Plan du cours

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.

Plan du cours

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
```


Les interfaces. "Démo"

Rappel de l'intro : *diversité des interfaces : langages BD, ETL, menus, saisies, rapports, ...*

Exemple avec SGBD Postgres.
Base NFP1071011 (localhost).

1. via pgAdmin,
2. via un shell SQL (équivalent de SQL Plus sous Oracle),
3. via l'ETL (Pentaho Data Integration),
4. ...

Retenir : Accès par différents programmes ou outils (interfaces) mais ce sont des requêtes SQL qui sont envoyées (au SGBD) par les programmes ou les outils.

Plan du cours

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 de sélection

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 un 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).
Projection et sélection, mais le critère de sélection n'est pas exprimable avec l'algèbre relationnelle.

Plan du cours

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 (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 $+, -, *, /$).

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 :

FNOM
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 :

FNOM
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?

Plan du cours

4. SQL
   - Principes
   - Projection
   - Sélection
   - Prise en compte de données manquantes (NULL)
   - Jointures
   - Union
   - Différence
   - Intersection
   - Imbrication des requêtes en SQL
   - Fonctions de calcul
   - Opérations d'agrégation
   - Création et mise à jour de relations

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
```

- 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 ENOM, DEPNO, DNOM
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.

θ-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 :

- soit juste ceux de la relation de gauche (en complétant par des valeurs NULL sur les attributs de la relation de droite), LEFT
- soit juste ceux de la relation de droite (en complétant par des valeurs NULL sur les attributs de la relation de gauche), RIGHT
- soit ceux des deux relations (en complétant par des valeurs NULL, à gauche ou à droite selon le cas). FULL

- 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
```

Jointure externe

- 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.

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

Plan du cours

Union

COMMANDES(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Produits qui coûtent plus que 1000 euros 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.

Plan du cours

Différence

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

```
SELECT DEPNO FROM   DEPARTEMENT
   EXCEPT
SELECT DEPNO 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.

Plan du cours

Intersection

L'intersection ne fait pas partie du standard.
EMPLOYE(EMPNO,ENOM,DEPNO,SAL)
DEPARTEMENT(DEPNO,DNOM,LOC)
REQUÊTE: *Départements ayant des employés qui gagnent plus que 2000 euros et qui se trouvent à Paris*

ALGÈBRE :

$$\pi_{DEPNO}(\sigma_{LOC="Paris"}(DEPARTEMENT)) \cap \pi_{DEPNO}(\sigma_{SAL>2000}(EMPLOYE))$$

SQL:

```
SELECT DEPNO
FROM   DEPARTEMENT
WHERE  LOC = 'Paris'
INTERSECT
SELECT DEPNO
FROM   EMPLOYE
WHERE  SAL > 2000
```

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.

Plan du cours

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:

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

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(DEPNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
ALGÈBRE :

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

SQL:

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

ou

```
SELECT DEPNO
FROM DEPARTEMENT
EXCEPT
SELECT DEPNO
FROM EMPLOYE
```


Requêtes imbriquées plus complexes : ANY(SOME)

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 = 'briques'
         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 ...).
SOME peut être utilisé à la place de ANY.

"IN" et "= SOME"

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 = SOME (SELECT PNOM
                    FROM   COMMANDE
                    WHERE  CNOM = 'JEAN')
```

NOTE: Les prédicats IN et = SOME 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 = 'briques' AND
       QUANTITE <= ALL (SELECT QUANTITE
                        FROM   COMMANDE
                        WHERE  PNOM = 'briques')
```

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(DEPNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
SQL:

```
SELECT DEPNO
FROM   DEPARTEMENT
WHERE  DEPNO NOT = ALL (SELECT DEPNO
                        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 (nom, ville) qui fournissent au moins un produit*
SQL :

```
SELECT FNOM, VILLE
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 (nom, ville) qui ne fournissent aucun produit*
SQL:

```
SELECT FNOM, VILLE
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 θ est un des opérateurs de comparaison $<, =, >, \ldots$

- La condition
  x θ SOME (SELECT Ri.y FROM R1, ... Rn WHERE p)
  est équivalente à
  EXISTS (SELECT * FROM R1, ... Rn WHERE p AND x θ 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 θ Ri.y))

Exemple : "EXISTS" et "= SOME"

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 = SOME (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))
```


Plan du cours

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: COUNT(FNOM) ne compte pas les valeurs NULL et n'élimine pas les dupliqués. COUNT(DISTINCT FNOM) ne compte pas les valeurs NULL mais élimine les dupliqués.

SUM et AVG

REQUÊTE: *Quantité totale de briques commandées*

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

REQUÊTE: *Coût moyen de Briques fournies*

```
SELECT AVG (PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'briques'
```

ou

```
SELECT SUM (PRIX)/COUNT(PRIX)
FROM FOURNITURE
WHERE PNOM = 'briques'
```


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';
```


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


Plan du cours

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)
BRIQUES	10.5
ARDOISE	9.8

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

Plan du cours