle : exemple

$$\pi_{R.A,R.B,D}(\sigma_{R.A=S.A \wedge R.B=S.B}(R \times S))$$

$\Rightarrow$

**R ⋈ S**

| A | B | D |
|---|---|---|
| 1 | a | b |
| 4 | a | a |





## Jointure naturelle : algorithme de calcul

Pour chaque $n$-uplet $a$ dans $R$ et pour chaque $n$-uplet $b$ dans $S$ :

1. on concatène $a$ et $b$ et on obtient un $n$-uplet qui a pour attributs

$$\overbrace{A_1, \ldots, A_m, X_1, \ldots, X_k}^{a}, \overbrace{B_1, \ldots, B_n, X_1, \ldots, X_k}^{b}$$

2. on ne le garde que si chaque attribut $X_i$ de $a$ est égal à l'attribut $X_i$ de $b$ : $\forall_{i=1..k} a.X_i = b.X_i$.
3. on élimine les valeurs (colonnes) dupliquées : on obtient un $n$-uplet qui a pour attributs

$$\overbrace{A_1, \ldots, A_m}^{a}, \overbrace{B_1, \ldots, B_m}^{b}, \overbrace{X_1, \ldots, X_k}^{a \text{ et } b}$$





## $\theta$-Jointure

- ARGUMENTS : deux relations qui ne partagent pas d'attributs :

$$R(A_1, \ldots, A_m) \qquad S(B_1, \ldots, B_n)$$

- NOTATION : $R \bowtie_{A_i \theta B_j} S$, $\theta \in \{=, \neq, <, \leq, >, \geq\}$
- SCHÉMA DE $T = R \bowtie_{A_I \theta B_J} S$ : $T(A_1, \ldots, A_m, B_1, \ldots, B_n)$
- VALEUR DE $T = R \bowtie_{A_I \theta B_J} S$ : $T = \sigma_{A_i \theta B_j}(R \times S))$
- ÉQUIJOINTURE : $\theta$ est l'**égalité**.

## Exemple de $\theta$-Jointure : R $\bowtie_{A \leq C}$ S

**R**

| A | B |
|---|---|
| 1 | a |
| 1 | b |
| 3 | a |

**S**

| C | D | E |
|---|---|---|
| 1 | b | a |
| 2 | b | c |
| 4 | a | a |

**T := R × S**

| | A | B | C | D | E |
|---|---|---|---|---|---|
| | 1 | a | 1 | b | a |
| | 1 | a | 2 | b | c |
| | 1 | a | 4 | a | a |
| | 1 | b | 1 | b | a |
| | 1 | b | 2 | b | c |
| | 1 | b | 4 | a | a |
| $A > C \rightarrow$ | **3** | a | **1** | b | a |
| $A > C \rightarrow$ | **3** | a | **2** | b | c |
| | 3 | a | 4 | a | a |

$\Rightarrow$ $\sigma_{A \leq C}$**(T)** = **R** $\bowtie_{A \leq C}$ **S**

| A | B | C | D | E |
|---|---|---|---|---|
| 1 | a | 1 | b | a |
| 1 | a | 2 | b | c |
| 1 | a | 4 | a | a |
| 1 | b | 1 | b | a |
| 1 | b | 2 | b | c |
| 1 | b | 4 | a | a |
| 3 | a | 4 | a | a |





## Exemple d'équijointure : R $\bowtie_{B=D}$ S

**R**

| A | B |
|---|---|
| 1 | a |
| 1 | b |
| 3 | a |

**S**

| C | D | E |
|---|---|---|
| 1 | b | a |
| 2 | b | c |
| 4 | a | a |

**T := R × S**

| | A | B | C | D | E |
|---|---|---|---|---|---|
| $B \neq D \rightarrow$ | 1 | **a** | 1 | **b** | a |
| $B \neq D \rightarrow$ | 1 | **a** | 2 | **b** | c |
| | 1 | a | 4 | a | a |
| | 1 | b | 1 | b | a |
| | 1 | b | 2 | b | c |
| $B \neq D \rightarrow$ | 1 | **b** | 4 | **a** | a |
| $B \neq D \rightarrow$ | 3 | **a** | 1 | **b** | a |
| $B \neq D \rightarrow$ | 3 | **a** | 2 | **b** | c |
| | 3 | a | 4 | a | a |

$\Rightarrow$ $\sigma_{B=D}$**(T)** = **R** $\bowtie_{B=D}$ **S**

| A | B | C | D | E |
|---|---|---|---|---|
| 1 | a | 4 | a | a |
| 1 | b | 1 | b | a |
| 1 | b | 2 | b | c |
| 3 | a | 4 | a | a |





## Utilisation de l'équijointure et jointure naturelle

$IMMEUBLE(ADI, NBETAGES, DATEC, PROP)$

$APPIM(ADI, NAP, OCCUP, ETAGE)$

1. Nom du propriétaire de l'immeuble où est situé l'appartement occupé par *Durand* :

$$\pi_{PROP}(\overbrace{IMMEUBLE \bowtie \sigma_{OCCUP=''DURAND''}(APPIM)}^{JointureNaturelle})$$

2. Appartements occupés par des propriétaires d'immeuble :

$$\pi_{ADI,NAP,ETAGE}\overbrace{(APPIM \bowtie_{OCCUP=PROP} IMMEUBLE)}^{équijointure}$$





Autre exemple de requête : Nom et adresse des clients qui ont commandé des parpaings.

- Schéma Relationnel :

$$COMMANDES(PNOM, CNOM, NUM\_CMDE, QTE)$$

$$CLIENTS(CNOM, CADRESSE, BALANCE)$$

- Requête Relationnelle :

$$\pi_{CNOM,CADRESSE}(CLIENTS \bowtie \sigma_{PNOM=''PARPAING''}(COMMANDES))$$

## Union

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \qquad S(A_1, \ldots, A_m)$$

- NOTATION : $R \cup S$
- SCHÉMA DE $T = R \cup S$ : $T(A_1, \ldots, A_m)$
- VALEUR DE $T$ : Union ensembliste sur $D_1 \times \ldots \times D_m$ :

$$T = \{t \mid t \in R \vee t \in S\}$$





## Exemple d'union

R

| A | B |
|---|---|
| a | b |
| a | c |
| d | e |

S

| A | B |
|---|---|
| a | b |
| a | e |
| d | e |
| f | g |

$\Rightarrow$

R ∪ S

| A | B |
|---|---|
| a | b |
| a | c |
| d | e |
| a | e |
| f | g |





## Différence

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \qquad S(A_1, \ldots, A_m)$$

- NOTATION : $R - S$
- SCHÉMA DE $T = R - S$ : $T(A_1, \ldots, A_m)$
- VALEUR DE $T$ : Différence ensembliste sur $D_1 \times \ldots \times D_m$ :

$$T = \{t \mid t \in R \wedge t \notin S\}$$





## Exemple de différence

R

| | A | B |
|---|---|---|
| → | a | b |
| | a | c |
| → | d | e |

S

| | A | B |
|---|---|---|
| → | a | b |
| | a | e |
| → | d | e |
| | f | g |

R - S

| A | B |
|---|---|
| a | c |

S - R

| A | B |
|---|---|
| a | e |
| f | g |

## Intersection

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \qquad S(A_1, \ldots, A_m)$$

- NOTATION : $R \cap S$
- SCHÉMA DE $T = R \cap S$ : $T(A_1, \ldots, A_m)$
- VALEUR DE $T$ : Intersection ensembliste sur $D_1 \times \ldots \times D_m$ :

$$T = \{t \mid t \in R \wedge t \in S\}$$





## Exemple d'intersection

R

| | A | B |
|---|---|---|
| → | a | b |
| | a | c |
| → | d | e |

S

| | A | B |
|---|---|---|
| → | a | b |
| | a | e |
| → | d | e |
| | f | g |

R - S

| A | B |
|---|---|
| a | c |

R ∩ S = R - (R - S)

| A | B |
|---|---|
| a | b |
| d | e |





## Semi-jointure

- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k)$$

$$S(B_1, \ldots, B_n, X_1, \ldots, X_k)$$

où $X_1, \ldots, X_k$ sont les attributs en commun.

- NOTATION : $R \ltimes S$
- SCHÉMA DE $T = R \ltimes S$ : $T(A_1, \ldots, A_m, X_1, \ldots, X_k)$
- VALEUR DE $T = R \ltimes S$ : Projection sur les attributs de $R$ de la jointure naturelle entre $R$ et $S$.





## Semi-jointure

La semi-jointure correspond à une sélection où la condition de sélection est définie par le biais d'une autre relation.
Soit $U$ l'ensemble des attributs de $R$.

$$R \ltimes S = \pi_U(R \bowtie S)$$

## Exemple de semi-jointure

R

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

S

| B | C | D |
|---|---|---|
| b | c | d |
| b | c | e |
| a | d | b |

$\Rightarrow \pi_{A,B,C}(R \ltimes S) \Rightarrow$

R $\ltimes$ S

| A | B | C |
|---|---|---|
| a | b | c |
| d | b | c |
| c | a | d |





## Exemple de division

REQUÊTE : Clients qui commandent tous les produits:

COMM

| NUM | NOM | PNOM | QTE |
|---|---|---|---|
| 1 | Jean | briques | 100 |
| 2 | Jean | ciment | 2 |
| 3 | Jean | parpaing | 2 |
| 4 | Paul | briques | 200 |
| 5 | Paul | parpaing | 3 |
| 6 | Vincent | parpaing | 3 |





$R = \pi_{NOM,PNOM}(COMM)$ :

R

| NOM | PNOM |
|---|---|
| Jean | briques |
| Jean | ciment |
| Jean | parpaing |
| Paul | briques |
| Paul | parpaing |
| Vincent | parpaing |

PROD

| PNOM |
|---|
| briques |
| ciment |
| parpaing |

$\Downarrow$

$R \div PROD$

| NOM |
|---|
| Jean |





## Exemple de division

**R**

| A | B | C | D |
|---|---|---|---|
| a | b | x | m |
| a | b | y | n |
| a | b | z | o |
| b | c | x | o |
| b | d | x | m |
| c | e | x | m |
| c | e | y | n |
| c | e | z | o |
| d | a | z | p |
| d | a | y | m |

**S**

| C | D |
|---|---|
| x | m |
| y | n |
| z | o |

**R÷S**

| A | B |
|---|---|
| a | b |
| c | e |

## Division

- ARGUMENTS : 2 relations :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) \qquad S(X_1, \ldots, X_k)$$

où **tous** les attributs de $S$ sont des attributs de $R$.
- NOTATION : $R \div S$
- SCHÉMA DE $T = R \div S$ : $T(A_1, \ldots, A_m)$
- VALEUR DE $T = R \div S$ :

$$R \div S = \{(a_1, \ldots, a_m) \mid \forall (x_1, \ldots, x_k) \in S \ : (a_1, \ldots, a_m, x_1, \ldots, x_k) \in R\}$$





## Division

La division s'exprime en fonction du produit cartésien, de la projection et de la différence : $R \div S = R_1 - R_2$ où

$$R_1 = \pi_{A_1,\ldots,A_m}(R) \text{ et } R_2 = \pi_{A_1,\ldots,A_m}((R_1 \times S) - R)$$





## Renommage

- NOTATION : $\rho$
- ARGUMENTS : 1 relation :

$$R(A_1, \ldots, A_n)$$

- SCHÉMA DE $T = \rho_{A_i \to B_i} R$ : $T(A_1, \ldots, A_{i-1}, B_i, A_{i+1}, \ldots, A_n)$
- VALEUR DE $T = \rho_{A_i \to B_i} R$ : $T = R$. La valeur de R est inchangée. Seul le nom de l'attribut $A_i$ a été remplacé par $B_i$





## Plan du cours

## Principe

- SQL (Structured Query Language) est le Langage de Requêtes standard pour les SGBD relationnels
- Expression d'une requête par un bloc *SELECT FROM WHERE*

  SELECT <liste des attributs a projeter>
  FROM <liste des relations arguments>
  WHERE <conditions sur un ou plusieurs attributs>

- Dans les requêtes simples, la correspondance avec l'algèbre relationnelle est facile à mettre en évidence.





## Projection

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE : *Toutes les commandes*
ALGÈBRE : *COMMANDES*
SQL:

```
SELECT NUM,CNOM,PNOM,QUANTITE
FROM   COMMANDES
```

ou

```
SELECT *
FROM   COMMANDES
```





## Projection avec élimination de doublons

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Produits commandés*
ALGÈBRE : $\pi_{PNOM}(COMMANDES)$
SQL :

```
SELECT PNOM
FROM   COMMANDES
```

**NOTE:** Contrairement à l'algèbre relationnelle, SQL n'élimine pas les doublons (sémantique multi-ensemble). Pour les éliminer on utilise DISTINCT :

```
SELECT DISTINCT PNOM
FROM   COMMANDES
```

Le DISTINCT peut être remplacé par la clause UNIQUE.





## Sélection

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Produits commandés par Jean*
ALGÈBRE: $\pi_{PNOM}(\sigma_{CNOM=''JEAN''}(COMMANDES))$
SQL:

```
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'JEAN'
```

REQUÊTE: *Produits commandés par Jean en quantité supérieure à 100*
ALGÈBRE: $\pi_{PNOM}(\sigma_{CNOM="JEAN" \wedge QUANTITE>100}(COMMANDES))$
SQL:

```
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'JEAN' AND QUANTITE > 100
```





## Conditions simples

Les conditions de base sont exprimées de deux façons:

1. *attribut comparateur valeur*
2. *attribut comparateur attribut*

où *comparateur* est $=, <, >, !=, \ldots,$
Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)
Exemple :

```
SELECT PNOM FROM FOURNITURE WHERE PRIX > 2000
```





## Exemple

SCHÉMA : **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Produits dont le nom est celui du fournisseur*
SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PNOM = FNOM
```





## Appartenance à une intervalle : BETWEEN

SCHÉMA : **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Produits avec un coût entre 1000 euros et 2000 euros*
SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX BETWEEN 1000 AND 2000
```

**NOTE:** La condition $y$ BETWEEN $x$ AND $z$ est équivalente à $y <= z$ AND $x <= y$.

## Chaînes de caractères : LIKE

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Clients dont le nom commence par "C"*
SQL:

```
SELECT CNOM
FROM   COMMANDES
WHERE  CNOM LIKE 'C%'
```

**NOTE:** Le littéral qui suit LIKE doit être une chaîne de caractères éventuellement avec des caractères jokers _ (un caractère quelconque) et % (une chaîne de caractères quelconque).
*Pas exprimable avec l'algèbre relationnelle.*





## Valeurs inconnues : NULL

La valeur NULL est une valeur "spéciale" qui représente une *valeur (information) inconnue*.

1. $A\ \theta\ B$ est `inconnu` (ni vrai, ni faux) si la valeur de $A$ ou/et $B$ est NULL ($\theta$ est l'un de $=, <, >, !=, ...$).
2. $A\ op\ B$ est NULL si la valeur de $A$ ou/et $B$ est NULL ($op$ est l'un de $+, -, *, /$).





## Comparaison avec valeurs nulles

SCHÉMA et INSTANCE :

| FOURNISSEUR | FNOM | VILLE |
|---|---|---|
| | Toto | Paris |
| | Lulu | *NULL* |
| | Marco | Marseille |

REQUÊTE: *Les Fournisseurs de Paris.*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE = 'Paris'
```

RÉPONSE : Toto





## Comparaison avec valeurs nulles

REQUÊTE: *Fournisseurs dont la ville est inconnue.*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE = NULL
```

La réponse est vide. Pourquoi?
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE IS NULL
```

RÉPONSE : Lulu

## Trois valeurs de vérité

Trois valeurs de vérité: `vrai`, `faux` et **inconnu**

1. `vrai AND inconnu = inconnu`
2. `faux AND inconnu = faux`
3. `inconnu AND inconnu = inconnu`
4. `vrai OR inconnu = vrai`
5. `faux OR inconnu = inconnu`
6. `inconnu OR inconnu = inconnu`
7. `NOT inconnu = inconnu`





## Exemple

SCHÉMA : **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)
SQL:

```
SELECT ENOM
  FROM EMPLOYE
 WHERE SAL > 2000 OR SAL <= 6000
```

On ne trouve que les noms des employés avec un salaire connu. Pourquoi?





## Jointures : exemple

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE : *Nom, Coût, Fournisseur des Produits commandés par Jean*
ALGÈBRE :

$\pi_{PNOM,PRIX,FNOM}(\sigma_{CNOM="JEAN"}(COMMANDES) \bowtie (FOURNITURE))$





## Jointure : exemple

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
**FOURNITURE**(PNOM,FNOM,PRIX)
SQL :

```
SELECT COMMANDES.PNOM, PRIX, FNOM
FROM   COMMANDES, FOURNITURE
WHERE  CNOM = 'JEAN' AND
       COMMANDES.PNOM = FOURNITURE.PNOM
```

**NOTES:**

- On exprime une jointure comme un produit cartésien suivi d'une sélection et d'une projection (on a déjà vu ça?)
- Algèbre : la requête contient une jointure naturelle.
- SQL : il faut expliciter les attributs de jointure.

## Auto-jointure et renommage

SCHÉMA : **FOURNISSEUR**(FNOM,STATUT,VILLE)
REQUÊTE: *"Couples" de fournisseurs situés dans la même ville*
SQL:

```
SELECT PREM.FNOM, SECOND.FNOM
FROM   FOURNISSEUR PREM, FOURNISSEUR SECOND
WHERE  PREM.VILLE =  SECOND.VILLE AND
       PREM.FNOM < SECOND.FNOM
```

La deuxième condition permet

1. l'élimination des paires (x,x)
2. de garder un exemplaire parmi les couples symétriques (x,y) et (y,x)

**NOTE:** PREM représente une instance de FOURNISSEUR, SECOND une autre instance de FOURNISSEUR.





## Auto-jointure

SCHÉMA : **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)
REQUÊTE: *Nom et Salaire des Employés gagnant plus que l'employé de numéro 12546*
SQL:

```
SELECT E1.ENOM, E1.SAL
FROM   EMPLOYE E1, EMPLOYE E2
WHERE  E2.EMPNO = 12546 AND
       E1.SAL > E2.SAL
```

- On confond souvent les auto-jointures avec des sélections simples.
- Requête en algèbre?





## Opérations de jointure

SQL2 introduit des opérations de jointure dans la clause FROM :

| **SQL2** | opération | **Algèbre** |
|---|---|---|
| R1 CROSS JOIN R2 | produit cartesien | $R1 \times R2$ |
| R1 JOIN R2 ON R1.A < R2.B | théta-jointure | $R1 \bowtie_{R1.A<R2.B} R2$ |
| R1 NATURAL JOIN R2 | jointure naturelle | $R1 \bowtie R2$ |





## Jointure naturelle : exemple

SCHEMA: **EMP**(EMPNO,ENOM,DEPNO,SAL)
**DEPT**(DEPNO,DNOM)
REQUÊTE: *Numéros des départements avec les noms de leurs employés.*
SQL2:

```
SELECT DEPNO, ENOM
FROM   DEPT NATURAL JOIN EMP
```

**Note :** L'expression `DEPT NATURAL JOIN EMP` fait la jointure naturelle (sur les attributs en commun) et l'attribut `DEPNO` n'apparaît qu'une seule fois dans le schéma du résultat.

## $\theta$-jointure : exemple

REQUÊTE: *Nom et salaire des employés gagnant plus que l'employé 12546*
SQL2:

```
SELECT E1.ENOM, E1.SAL
FROM   EMPLOYE E1 JOIN EMPLOYE E2 ON E1.SAL > E2.SAL
WHERE  E2.EMPNO = 12546
```





## Jointure interne

**EMP**

| **EMPNO** | *DEPNO* | SAL |
|---|---|---|
| Tom | 1 | 10000 |
| Jim | 2 | 20000 |
| Karin | 3 | 15000 |

**DEPT**

| **DEPNO** | DNOM |
|---|---|
| 1 | Comm. |
| 2 | Adm. |
| 4 | Tech. |

**Jointure (interne) :** les n-uplets qui ne peuvent pas être joints sont éliminés :

**EMP NATURAL JOIN DEPT**

| | | | |
|---|---|---|---|
| Tom | 1 | 10000 | Comm. |
| Jim | 2 | 20000 | Adm. |





## Jointure externe

**Jointure externe :** les n-uplets qui ne peuvent pas être joints *ne sont pas éliminés*.

- On garde tous les n-uplets des deux relations :

**EMP NATURAL *FULL* OUTER JOIN DEPT**

| | | | |
|---|---|---|---|
| Tom | 1 | 10000 | Comm. |
| Jim | 2 | 20000 | Adm. |
| Karin | 3 | 15000 | NULL |
| NULL | 4 | NULL | Tech. |





- On garde tous les n-uplets de la première relation (gauche) :

**EMP NATURAL *LEFT* OUTER JOIN DEPT**

| | | | |
|---|---|---|---|
| Tom | 1 | 10000 | Comm. |
| Jim | 2 | 20000 | Adm. |
| Karin | 3 | 15000 | NULL |

- On peut aussi écrire (dans Oracle) :

```
select EMP.*, DEP.DNOM
from EMP, DEPT
where EMP.DEPNO = DEPT.DEPNO (+)
```

- On garde tous les n-uplets de la deuxième relation (droite) :

**EMP NATURAL *RIGHT* OUTER JOIN DEPT**

| | | | |
|---|---|---|---|
| Tom | 1 | 10000 | Comm. |
| Jim | 2 | 20000 | Adm. |
| NULL | 4 | NULL | Tech. |

- On peut aussi écrire (dans Oracle) :

```
select EMP.*, DEP.DNOM
from EMP, DEPT
where EMP.DEPNO (+) = DEPT.DEPNO
```





## Jointures externes dans SQL2

- R1 NATURAL FULL OUTER JOIN R2 : Remplir R1.* et R2.*
- R1 NATURAL LEFT OUTER JOIN R2 : Remplir R2.*
- R1 NATURAL RIGHT OUTER JOIN R2 : Remplir R1.*

avec NULL quand nécessaire.
D'une manière similaire on peut définir des théta-jointures externes :

- R1 (FULL|LEFT|RIGHT) OUTER JOIN R2 ON prédicat





## Union

**COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Produits qui coûtent plus que 1000F ou ceux qui sont commandés par Jean*

ALGÈBRE:

$$\pi_{PNOM}(\sigma_{PRIX>1000}(FOURNITURE))$$
$$\cup$$
$$\pi_{PNOM}(\sigma_{CNOM='Jean'}(COMMANDES))$$





SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX >= 1000
UNION
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'Jean'
```

**NOTE:** L'union élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `UNION ALL` : le résultat contient chaque n-uplet $a + b$ fois, où $a$ et $b$ est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

## Différence

La différence ne fait pas partie du standard.
**EMPLOYE**(EMPNO,ENOM,DEPTNO,SAL)
**DEPARTEMENT**(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
ALGÈBRE: $\pi_{DEPTNO}(DEPARTEMENT) - \pi_{DEPTNO}(EMPLOYE)$
SQL:

```
SELECT DEPTNO FROM   DEPARTEMENT
   EXCEPT
SELECT DEPTNO FROM   EMPLOYE
```

**NOTE:** La différence élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `EXCEPT ALL` : le résultat contient chaque n-uplet $a - b$ fois, où $a$ et $b$ est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.





## Intersection

L'intersection ne fait pas partie du standard.
**EMPLOYE**(EMPNO,ENOM,DEPTNO,SAL)
**DEPARTEMENT**(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements ayant des employés qui gagnent plus que 20000F et qui se trouvent à Paris*

ALGÈBRE :

$$\pi_{DEPTNO}(\sigma_{LOC=\text{"}Paris\text{"}}(DEPARTEMENT)) \cap \pi_{DEPTNO}(\sigma_{SAL>20000}(EMPLOYE))$$





SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  LOC = 'Paris'
INTERSECT
SELECT DEPTNO
FROM   EMPLOYE
WHERE  SAL > 20000
```

**NOTE:** L'intersection élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `INTERSECT ALL` : le résultat contient chaque n-uplet $min(a, b)$ fois, où $a$ et $b$ est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.





## Requêtes imbriquées simples

La Jointure s'exprime par deux blocs SFW imbriqués

Soit le schéma de relations
**COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Nom, prix et fournisseurs des Produits commandés par Jean*
ALGÈBRE:
$\pi_{PNOM,PRIX,FNOM}(\sigma_{CNOM=\text{"}JEAN\text{"}}(COMMANDES) \bowtie (FOURNITURE))$

SQL:

```
SELECT PNOM,PRIX,FNOM
FROM   FOURNITURE
WHERE  PNOM IN (SELECT PNOM
                FROM   COMMANDES
                WHERE  CNOM = 'JEAN')
```

ou

```
SELECT DISTINCT FOURNITURE.PNOM,PRIX,FNOM
FROM   FOURNITURE,COMMANDES
WHERE  FOURNITURE.PNOM = COMMANDES.PNOM
AND    CNOM = ''JEAN''
```





La Différence s'exprime aussi par deux blocs SFW imbriqués

Soit le schéma de relations
**EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)
**DEPARTEMENT**(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
ALGÈBRE :

$$\pi_{DEPTNO}(DEPARTEMENT) - \pi_{DEPTNO}(EMPLOYE)$$





SQL:

```
  SELECT DEPTNO
  FROM   DEPARTEMENT
  WHERE  DEPTNO NOT IN (SELECT DEPTNO FROM   EMPLOYE)
```

ou

```
  SELECT DEPTNO
  FROM DEPARTEMENT
  EXCEPT
  SELECT DEPTNO
  FROM EMPLOYE
```





## Requêtes imbriquées plus complexes : ANY - ALL

SCHÉMA: **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs des briques à un coût inférieur au coût maximum des ardoises*

```
SQL :    SELECT FNOM
         FROM   FOURNITURE
         WHERE  PNOM = 'Brique'
         AND    PRIX < ANY (SELECT PRIX
                            FROM   FOURNITURE
                            WHERE  PNOM = 'Ardoise')
```

La condition $f\ \theta$ ANY (SELECT ... FROM ...) est vraie ssi la comparaison $f\theta v$ est vraie au moins pour une valeur $v$ du résultat du bloc (SELECT F FROM ...).

## "IN" et "= ANY"

**COMMANDE**(NUM,CNOM,PNOM,QUANTITE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Nom, prix et fournisseur des produits commandés par Jean*
SQL:

```
SELECT PNOM, PRIX, FNOM
FROM   FOURNITURE
WHERE  PNOM = ANY (SELECT PNOM
                   FROM   COMMANDE
                   WHERE  CNOM = 'JEAN')
```

**NOTE:** Les prédicats IN et = ANY sont utilisés de façon équivalente.





## ALL

SCHÉMA: **COMMANDE**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Client ayant commandé la plus petite quantité de briques*
SQL:

```
SELECT CNOM
FROM   COMMANDE
WHERE  PNOM = 'Brique' AND
       QUANTITE <= ALL (SELECT QUANTITE
                        FROM   COMMANDE
                        WHERE  PNOM = 'Brique')
```

La condition $f\ \theta$ ALL (SELECT ... FROM ...) est vraie ssi la comparaison $f\theta v$ est vraie pour toutes les valeurs $v$ du résultat du bloc (SELECT ... FROM ...).





## "NOT IN" et "NOT = ALL"

**EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)
**DEPARTEMENT**(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  DEPTNO NOT = ALL (SELECT DEPTNO
                         FROM   EMPLOYE)
```

**NOTE:** Les prédicats NOT IN et NOT = ALL sont utilisés de façon équivalente.





## EXISTS

**FOURNISSEUR**(FNOM,STATUS,VILLE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs qui fournissent au moins un produit*
SQL :

```
SELECT FNOM
  FROM FOURNISSEUR
 WHERE EXISTS (SELECT *
                 FROM FOURNITURE
                WHERE FNOM = FOURNISSEUR.FNOM)
```

La condition EXISTS (SELECT * FROM ...) est vraie ssi le résultat du bloc (SELECT F FROM ...) n'est pas vide.

## NOT EXISTS

**FOURNISSEUR**(FNOM,STATUS,VILLE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs qui ne fournissent aucun produit*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS (SELECT *
                   FROM   FOURNITURE
                   WHERE  FNOM = FOURNISSEUR.FNOM)
```

La condition NOT EXISTS (SELECT * FROM ...) est vraie ssi le résultat du bloc (SELECT F FROM ...) est vide.





## Formes équivalentes de quantification

Si $\theta$ est un des opérateurs de comparaison $<, =, >, \ldots$

- La condition
  x $\theta$ ANY (SELECT Ri.y FROM R1, ... Rn WHERE p)
  est équivalente à
  EXISTS (SELECT * FROM R1, ... Rn WHERE p AND x $\theta$ Ri.y)
- La condition
  x $\theta$ ALL (SELECT Ri.y FROM R1, ... Rn WHERE p)
  est équivalente à

NOT EXISTS (SELECT * FROM R1, ... Rn WHERE (p) AND NOT (x $\theta$ Ri.y))





## Exemple : "EXISTS" et "= ANY"

**COMMANDE**(NUM,CNOM,PNOM,QUANTITE)
**FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Nom, prix et fournisseur des produits commandés par Jean*

```
SELECT PNOM, PRIX, FNOM  FROM   FOURNITURE
WHERE  EXISTS  (SELECT * FROM   COMMANDE
                WHERE CNOM = 'JEAN'
                  AND PNOM = FOURNITURE.PNOM)
```

ou

```
SELECT PNOM, PRIX, FNOM FROM   FOURNITURE
WHERE  PNOM = ANY (SELECT PNOM  FROM   COMMANDE
                   WHERE  CNOM = 'JEAN')
```





## Encore plus compliqué...

SCHÉMA: **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs qui fournissent au moins un produit avec un coût supérieur au coût de tous les produits fournis par Jean*

```
SELECT DISTINCT P1.FNOM
FROM   FOURNITURE P1
WHERE  NOT EXISTS  (SELECT * FROM   FOURNITURE P2
                    WHERE  P2.FNOM = 'JEAN'
                    AND    P1.PRIX <= P2.PRIX)
```

ou

```
SELECT DISTINCT FNOM FROM   FOURNITURE
WHERE  PRIX > ALL (SELECT PRIX FROM   FOURNITURE
                   WHERE FNOM = 'JEAN')
```

## Et la division?

**FOURNITURE**(FNUM,PNUM,QUANTITE)
**PRODUIT**(PNUM,PNOM,PRIX)
**FOURNISSEUR**(FNUM,FNOM,STATUS,VILLE)
REQUÊTE: *Noms des fournisseurs qui fournissent tous les produits*
ALGÈBRE:

$$R1 := \pi_{FNUM,PNUM}(FOURNITURE) \div \pi_{PNUM}(PRODUIT)$$
$$R2 := \pi_{FNOM}(FOURNISSEUR \bowtie R1)$$





SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS
       (SELECT *
        FROM  PRODUIT
        WHERE NOT EXISTS
              (SELECT *
               FROM  FOURNITURE
               WHERE FOURNITURE.FNUM = FOURNISSEUR.FNUM
               AND   FOURNITURE.PNUM = PRODUIT.PNUM))
```





## COUNT, SUM, AVG, MIN, MAX

REQUÊTE: *Nombre de fournisseurs parisiens*

```
SELECT COUNT(*)
FROM   FOURNISSEUR
WHERE  VILLE = 'Paris'
```

REQUÊTE: *Nombre de fournisseurs qui fournissent des produits*

```
SELECT COUNT(DISTINCT FNOM)
FROM   FOURNITURE
```

**NOTE:** La fonction COUNT(*) compte le nombre des *n*-uplets du résultat d'une requête sans élimination des dupliqués ni vérification des valeurs nulles. Dans le cas contraire on utilise la clause COUNT(DISTINCT ...).





## SUM et AVG

REQUÊTE: *Quantité totale de Briques commandées*

```
SELECT SUM (QUANTITE)
FROM   COMMANDES
WHERE  PNOM = 'Brique'
```

REQUÊTE: *Coût moyen de Briques fournies*

```
SELECT AVG (PRIX)              SELECT SUM (PRIX)/COUNT(PRIX)
FROM   FOURNITURE       ou     FROM FOURNITURE
WHERE  PNOM = 'Brique'         WHERE PNOM = 'Brique'
```

## MIN et MAX

REQUÊTE: *Le prix des briques le moins chères.*

```
SELECT MIN(PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Briques';
```

REQUÊTE: *Le prix des briques le plus chères.*

```
SELECT MAX(PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Briques';
```

Comment peut-on faire sans MIN et MAX?





## Requête imbriquée avec fonction de calcul

REQUÊTE: *Fournisseurs de briques dont le prix est en dessous du prix moyen*

```
SELECT FNOM
FROM   FOURNITURE
WHERE  PNOM = 'Brique' AND
       PRIX < (SELECT AVG(PRIX)
               FROM   FOURNITURE
               WHERE PNOM = 'Brique')
```





## GROUP BY

REQUÊTE: *Nombre de fournisseurs* ***par ville***

| VILLE | FNOM |
|---|---|
| PARIS | TOTO |
| PARIS | DUPOND |
| LYON | DURAND |
| LYON | LUCIEN |
| LYON | REMI |

| VILLE | COUNT(FNOM) |
|---|---|
| PARIS | 2 |
| LYON | 3 |

```
SELECT VILLE, COUNT(FNOM) FROM FOURNISSEUR GROUP BY VILLE
```

**NOTE:** La clause GROUP BY permet de préciser les attributs de partitionnement des relations declarées dans la clause FROM.





REQUÊTE: *Donner pour chaque produit son prix moyen*

```
SELECT PNOM, AVG (PRIX)
FROM   FOURNITURE
GROUP BY PNOM
```

RÉSULTAT:

| PNOM | AVG (PRIX) |
|---|---|
| BRIQUE | 10.5 |
| ARDOISE | 9.8 |

**NOTE:** Les fonctions de calcul appliquées au résultat de régroupement sont directement indiquées dans la clause SELECT: le calcul de la moyenne se fait par produit obtenu au résultat après le regroupement.

## HAVING

REQUÊTE: *Produits fournis par deux ou plusieurs fournisseurs avec un prix supérieur à 100 Euros*

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX > 100
GROUP BY PNOM
HAVING COUNT(*) >= 2
```





## HAVING

AVANT LA CLAUSE HAVING

| PNOM | FNOM | PRIX |
|---|---|---|
| BRIQUE | TOTO | 105 |
| ARDOISE | LUCIEN | 110 |
| ARDOISE | DURAND | 120 |

APRÈS LA CLAUSE HAVING

| PNOM | FNOM | PRIX |
|---|---|---|
| ARDOISE | LUCIEN | 110 |
| ARDOISE | DURAND | 120 |

**NOTE:** La clause HAVING permet d'éliminer des partitionnements, comme la clause WHERE élimine des *n*-uplets du résultat d'une requête: on garde les produits dont le nombre des fournisseurs est $>= 2$.
Des conditions de sélection peuvent être appliquées avant le calcul d'agrégat (clause WHERE) mais aussi après (clause HAVING).





REQUÊTE: *Nom et prix moyen des produits fournis par des fournisseurs Parisiens et dont le prix minimum est supérieur à 1000 Euros*

```
SELECT PNOM, AVG(PRIX)
FROM   FOURNITURE, FOURNISSEUR
WHERE  VILLE = 'Paris' AND
       FOURNITURE.FNOM = FOURNISSEUR.FNOM
GROUP BY PNOM
HAVING MIN(PRIX) > 1000
```





## ORDER BY

En général, le résultat d'une requête SQL n'est pas trié. Pour trier le résultat par rapport aux valeurs d'un ou de plusieurs attributs, on utilise la clause `ORDER BY` :

```
SELECT VILLE, FNOM, PNOM
  FROM FOURNITURE, FOURNISSEUR
 WHERE FOURNITURE.FNOM = FOURNISSEUR.FNOM
 ORDER BY VILLE, FNOM DESC
```

Le résultat est trié par les villes (ASC) et le noms des fournisseur dans l'ordre inverse (DESC).

## Historique

**SQL86 - SQL89 ou SQL1** La référence de base:

- Requêtes compilées puis exécutées depuis un programme d'application.
- Types de données simples (entiers, réels, chaînes de caractères de taille fixe)
- Opérations ensemblistes restreintes (UNION).

**SQL91 ou SQL2** Standard actuel:

- Requêtes dynamiques
- Types de données plus riches (intervalles, dates, chaînes de caractères de taille variable)
- Différents types de jointures: jointure naturelle, jointure externe
- Opérations ensemblistes: différence (EXCEPT), intersection (INTERSECT)
- Renommage des attributs dans la clause SELECT





## Historique

**SQL:1999 (SQL3)** : SQL devient un langage de programmation :

- Extensions orientées-objet (héritage, méthodes)
- Types structurés
- BLOB, CLOB
- Opérateur de fermeture transitive (recursion)





## Récursivité dans SQL

schéma **ENFANT**(NOMPARENT,NOMENFANT)
REQUÊTE: *Les enfants de Charlemagne*
SQL:

```
SELECT NOMENFANT
  FROM ENFANT
 WHERE NOMPARENT='Charlemagne';
```





## Récursivité dans SQL

schéma **ENFANT**(NOMPARENT,NOMENFANT).
REQUÊTE: *Les* **enfants et petits-enfants** *de Charlemagne*
SQL:

```
(SELECT NOMENFANT
  FROM ENFANT
 WHERE NOMPARENT='Charlemagne')

 UNION

(SELECT E2.NOMENFANT
 FROM ENFANT E1,E2
 WHERE E1.NOMPARENT='Charlemagne'
 AND
 E1.NOMENFANT=E2.NOMPARENT)
```

## Descendants

schéma **ENFANT**(NOMPARENT,NOMENFANT).
REQUÊTE: *Les* **descendants** *de Charlemagne*

- Nécéssite un nombre a priori inconnu de jointures
- Th : impossible à exprimer en logique du premier ordre
- Th : donc, impossible en algèbre relationnel

En pratique, on étend le langage SQL avec des opérateurs récursifs





## Descendants

schéma **ENFANT**(NOMPARENT,NOMENFANT)
REQUÊTE: *Les* **descendants** *de Charlemagne*
SQL:

```
WITH RECURSIVE DESCENDANT(NOMANC, NOMDESC) AS
      (SELECT NOMPARENT, NOMENFANT FROM ENFANT)
  UNION
      (SELECT R1.NOMANC, R2.NOMDESC
         FROM DESCENDANT R1, DESCENDANT R2
        WHERE R1.NOMDESC=R2.NOMANC)
SELECT NOMDESC FROM DESCENDANT
 WHERE NOMANC='Charlemagne';
```





## Création de tables

Une table (relation) est créée avec la commande **CREATE TABLE** :

```
CREATE TABLE Produit (pnom VARCHAR(20),
                      prix INTEGER);

CREATE TABLE Fournisseur(fnom VARCHAR(20),
                         ville VARCHAR(16));
```

- Pour chaque attribut, on indique le domaine (type)





## Nombreux types : exemple d'Oracle 8i

| | |
|---|---|
| `decimal(p, s)` | <p digits>,<s digits> |
| `integer` | nombre entier |
| `real` | nombre réel |
| `char (size)` | chaîne de caractère de taille fixe |
| `varchar (size)` | chaîne de caractère de taille variable |
| `date,timestamp` | horodatage |
| `boolean` | booléen |
| `blob` | données binaires de grande taille |
| etc. | |

## Contraintes d'intégrité

Pour une application donnée, pour un schéma relationnel donné, toutes les instances ne sont pas significatives

**Exemple**

- Champs important non renseigné : autorisation des NULL
- Prix négatifs : contrainte d'intégrité sémantique
- Code de produit dans une commande ne correspondant à aucun produit dans le catalogue : contrainte d'intégrité référentielle





## Valeurs NULL

La valeur NULL peut être interdite :

```
CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL
)
```





## Unicité des valeurs

Interdiction de deux valeurs identiques pour le même attribut :

```
CREATE TABLE Fourniture (pnom VARCHAR(20) UNIQUE,
                         fnom VARCHAR(20)
)
```

- UNIQUE et NOT NULL : l'attribut peut servir de clé primaire





## Ajout de contraintes référentielles : clés primaires

```
CREATE TABLE Produit (pnom VARCHAR(20),
                      prix INTEGER,
                      PRIMARY KEY (pnom));

CREATE TABLE Fournisseur(fnom VARCHAR(20) PRIMARY KEY,
                         ville VARCHAR(16));
```

- L'attribut pnom est une clé dans la table Produit
- L'attribut fnom est une clé dans la table Fournisseur
- Une seule clé primaire par relation
- Une clé primaire peut être référencée par une autre relation

## Ajout de contraintes référentielles : clés étrangères

La table `Fourniture` relie les produits à leurs fournisseurs :

```
CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL,
       FOREIGN KEY (pnom) REFERENCES Produit,
       FOREIGN KEY (fnom) REFERENCES Fournisseur);
```

- Les attributs `pnom` et `fnom` sont des clés étrangères (`pnom` et `fnom` existent dans les tables référencées)
- Pour sélectionner un attribut de nom différent :

```
FOREIGN KEY (pnom) REFERENCES Produit(autrenom)
```





## Valeurs par défaut

```
CREATE TABLE Fournisseur(fnom VARCHAR(20),
                ville VARCHAR(16) DEFAULT 'Carcassonne');
```

- Valeur utilisée lorsque l'attribut n'est pas renseigné
- Sans précision, la valeur par défaut est `NULL`





## Contraintes sémantiques

- Clause `CHECK`, suivie d'une condition

**Exemple : prix positifs**

```
prix INTEGER CHECK (prix>0)
```

- Condition générale : requête booléenne (dépend du SGBD)





## Destruction de tables

On détruit une table avec la commande **DROP TABLE** :

```
DROP TABLE Fourniture;
DROP TABLE Produit;
DROP TABLE Fournisseur;
```

La table `Fourniture` doit être détruite en premier car elle contient des clés étrangères vers les deux autres tables;

## Insertion de n-uplets

On insère dans une table avec la commande **INSERT** :

**INSERT INTO** $R(A_1, A_2, \ldots, A_n)$ **VALUES** $(v_1, v_2, \ldots v_n)$

Donc on donne deux listes : celles des attributs (les $A_i$) de la table et celle des valeurs respectives de chaque attribut (les $v_i$).

1. Bien entendu, chaque $A_i$ doit être un attribut de $R$
2. Les attributs non-indiqués restent à **NULL** ou à leur valeur par défaut.
3. On doit toujours indiquer une valeur pour un attribut déclaré **NOT NULL**





## Insertion : exemples

Insertion d'une ligne dans *Produit* :

**INSERT INTO** *Produit (pnom, prix)*
**VALUES** (*'Ojax'*, *15*)

Insertion de deux fournisseurs :

**INSERT INTO** *Fournisseur (fnom, ville)*
**VALUES** (*'BHV'*, *'Paris'*), (*'Casto'*, *'Paris'*)

Il est possible d'insérer plusieurs lignes en utilisant **SELECT**

**INSERT INTO** *NomsProd (pnom)*
**SELECT DISTINCT** *pnom* **FROM** *Produit*





## Modification

On modifie une table avec la commande **UPDATE** :

**UPDATE** $R$ **SET** $A_1 = v_1, A_2 = v_2, \ldots, A_n = v_n$
**WHERE** *condition*

Contrairement à **INSERT**, **UPDATE** s'applique à un ensemble de lignes.

1. On énumère les attributs que l'on veut modifier.
2. On indique à chaque fois la nouvelle valeur.
3. La clause **WHERE** *condition* permet de spécifier les lignes auxquelles s'applique la mise à jour. Elle est identique au **WHERE** du **SELECT**

Bien entendu, on ne peut pas violer les contraintes sur la table.





## Modification : exemples

Mise à jour du prix d'Ojax :

**UPDATE** *Produit* **SET** *prix=17*
**WHERE** *pnom = 'Ojax'*

Augmenter les prix de tous les produits fournis par BHV par 20% :

**UPDATE** *Produit* **SET** *prix = prix*1.2*
**WHERE** *pnom* in (**SELECT** *pnom*
**FROM** *Fourniture*
**WHERE** *fnom = 'BHV'*)

## Destruction

On détruit une ou plusieurs lignes dans une table avec la commande **DELETE** :

**DELETE FROM** *R*
**WHERE** *condition*

C'est la plus simple des commandes de mise-à-jour puisque elle s'applique à des lignes et pas à des attributs. Comme précédemment, la clause **WHERE** *condition* est indentique au **WHERE** du **SELECT**





## Destruction : exemples

Destruction des produits fournis par le BHV :

**DELETE FROM** *Produit*
**WHERE** *pnom* in (**SELECT** *pnom*
**FROM** *Fourniture*
**WHERE** *fnom* = *'BHV*)

Destruction du fournisseur BHV :

**DELETE FROM** *Fournisseur*
**WHERE** *fnom* = *'BHV'*





## Déclencheurs associés aux destructions de *n*-uplets

- Que faire lorsque le *n*-uplet référence une autre table ?

```
CREATE TABLE Produit (pnom VARCHAR(20),
                      prix INTEGER,
                      PRIMARY KEY (pnom));

CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL,
       FOREIGN KEY (pnom) REFERENCES Produit
       on delete <action>);
```

<action> à effectuer lors de la destruction dans Produit :

- CASCADE : destruction si destruction dans Produit
- RESTRICT : interdiction si existe dans Fourniture
- SET NULL : remplacer par NULL
- SET DEFAULT <valeur> : remplacement par une valeur par défaut





## Déclencheurs associés aux mise à jour de *n*-uplets

```
CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
fnom VARCHAR(20) NOT NULL,
FOREIGN KEY (pnom) REFERENCES Produit
on update <action>);
```

<action> à effectuer lors d'un changement de clé dans Produit :

- CASCADE : propagation du changement de clé de Produit
- RESTRICT : interdiction si clé utilisée dans Fourniture
- SET NULL : remplace la clé dans Fourniture par NULL
- SET DEFAULT <valeur> : remplace la clé par une valeur par défaut