Le langage SQL

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille

Février 2016

- **Historique**
- **Opérations de base de l'algèbre relationnelle : SELECT,…**
  Projection, Sélection, Jointures, Opérations ensemblistes, Division, …
- **Imbrication de requêtes :** simples, ANY, ALL, EXIST**, …**
- **Fonctions de calcul :** COUNT, AVG, SUM, MAX, MIN, …
- **Opérations de mises à jour :** INSERT, UPDATE, DELETE
- **Opérations additionnelles :** création de tables, vues, administration des tables, …

Plan

Historique

Origine : SQL (Structured Query Language), langage de requêtes standard pour les SGBD relationnels est dérivé de SEQUEL, interface de consultation pour System R (Astr IBM 81)

3 niveaux de normes :

SQL86 (standard ANSI en 86 puis ISO en 87) **: la base puis SQL89 ou SQL1 : l'intégrité**:

- 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: exécution différée ou immédiate
- 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

SQL3 (98) : SQL devient un langage de programmation et évolue vers l'objet

- Extensions orientées-objet
- Opérateur de fermeture transitive (recursion), …

1 - Opérations de base associées à l'algèbre relationnelle : SELECT, …

- **Projection**
- **Sélection :** condition de sélection, valeur NULL,
- **Jointure :** produit cartésien, jointure naturelle, théta-jointure, jointure externe
- **Opérations ensemblistes :** union, différence, intersection
- **Division, …**

Opérations de base : SELECT - Principe général

- **structure de base d'une expression SQL SELECT** comporte **3 clauses** :
  - **SELECT** <liste des attributs a projeter> : projection de l'algèbre relationnelle désigne la liste des **attributs désirés**
  - **FROM** <liste des tables arguments> : liste des **tables à considérer**
  - **WHERE** <conditions sur un ou plusieurs attributs> : **prédicat** à vérifier sur des attributs de tables de la clause FROM
- **requêtes simples : correspondance avec l'algèbre immédiate:**

```
SELECT a1, a2,…, an
FROM   R1, R2,…,Rm
WHERE  P
```

Soit :

$$\pi_{a1,\ a2,\ldots,\ an}\ \sigma_{P}\ (R1xR2x\ldots xRm)$$

Base de données exemple 1

- **COMMANDES**(**CNUM**, CNOM, *PNUM*, QUANTITE) :
  - **CNUM** *: N° commande*
  - *CNOM : nom client*
  - *PNUM = N° Produit = clé étrangère vers PRODUIT*
  - *QUANTITE : quantité commandée*
- **PRODUIT**(**PNUM**, PNOM, PRIXV)
  - **PNUM** *: N° produit = clé primaire PRODUIT*
  - *PNOM : nom produit*
  - *PRIXV : prix vente produit*
- **FOURNISSEUR**(**FNUM**, FNOM, STATUT, VILLE)
  - **FNUM** *: N° fournisseur = Clé primaire table FOURNISSEUR*
  - *FNOM : nom fournisseur*
  - *STATUT : statut fournisseur*
  - *VILLE : ville fournisseur*
- **FOURNITURE**(***PNUM, FNUM***, PRIXA) :
  - *PNUM : N° produit clé = étrangère vers PRODUIT*
  - *FNUM : N° fournisseur = clé étrangère vers FOURNISSEUR*
  - ***PNUM, FNUM*** *= clé primaire composée de FOURNITURE*
  - *PRIXA: prix d'achat;*

Base de données exemple 2

- **DEPT**(**DEPNO**, DNOM, LOC)
  - **DEPTNO** *: N° département = clé primaire table DEPT*
  - *DNOM : nom département*
  - *LOC: localisation*
- **EMPLOYE**(**EMPNO**, ENOM, *DEPTNO*, SAL)
  - **EMPNO** *: N° employé = clé primaire table EMPLOYE*
  - *ENOM : nom employé*
  - *DEPNO = clé étrangère vers table DEPT*
  - *DEPTNO : N° département*
  - *SAL : Salaire de l'employé*

Projection

Soit le schéma de table **COMMANDES**(**CNUM**,CNOM,*PNUM*,QUANTITE)

- **Requête:** ***Informations sur toutes les commandes***

SQL:

```
SELECT CNUM,CNOM,PNUM,QUANTITE
FROM COMMANDES
```

ou

```
SELECT *
FROM COMMANDES
```

- **Requête: N° des produits commandés**

```
SELECT PNUM
FROM COMMANDES
```

Remarque :

Contrairement à l'algèbre relationnelle, SQL n'élimine pas les doublons.

Pour les éliminer on utilise DISTINCT :

```
SELECT DISTINCT PNOM
FROM COMMANDES
```

Le **DISTINCT** peut être remplacé par la clause **UNIQUE** dans certains systèmes

Sélection

Soit la table **COMMANDES**(**CNUM**,CNOM,*PNUM*,QUANTITE)

- **Requête: N° des produits commandés par Paul**

Algèbre:

$$\pi_{PNOM}\ (\sigma_{CNOM= \text{'PAUL'}}\ (COMMANDES))$$

SQL :

```
SELECT PNUM
FROM COMMANDES
WHERE CNOM = 'PAUL'
```

Conditions de sélection :

- **=, <, >, <>,...**
- **BETWEEN, LIKE**
- **IS NULL, IS NOT NULL, IN**

Conditions de sélection simples : =, <, >, <>,...

Les conditions de base sont exprimées de 2 façons:

- `attribut comparateur valeur`
- `attribut comparateur attribut`

où le comparateur est : =, <, >, < >, …

Soit la table : **FOURNITURE**(*PNUM, FNUM*, PRIXA)

- **Requête: Produits de prix d'achat est supérieur à 2000€**

SQL:

```
SELECT PNUM
FROM FOURNITURE
WHERE PRIXA > 2000
```


Conditions de sélection : BETWEEN, LIKE

Le comparateur est : **BETWEEN, LIKE, IS NULL, IN**

Table FOURNITURE (***PNUM, FNUM***, PRIXA)

- **Requête: Produits avec un prix d'achat entre 1000€ et 2000€**

SQL:

```
SELECT PNUM
FROM FOURNITURE
WHERE PRIXA BETWEEN 1000 AND 2000
```

Remarque: La condition y BETWEEN x AND z est équivalente à y <= z AND x <= y

Table COMMANDES (CNUM, CNOM, *PNUM*, QUANTITE)

- **Requête: Clients dont le nom commence par "C"**

SQL:

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

Remarque:

le littéral qui suit **LIKE**, pas exprimable avec l'algèbre relationnelle, doit être une chaîne de caractères éventuellement avec des caractères jokers (_, %).

Rappel sur la valeur NULL

La valeur NULL est une valeur "spéciale" représentant une *valeur (information) inconnue* :

1. *A* θ *B* est inconnu (ni vrai, ni faux) si la valeur de *A* ou/et *B* est NULL (θ 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 : +, -, *, /).

Table FOURNISSEUR (FNUM, FNOM, STATUT, VILLE)

- **Requête: Nom des fournisseurs de Paris.**

SQL:

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

Remarque : On ne trouve pas les fournisseurs avec VILLE = NULL !

Conditions de sélection: IS NULL, IS NOT NULL, IN

Table FOURNISSEUR (**FNUM**, FNOM, STATUT, VILLE)

- **Requête: nom des fournisseurs dont l'adresse est inconnu.**

SQL:

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

Remarque : Le prédicat IS NULL (ou IS NOT NULL) n'est pas exprimable en algèbre relationnelle.

Table FOURNITURE (***PNUM, FNUM***, PRIXA)

- **Requête: numéro des produits avec un prix d'achat de 100€, de 200€ ou de 300€**

SQL:

```
SELECT PNUM
FROM FOURNITURE
WHERE PRIXA IN {100,200,300}
```

Remarque : La condition *x* IN {*a, b, …, z*} est équivalente à *a* OR *x* = *b* OR …OR *x* = *z*.

Jointure simple en SQL de base

COMMANDES (**CNUM**, CNOM, *PNUM*, QUANTITE)
FOURNITURE (***PNUM, FNUM***, PRIXA)

- **Requête: N°, prix d'achat, N° des fournisseurs, des Produits commandés par Paul**

Algèbre : $\pi_{PNOM, PRIX, FNOM}$ ($\sigma_{CNOM= 'PAUL'}$ (COMMANDES) $\bowtie$ (FOURNITURE))

En SQL de base :

```
SELECT COMMANDES.PNUM, PRIXA, FNUM
FROM COMMANDES, FOURNITURE
WHERE CNOM = 'PAUL' AND COMMANDES.PNUM = FOURNITURE.PNUM
```

En utilisant les alias :

```
SELECT COMMANDES.PNUM, PRIXA, FNUM
FROM COMMANDES C, FOURNITURE F
WHERE CNOM = 'PAUL' AND C.PNUM = F.PNUM
```

Remarques :

1- Cette requête est équivalente à une jointure naturelle. Les attributs de jointure sont explicités.

2- `SELECT COMMANDES.PNUM, PRIXA, FNUM FROM COMMANDES, FOURNITURE` équivaut à un **produit cartésien** des 2 tables, suivi d'une projection.

Jointures dans SQL2

Opérations de Jointure :

SQL2	Opération	Algèbre
`R1 CROSS JOIN R2`	produit cartesien	R1 X R2
`R1 JOIN R2 ON prédicat` (`R1 JOIN R2 ON R1.A<R2.B`)	théta-jointure	R1 $\bowtie_{R1.A<R2.B}$ R2
`R1 FULL/LEFT/RIGHT OUTER JOIN R2 ON prédicat`	théta-jointure externes	R1 $\bowtie_{R1.A<R2.B}$ R2 R1 $\bowtie_{R1.A<R2.B}$ R2 R1 $\bowtie_{R1.A<R2.B}$ R2
`R1 NATURAL JOIN R2`	jointure naturelle	R1 $\bowtie$ R2
`R1 NATURAL FULL/LEFT/RIGHT OUTER JOIN R2`	jointure externes	R1 $\bowtie$ R2 R1 $\bowtie$ R2 R1 $\bowtie$ R2

Jointure en SQL2 : « INNER JOINT »

COMMANDES (**CNUM**, CNOM, *PNUM*, QUANTITE)
FOURNITURE (***PNUM, FNUM***, PRIXA)

- **Requête: N°, prix d'achat, N°fournisseurs des Produits commandés par Paul**

En SQL2 :

```
SELECT COMMANDES.PNUM, PRIXA, FNUM
FROM COMMANDES INNER JOINT FOURNITURE
ON COMMANDES.PNUM = FOURNITURE.PNUM
WHERE CNOM = 'PAUL'
```

Remarques :

- `INNER` est facultatif dans la plupart des SGBDR
- Cette notation rend plus lisible la requête en distinguant clairement les conditions de jointures, derrière `ON`, et les éventuelles conditions de sélection ou restriction, derrière `WHERE`.
- L'oubli du `ON` empêche l'exécution de la requête (évite de lancer un couteux produit cartésien en SQL de base)

Jointure naturelle

Exemple de jointure naturelle :

Table EMP :

EMPNO	*DEPNO*	SAL
Tom	1	10000
Jim	2	20000
Karim	3	15000

Table DEPT :

DEPNO	DNOM	LOC
1	Commercial	Marseille
2	Administration	Paris
4	Technique	Paris

Jointure naturelle : les tuples qui ne peuvent pas être joints sont éliminés :

EMPLOYE ⋈ DEPT

Résultat :

EMPNO	DEPNO	SAL	DNOM	LOC
Tom	1	10000	Commercial	Marseille
Jim	2	20000	Administration	Paris

Jointure en SQL2 : « NATURAL JOINT » (1)

Tables :

EMPLOYE (**EMPNO**, ENOM, *DEPTNO*, SAL)
DEPT (**DEPNO**, DNOM, LOC)

EMPLOYE ⋈ DEPT

- **Requête: Noms des départements avec les noms de leurs employés :**

En SQL de base :

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

Remarques :

1. Comme en algèbre DEPT NATURAL JOIN EMP fait la jointure naturelle (sur l'attribut DEPNO)
2. l'attribut DEPNO n'apparaît qu'une **seule fois** dans la table résultat.

Jointure en SQL2 : « NATURAL JOINT » (2)

Tables :
COMMANDES (**CNUM**, CNOM, *PNUM*, QUANTITE)
FOURNITURE (***PNUM, FNUM***, PRIXA)

- **Requête: N°, prix d'achat, N°fournisseurs des Produits commandés par Paul**

En SQL de base :

```
SELECT COMMANDES.PNUM, PRIXA, FNOM
FROM COMMANDES, FOURNITURE
WHERE CNOM = 'PAUL' AND COMMANDES.PNUM = FOURNITURE.PNUM
```

On a une **jointure naturelle** car les attributs de jointure ont le même nom

En SQL2 :

```
SELECT COMMANDES.PNUM, PRIXA, FNUM
FROM COMMANDES NATURAL JOINT FOURNITURE
WHERE CNOM = 'PAUL'
```

Remarque :

- Il est possible de restreindre ou préciser le ou les attributs de jointure avec `USING PNOM`

Auto-Jointure (1)

Table EMPLOYE (**EMPNO**, ENOM, *DEPTNO*, SAL)

- **Requête: Nom et Salaire des Employés gagnant plus que l'employé de numéro 12546**

Algèbre :

$$R1 := \pi_{SAL} (\sigma_{EMPNO=12546} (EMPLOYE)$$

$$R2 := \pi_{ENOM,\ EMPLOYE.SAL} (EMPLOYE) \bowtie_{((EMPLOYE.SAL>R1.SAL)} (R1))$$

En SQL de base :

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

Remarque :

`E1` et `E2` sont 2 instances différentes de la table EMPLOYE

Auto-Jointure (2)

Table FOURNISSEUR (**FNUM**, FNOM, STATUT, VILLE)

- **Requête: Nom des fournisseurs qui habitent deux à deux dans la même ville**

En SQL de base :

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

Remarques :

1. **PREM** et **SECOND** sont 2 instances différentes de FOURNISSEUR
2. la 2° condition permet d'éliminer les paires (x,x) et éviter d'obtenir à la fois (x,y) et (y,x)

Jointure : valeurs de Vérité sur les conditions

Trois valeurs de vérité: vrai, faux et inconnu :

- **vrai AND inconnu = inconnu**
- **faux AND inconnu = faux**
- **inconnu AND inconnu = inconnu**
- **vrai OR inconnu = vrai**
- **faux OR inconnu = inconnu**
- **inconnu OR inconnu = inconnu**
- **NOT inconnu = inconnu**

Exemple :

Table EMPLOYE (**EMPNO**, ENOM, *DEPTNO*, SAL)

```
SELECT E1.ENOM
FROM EMPLOYE E1, EMPLOYE E2
WHERE E1.SAL > 20000 OR
E1.SAL <= 20000
```

Trouve-t-on les noms de tous les employés s'il y a des employés avec un salaire inconnu ?

Theta-jointure : JOINT … ON prédicat

Table EMPLOYE(**EMPNO**, ENOM, *DEPNO*, SAL)

EMPLOYE1 ⋈ [E1.SAL > E2.SAL] EMPLOYE2

- **Requête: Nom et salaire des employés gagnant plus que l'employé 12546 (AUTOJOINTURE)**

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


Jointure externe pleine : *FULL* OUTER JOINT

Table EMP :

EMPNO	*DEPNO*	SAL
Tom	1	10000
Jim	2	20000
Karim	3	15000

Table DEPT ::

DEPNO	DNOM	LOC
1	Commercial	Marseille
2	Administration	Paris
4	Technique	Paris

- les tuples qui ne peuvent pas être joints ***ne sont pas éliminés.***
- On garde tous les tuples des 2 tables

EMP ⟗ DEPT

SQL:

```
EMP NATURAL FULL OUTER JOIN DEPT
```

Résultat :

EMPNO	DEPNO	SAL	DNOM	LOC
Tom	1	10000	Commercial.	Marseille
Jim	2	20000	Administration	Paris
Karim	3	15000	**NULL**	**NULL**
NULL	4	**NULL**	Technique.	Paris

Jointure externe droite/gauche : *LEFT/RIGHT OUTER JOIN*

Table EMP :

EMPNO	***DEPNO***	**SAL**
Tom	1	10000
Jim	2	20000
Karim	3	15000

Table DEPT ::

DEPNO	**DNOM**	**LOC**
1	Commercial	Marseille
2	Administration	Paris
4	Technique	Paris

- **Jointure externe gauche:** On garde tous les n-uplets de la première table (gauche) :

```
EMP NATURAL LEFT OUTER JOIN DEPT
```

EMPNO	**DEPNO**	**SAL**	**DNOM**	**LOC**
Tom	1	10000	Commercial	Marseille
Jim	2	20000	Administration	Paris
Karim	3	15000	**NULL**.	**NULL**.

- **Jointure externe droite:** On garde tous les n-uplets de la deuxième table (droite) **:**

```
EMP NATURAL RIGHT OUTER JOIN DEPT
```

EMPNO	**DEPNO**	**SAL**	**DNOM**	**LOC**
Tom	1	10000	Commercial	Marseille
Jim	2	20000	Administration	Paris
NULL	4	**NULL**	Technique	Paris

Jointures et théta-jointures externes dans SQL2

- **Jointures (équi-jointures) externes :**
  - R1 **NATURAL FULL OUTER JOIN** R2 **:** Remplir R1.* et R2.* avec **NULL** quand nécessaire
  - R1 **NATURAL LEFT OUTER JOIN** R2 **:** Remplir R2.* avec **NULL** quand nécessaire
  - R1 **NATURAL RIGHT OUTER JOIN** R2 **:** Remplir R1.* avec **NULL** quand nécessaire

- **Théta-Jointures externes :**

  Elles sont définies de façon similaire :
  - R1 **FULL OUTER JOIN** R2 **ON** prédicat
  - R1 **LEFT OUTER JOIN** R2 **ON** prédicat
  - R1 **RIGHT OUTER JOIN** R2 **ON** prédicat

Expressions Ensemblistes : Union

COMMANDES (**CNUM**, CNOM, *PNOM*, QUANTITE) ; **FOURNITURE** (***PNOM, FNOM***, PRIXA)

- **Requête: Produits qui coûtent plus que 1000€ ou ceux qui sont commandés par Jules**

Algèbre:

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

SQL :

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

Remarque:

- l'union **élimine** les doublons
- Pour les garder on utilise l'opération **UNION ALL** : le résultat contient chaque tuple *a* + *b* fois, où *a* et *b* est le nombre d'occurrences du tuple dans le 1° et le 2° SELECT.

Expressions Ensemblistes : Différence

Table EMPLOYE (**EMPNO**, ENOM, *DEPTNO*, SAL) - **DEPT** (**DEPNO**, DNOM, LOC)

- **Requête: Départements sans employés ?**

Algèbre:

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

SQL :

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

Remarque:

- la différence **ne fait pas partie du standard**
- la différence **élimine** les doublons
- pour les garder on utilise l'opération **EXCEPT ALL** : le résultat contient chaque tuple *a* - *b* fois, où *a* et *b* est le nombre d'occurrences du tuple dans la 1° et le 2° SELECT.

Expressions Ensemblistes : Intersection

EMPLOYE (**EMPNO**, ENOM, *DEPTNO*, SAL) ; **DEPARTEMENT** (**DEPTNO**, DNOM, LOC)

- **Requête: Départements ayant des employés gagnant plus que 20000€ et se trouvant à Paris ?**

Algèbre :

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

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

Remarques :

- l'intersection ne fait **pas partie du standard**
- l'intersection **élimine** les doublons
- pour les garder 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 tuple dans la 1° et le 2° SELECT

Division

FOURNITURE (***FNUM, PNUM***, QUANTITE) - **PRODUIT** (**PNUM**, PRIXV)
FOURNISSEUR (**FNUM**, FNOM, STATUS, VILLE)

- **Requête: Nom 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))
```


2 – Imbrication de requêtes

- **Cas simples,**
- **ANY**
- **ALL**
- **EXISTS**
- **Formes équivalentes de quantification**

Requêtes imbriquées : cas simple de jointure

la **Jointure** peut aussi s'exprimer par 2 blocs imbriqués

COMMANDES (**CNUM**, CNOM, *PNUM*, QUANTITE) ; **FOURNITURE** (***PNUM, FNUM***, PRIXA)

- **Requête: N°, prix et N° des fournisseurs des Produits commandés par Paul ?**

Algèbre:

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

SQL :

```
SELECT FOURNITURE.PNUM, PRIXA, FNUM
FROM FOURNITURE,COMMANDES
WHERE FOURNITURE.PNUM = COMMANDES.PNUM AND CNOM = 'PAUL'
```

Equivalent à :

```
SELECT PNUM,PRIXA,FNUM
FROM FOURNITURE
WHERE PNUM IN (SELECT PNUM
               FROM COMMANDES
               WHERE CNOM = 'PAUL')
```

Requêtes imbriquées : cas simple de Différence

la **Différence** peut aussi s'exprimer par 2 blocs imbriqués

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 DISTINCT DEPTNO
FROM EMPLOYE
```

Equivalent à :

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


Requêtes imbriquées plus complexes : ANY

FOURNITURE (***PNUM, FNUM,*** PRIXA)

COMMANDE (*CNUM*, CNOM, *PNUM*, QUANTITE)

- **Requête: N° des fournisseurs du produit N°130 à un prix d'achat inférieur au prix d'achat maximum des produit n°140 ?**

```
SELECT FNUM FROM FOURNITURE
   WHERE PNUM = 130 AND PRIX < ANY (SELECT PRIX
                    FROM FOURNITURE WHERE PNUM = 140)
```

Remarque: la condition < ANY (SELECT F FROM . . .) est vraie ssi la comparaison >v est vraie au moins pour une valeur v du résultat du bloc (SELECT F FROM . . .).

- **Requête: N°, prix d'achat et n° fournisseur des produits commandés par Paul**

```
SELECT PNUM, PRIXA, FNUM FROM FOURNITURE
WHERE PNUM = ANY (SELECT PNUM FROM COMMANDE WHERE CNOM = 'PAUL')
```

Remarque: les prédicats IN et = ANY sont utilisés de la même façon.

Requêtes imbriquées plus complexes : ALL (1)

Soit la table :

COMMANDE (*CNUM*, CNOM, *PNUM*, QUANTITE)

- **Requête: Clients ayant commandé la plus petite quantité de produit N°130 ?**

```
SELECT CNOM
FROM COMMANDE
WHERE PNOM = 'Brique' AND
QUANTITE <= ALL (SELECT QUANTITE
                 FROM COMMANDE
                 WHERE PNOM = 130)
```

Remarque: la condition <= ALL (SELECT F FROM . . .) est vraie ssi la comparaison ≤ v est vraie pour toutes les valeurs v du résultat du bloc (SELECT F FROM . . .).

Requêtes imbriquées plus complexes : ALL (2)

Soit les tables :

- **EMPLOYE** (**EMPNO**, ENOM, *DEPNO*, SAL)
- **DEPARTEMENT** (**DEPTNO**, DNOM, LOC)

- **Requête: Départements sans employés**

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

Remarque: les prédicats NOT IN et NOT = ALL sont utilisés de la même façon.

Requêtes imbriquées plus complexes : EXISTS

FOURNISSEUR (**FNUM**, FNOM, STATUS, VILLE) ; **FOURNITURE** (***PNUM, FNUM***, PRIX)

- **Requête: N° des fournisseurs qui fournissent au moins un produit**

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

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

- **Requête: N° des fournisseurs qui ne fournissent aucun produit**

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

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

Formes Équivalentes de Quantification

Si θ est un des opérateurs de comparaison <, =, >,…

- la condition x θ **ANY** (SELECT Ri.y FROM R1,… Rn WHERE p) est équivalente à :

```
EXISTS (SELECT * FROM R1, … Rn WHERE p AND x M Ri.y)
```

- la condition x θ **ALL** (SELECT Ri.y FROM R1, … Rn WHERE p) est équivalente à :

```
NOT EXISTS (SELECT * FROM R1, … Rn WHERE (p) AND NOT (x M Ri.y))
```

Exemple :

COMMANDE (**CNUM**, CNOM, *PNUM*, QUANTITE) ; **FOURNITURE** (***PNUM, FNUM***, PRIXA)

- **Requête: N°, prix et n°fournisseur des produits commandés par Paul**

```
SELECT PNUM, PRIXA, FNUM FROM FOURNITURE
WHERE EXISTS (SELECT * FROM COMMANDE
              WHERE CNOM = 'PAUL' AND PNUM = FOURNITURE.PNUM)
```

Equivalente à :

```
SELECT PNUM, PRIXA, FNUM FROM FOURNITURE
WHERE PNUM = ANY (SELECT PNUM FROM COMMANDE WHERE CNOM = 'PAUL')
```


Formes Équivalentes de Quantification (suite)

FOURNITURE (***PNUM, FNUM***, PRIXA)

- **Requête: N° des fournisseurs qui fournissent au moins un produit avec un prix d'achat supérieur au prix d'achat des produits fournis par le fournisseur N°125**

```
SELECT DISTINCT P1.FNUM
FROM FOURNITURE P1
WHERE NOT EXISTS (SELECT * FROM FOURNITURE P2
                  WHERE P2.FNUM = 125 AND P1.PRIX <= P2.PRIX)
```

Equivalent à :

```
SELECT DISTINCT FNOM FROM FOURNITURE
WHERE PRIX > ALL (SELECT PRIX FROM FOURNITURE WHERE FNUM = 125)
```


3 – Fonctions de calculs et opérateurs d'agrégation de SQL

- **Fonction de calcul :**
  - **MAX, MIN, AVG, SUM,**
- **Opérations d'Agrégation :**
  - **GROUP BY, HAVING, ORDER BY**
- **Récursion dans SQL3**

Fonctions de Calcul : COUNT, AVG, SUM

La fonction **COUNT(*)** compte le nombre des tuples du résultat d'une requête **sans élimination des tuples doubles ni vérification des valeurs nulles**. Dans le cas contraire on utilise la clause **COUNT(UNIQUE…).**

Tables :

PRODUIT (**PNUM**, PNOM, PRIXV)
FOURNISSEUR (***FNUM***, FNOM, STATUT, VILLE)
FOURNITURE (*FNUM, PNUM*, PRIXA**)**
COMMANDE (*CNUM*, CNOM, *PNUM*, QUANTITE)

- **Requête: Nombre de Fournisseurs de Paris**

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

- **Requête: Nombre de Fournisseurs qui fournissent actuellement des produits**

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

- **Requête: Quantité totale de produit N°125 commandés**

```
SELECT SUM (QUANTITE) FROM COMMANDES
WHERE PNUM = 125
```


Fonctions de Calcul : MAX, MIN, AVG

Tables :

PRODUIT (**PNUM**, PNOM, PRIXV)
FOURNISSEUR (***FNUM***, FNOM, STATUT, VILLE)
FOURNITURE (*FNUM, PNUM*, PRIXA**)**
COMMANDE (*CNUM*, CNOM, *PNUM*, QUANTITE)

- **Requête: Prix de vente moyen de Briques**

```
SELECT AVG (PRIXV)
FROM PRODUIT
WHERE PNOM = 'Brique'
```

- **Requête: Quantité maximum de produit n°125 commandés dans les commandes**

```
SELECT MAX (QUANTITE)
FROM COMMANDES
WHERE PNOM = 125;
```

- **Requête: Prix d'achat minimum du produit n°125**

```
SELECT MIN (PRIXA)
FROM FOURNITURE
WHERE PNUM = 125;
```


Opérations d'Agrégation : GROUP BY (1)

La clause **GROUP BY** permet de préciser les attributs de partitionnement des tables déclarées dans la clause FROM, par exemple un regroupement des fournisseurs par ville :

- **Requête: Nombre de fournisseurs par ville :**

FOURNISSEUR (***FNUM***, FNOM, STATUT, VILLE)

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

Base :

VILLE	FNOM
Paris	DUVAL
Paris	DURAND
Lyon	DUPONT
Lyon	LAFRANCE
Lyon	SMITH

Résultat :

VILLE	COUNT(FNOM)
Paris	2
Lyon	3

Opérations d'Agrégation : GROUP BY (2)

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

- **Requête: Donner pour chaque produit son prix de vente moyen :**

PRODUIT (**PNUM**, PNOM, PRIXV)

```
SELECT PNOM, AVG (PRIXV) FROM PRODUIT GROUP BY PNOM
```

Résultat :

PNOM	AVG(PRIX)
Brique	10,5
Parpaing	9,8

Opérations d'Agrégation : HAVING

La clause **HAVING** permet d'éliminer des partitionnements, comme la clause WHERE élimine des tuples du résultat d'une requête.

- **Requête: N° des produits fournis par deux ou plusieurs fournisseurs avec un prix d'achat supérieur à 100 euros**

FOURNITURE (*FNUM, PNUM*, PRIXA)

```
SELECT PNOM FROM FOURNITURE WHERE PRIXA > 20 GROUP BY PNOM
HAVING COUNT(*) >= 2
```

Avant la clause HAVING :

PNUM	FNUM	PRIXA
125	231	40
145	232	33
145	233	35

Après la clause HAVING :

PNUM	FNUM	PRIXA
145	232	33
145	233	35

On garde les produits dont le nombre des fournisseurs est ≥ 2.

Les conditions de sélection peuvent être appliquées avant le calcul d'agrégat (clause WHERE) mais aussi après (clause HAVING).

- **Requête: N° et prix d'achat moyen des produits fournis par les fournisseurs dont le siège est à Paris, seulement si leur prix d'achat minimum est > à 100€**

```
SELECT PNUM, AVG(PRIXA) FROM FOURNITURE, FOURNISSEUR
WHERE VILLE = 'Paris' AND FOURNITURE.FNOM = FOURNISSEUR.FNOM
GROUP BY PNOM HAVING MIN(PRIX) > 100
```


Opérations d'Agrégation : ORDER BY

En général, le résultat d'une requête SQL n'est pas trié.

FOURNISSEUR (*FNUM*, FNOM, STATUT, VILLE)
FOURNITURE (*FNUM, PNUM*, PRIXA)

Pour trier le résultat par rapport aux valeurs d'un ou de plusieurs attributs, on utilise la clause ORDER BY :

- **Requête: Ville, Nom des fournisseurs et N° des produits fournis triés par ville (1er critère de tri), nom de fournisseur (second critère de tri), et par n° de produit par ordre descendant (troisième critère de tri)**

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

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

Récursion dans SQL3

ENFANT (*NOMPAR, NOMENF*)

- **Requête: Les enfants de Charlemagne**

```
SELECT NOMENF
FROM ENFANT
WHERE NOMPAR='Charlemagne';
```

- **Requête: Les descendants de Charlemagne**

```
WITH RECURSIVE DESCENDANT(NOMANC, NOMDESC) AS
(SELECT NOMPAR, NOMENF 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';
```


4 – Mises à jour dans SQL

- **Insertion de tuples : INSERT**
- **Modification de tuples : UPDATE**
- **Suppression de tuples : DELETE**

Insertion de tuples : INSERT

Syntaxe : **INSERT INTO** R(a_1, a_2,…, a_n) **VALUES** (v_1, v_1,… v_n)

liste des **attributs (a_i)** de la table & liste des **valeurs** respectives de chaque attribut **(v_i).**

1. chaque a_i doit être un attribut de R
2. les attributs non indiqués restent à **NULL** ou à leur valeur par défaut.

toujours indiquer une valeur pour un attribut déclaré **NOT NULL**

PRODUIT (**PNUM**, PNOM, PRIXV)
FOURNISSEUR (***FNUM***, FNOM, STATUT, VILLE)

- **Insertion d'une ligne dans Produit :**

```
INSERT INTO PRODUIT (PNUM, PNOM, PRIXV)
VALUES (125, 'lasure', 15)
```

- **Insertion de 2 fournisseurs :**

```
INSERT INTO FOURNISSEUR (FNUM, FNOM, VILLE)
VALUES (234, 'Leroy-Merlin', 'Aubagne'), (235, 'Castorama',
'Marseille')
```

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

```
INSERT INTO PRODUIT (PNOM)
SELECT DISTINCT PNOM FROM PRODUIT
```


Modification de tuples : UPDATE

Syntaxe : **UPDATE** R **SET** a_1, = v_1, a_2, = v_2, … a_n, = v_n **WHERE** *condition*

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

1. énumération des attributs à modifier
2. indication pour chaque attribut à modifier de la nouvelle valeur
3. la clause **WHERE** *condition* permet de spécifier les lignes auxquelles s'applique la mise à jour : identique au **WHERE** du **SELECT**

Remarque : on ne peut pas violer les contraintes sur la table.

PRODUIT (**PNUM**, PNOM, PRIXV)
FOURNITURE (*FNUM, PNUM*, PRIXA**)**

- **Mise à jour du prix de vente du produit N°234**

```
UPDATE PRODUIT SET PRIXV = 12
WHERE PNUM = 234
```

- **Augmenter les prix de vente de tous les produits fournis par le fournisseur n°130 de 10% :**

```
UPDATE PRODUIT SET PRIXV = prix*1.1
WHERE PNUM IN (SELECT PNUM FROM FOURNITURE WHERE FNUM = 130)
```


Suppression de tuples : DELETE

Syntaxe : **DELETE FROM** R **WHERE** *condition*

Permet de supprimer une ou plusieurs lignes dans la table R.

- elle s'applique à des lignes et pas à des attributs
- la clause **WHERE** *condition* est indentique au **WHERE** du **SELECT**

PRODUIT (**PNUM**, PNOM, PRIXV)
FOURNISSEUR (***FNUM***, FNOM, STATUT, VILLE)
FOURNITURE (*FNUM, PNUM*, PRIXA**)**
COMMANDE (*CNUM*, CNOM, *PNUM*, QUANTITE)

- **Destruction des produits fournit par le fournisseur n°130 :**

```
DELETE FROM PRODUIT
WHERE PNUM IN (SELECT PNUM
                FROM FOURNITURE WHERE FNUM = 130)
```

- **Destruction de Castorama :**

```
DELETE FROM FOURNISSEUR
WHERE FNOM = 'Castorama'
```


5 – Opérations additionnelles

- **Opérations de définition de schémas : création/suppression de tables**
- **Les vues**
- **Administration des tables**

Opérations de définition de schémas : création/suppression de tables

Création de Tables : CREATE TABLE

```
CREATE TABLE PRODUIT(PNUM INTEGER, PNOM VARCHAR(20), PRIXV INTEGER,
PRIMARY KEY (PNUM));

CREATE TABLE FOURNISSEUR(FNUM INTEGER PRIMARY KEY, FNOM
VARCHAR(20), ville VARCHAR(16));

CREATE TABLE FOURNITURE (PNUM INTEGER NOT NULL, FNUM INTEGER NOT
NULL, PRIXA INTEGER,
FOREIGN KEY (PNUM) REFERENCES PRODUIT,
FOREIGN KEY (FNUM) REFERENCES FOURNISSEUR,
PNUM, FNUM PRIMARY KEY);
```

Destruction de Tables : DROP TABLE :

```
DROP TABLE FOURNITURE;
DROP TABLE PRODUIT;
DROP TABLE FOURNISSEUR;
```


Notion de VUE

Vue :

- "table **virtuelle**" composée à partir d'une ou plusieurs table(s) dont le **schéma** et les **tuples** se déduisent de ces tables comme étant le **résultat d'une requête d'interrogation**
- seules les **définitions de vues sont stockés dans la méta-base** (dictionnaire)
- **syntaxe SQL** :

```
CREATE VIEW <nom_vue> [(liste d'attributs)]  As <requête>
```

Usages des vues :

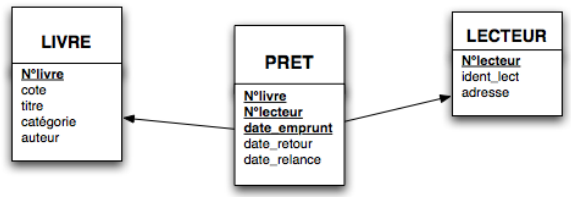
- **simplification de requêtes**
- **confidentialité en définissant des sous-ensembles de la BdD :**
  - type (attributs): **->** *cacher un ou plusieurs colonnes à un utilisateur*
  - occurrences (tuples) *-> cacher un ou plusieurs enregistrements à un utilisateur*
  - combinaison de types et d'occurrences: *-> confidentialité sophistiquée*

Proposées par certains SGBD relationnels :

- **vue concrète:** le contenu de la vue est calculé à sa définition et **stocké** dans la base de données, il reflète à instant t l'état de la BdD
- **clichés:** vue concrète **recalculée périodiquement** par le système

Usages des vues

Exemple de base de données relationnelle (formalisme Merise) :



Tables associées :

LIVRE (**N°livre**, cote, titre, catégorie, auteur)

LECTEUR (**N°lecteur**, ident_lect, adresse)

PRET (***N°livre, N°lecteur*, date_emprunt**, date_retour, date_relance)

Usages des vues

1- sous ensemble d'occurrences

```
CREATE VIEW policier AS
 SELECT N°livre, titre, auteur FROM livre
 WHERE catégorie = "roman policier" ;
```

2- restructuration d'information pour l'usager

```
CREATE VIEW lecteur_de_policier AS
 SELECT N°lecteur, ident_lecteur, adresse,N°livre, date_emprunt,
date_retour FROM livre, lecteur, pret
 WHERE  livre.N°livre = pret.N°livre
 AND lecteur.N°lecteur = pret.N°lecteur
 AND livre.catégorie = "roman policier" ;
```

3- dérivation par calculs de nouvelles informations, renomination,...

```
CREATE VIEW stat_prets (N°lecteur, nb_prets ) AS
 SELECT N°lecteur, COUNT(*) N°livre, date_emprunt, date_retour
 FROM lecteur, pret
 WHERE  lecteur.N°lecteur = pret.N°lecteur
 AND date_emprunt >"date D1"
 GROUP BY N°lecteur ;
```

Usages des vues

4- simplification de requêtes

- **Nom des lecteurs ayant emprunté plus de 3 romans policier de Simenon le même jour ?**

```
SELECT DISTINCT ident_lecteur
FROM lecteur_de_policier, policier
WHERE lecteur_de_policier.N°livre = policier.N°livre
AND auteur = "Simenon"
GROUP BY N°lecteur, date_emprunt
HAVING COUNT (N°livre) >3;
```

L'exécution de cette requête fait appel aux 3 vues précédemment définies :

```
CREATE VIEW policier AS
  SELECT N°livre, titre, auteur FROM livre
  WHERE catégorie = "roman policier" ;
CREATE VIEW lecteur_de_policier AS
 SELECT N°lecteur, ident_lecteur, adresse,N°livre, date_emprunt, date_retour
 FROM livre, lecteur, pret
 WHERE  livre.N°livre = pret.N°livre
 AND lecteur.N°lecteur = pret.N°lecteur
 AND livre.catégorie = "roman policier" ;
CREATE VIEW stat_prets (N°lecteur, nb_prets ) AS
 SELECT N°lecteur, COUNT(*) N°livre, date_emprunt, date_retour
 FROM lecteur, pret
 WHERE  lecteur.N°lecteur = pret.N°lecteur
 AND date_emprunt >"date D1"
 GROUP BY N°lecteur ;
```


Mises à jour sur vues

interdite en général

car :

- **attributs non définis dans la vue :**

  toute insertion d'un tuple dans la vue entraîne l'insertion d'un tuple dans la BdD ayant une valeur indéterminée pour des attributs non visibles da la vue

- **risques d'incohérences :**

  lorsque la vue est obtenue par jointure de relations réelles

  ex: V = A joint B :

  insert V => insert A ? insert B? insert A et B?

d'où :

- **la mise à jour sur vue ne peut être automatique**
- **il faut décrire pour chaque vue la sémantique des opérations de mise à jour**

Administration des tables

2 types de d'administration :

1- Centralisée :

- **l'administrateur** définit les **usagers** et ce qu'ils ont le droit de faire sur les tables et les vues

2 - Décentralisée :

- lorsqu'un **utilisateur** crée une table, il devient **administrateur** de sa table
- il possède sur cette table les droits de : *lecture, insertion, suppression, modification, modification de la structure physique (alter)*

Transmission/révocation de droits : commandes GRANT et REVOKE

```
GRANT <liste_droits> ON <nom de la table|vue>
TO <liste_usagers> [With Grant Option]

REVOKE <liste_droits> ON <nom de la table|vue>
FROM <liste_usagers>
```