

Introduction aux bases de données

Langage SQL (Partie 1)

Le langage SQL

- I. **Introduction à SQL et au SGBD Oracle**
- II. Interrogation des données (SELECT)

I. Introduction à SQL et à Oracle

- SQL : Structured Query Language
- SQL : langage pour les BDR basé sur l'algèbre relationnelle
- SQL est un standard ANSI/ISO depuis 1986
 - Version SQL-92 ou SQL2 (standard bien supporté)
 - Version SQL-99 ou SQL3 (dernier standard, peu supporté)
- Le SQL d'Oracle est proche de la norme
 - Oracle 8.i : Compatible SQL2
 - A partir d'Oracle 9.i : Compatible SQL3
 - Version actuelle : Oracle 12c

I. Introduction à SQL et à Oracle

Généralités sur le langage SQL

SQL comporte trois parties :

- **LDD** (Langage de Définition de Données) pour créer, modifier et supprimer des définitions de tables (schémas)
 - Commandes : `create table`, `drop table`, `alter table`
- **LMD** (Langage de Manipulation de Données) pour rechercher, ajouter, supprimer, modifier les données
 - Commandes : `select`, `insert`, `delete`, `update`
 - Valider/annuler les mise à jour : `commit`, `rollback`
- **LCD** (Langage de Contrôle de Données) pour gérer les protections d'accès.
 - Commandes : `grant`, `revoke`

Oracle-SQL et la casse des caractères (majuscule/minuscule)

- Il n'y a pas de différence entre des noms (ou identifiants) écrits en majuscules ou en minuscules
 - ex. noms d'attributs, de tables, de contraintes...
- Les noms sont mémorisés en majuscules dans le catalogue
- Seul cas où la casse est prise en compte : la comparaison de chaînes de caractères dans les données

Nommage des relations/attributs dans Oracle

- Propriétaire d'un objet = utilisateur qui l'a créé
- Nommage d'une **relation**
 - *nom-de-la-relation* si elle m'appartient
 - *nom-du-propriétaire.nom-de-la-relation* si non
- Nommage d'un **attribut** dans une requête
 - si pas d'ambiguïté : *nom-attribut*
 - sinon : *nom-de-relation.nom-attribut*
 - ou : *alias-de-relation.nom-attribut*

Le langage SQL

- I. Introduction à SQL et au SGBD Oracle
- II. **Interrogation des données (SELECT)**

Base de données Produits (1/2)

Client (noClient, nom, prénom, ddn, rue, CP, ville)

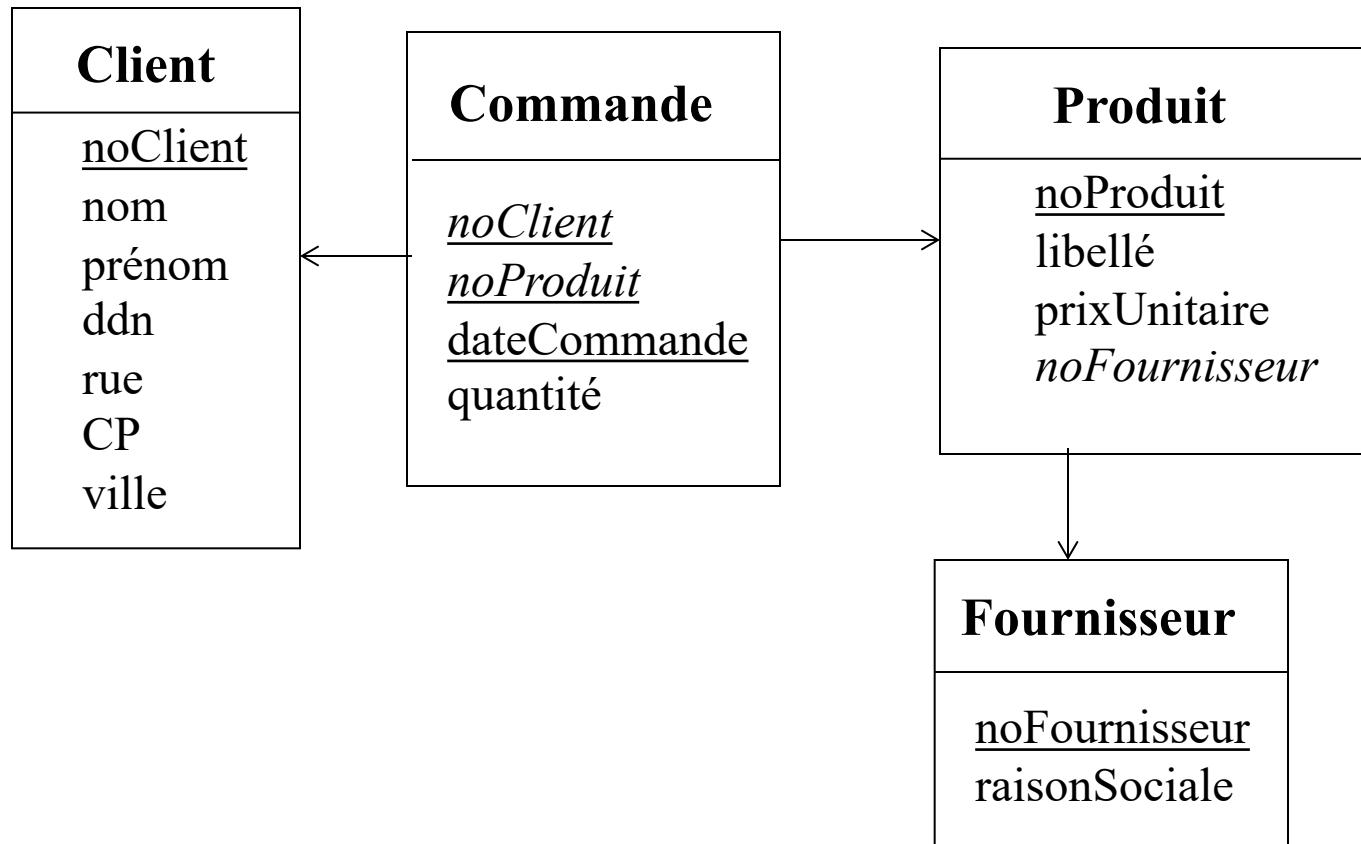
Produit (noProduit, libellé, prixUnitaire, *noFournisseur*)

Fournisseur (noFournisseur, raisonSociale)

Commande (noClient, noProduit, dateCommande, quantité)

Clés primaires soulignées
Clés étrangères en italique

Base de données Produits (2/2)



Syntaxe générale de la commande SELECT

```
SELECT * | { [ALL|DISTINCT]
    expression [[AS] nomColonne]
    [,expression [[AS] nomColonne]]... }

FROM   relation [alias] [,relation [alias] ...]
[WHERE condition]
[GROUP BY nomColonne [,nomColonne]...]
[HAVING condition]
[ORDER BY nomColonne [ASC|DESC]
          [,nomColonne [ASC|DESC]...]]
```

La commande SELECT

- La commande SELECT permet de rechercher des données à partir de plusieurs tables ; le résultat est présenté sous forme d'une table réponse

Q1. Donner les noms, libelle et prix des produits

```
SELECT      nom, libelle, prixUnitaire  
FROM        Produit
```

La commande SELECT

Q1. Donner les noms, libelle et prix des produits

```
SELECT      P.nom,  P.libelle,  P.prixUnitaire  
FROM        Produit  P
```

Alias de nom de table

- On peut introduire dans la clause FROM un synonyme (alias) à un nom de table en le plaçant après le nom de la table
- Noms de table ou synonymes peuvent être utilisés pour préfixer les noms de colonnes dans le SELECT
- Préfixes obligatoires dans des cas particuliers (ex. auto-jointure)
Leur emploi est cependant conseillé pour la clarté

La commande SELECT

Q2. Donner tous les renseignements sur les fournisseurs

```
SELECT      *
FROM        Fournisseur
```

- Une étoile (*) permet de lister tous les attributs

Expression de calcul sur les colonnes dans la liste de projection

Q3. Donner les références des produits et leur prix majoré de 20%

```
SELECT      noProduit,  prixUnitaire*1.2  
FROM        Produit
```

- Il est possible d'effectuer des opérations arithmétiques (+, -, *, /) sur les colonnes extraites

La commande SELECT et la clause DISTINCT

Q4. Donner les différents clients qui ont passé des commandes

```
SELECT      DISTINCT noClient  
FROM        Commande
```

- Contrairement à l'algèbre relationnelle, SQL n'élimine pas les doublons. Pour éliminer les doublons il faut spécifier **DISTINCT**
- Cette requête correspond à : PROJECT(Commande, noClient)

Expression des restrictions

Q5. Donner les noms des produits dont le prix est < 20

```
SELECT      noProduit  
FROM        produit  
WHERE      prixUnitaire < 20
```

- La condition de recherche est spécifiée après la clause WHERE
- Une condition simple avec :
 - un prédicat d'égalité ou d'inégalité ($=, \neq, <, >, \leq, \geq$)
 - un prédicat **LIKE**, **BETWEEN**, **IN**, **EXISTS**, **ALL** ou **ANY**
 - un test de valeur **NULL**
- Possible de connecter les conditions simples à l'aide des connecteurs **AND**, **OR** (et **NOT**)

Restriction d'une relation sur une condition

Q6. Sélectionner les Produits dont le prix est inférieur à 20€ et le numéro est supérieur à 30

```
SELECT      *
FROM        Produit
WHERE       prixUnitaire < 20 AND noArticle > 30
```

- Cette requête correspond à l'expression algébrique :

```
RESTRICT (Produit, prixUnitaire < 20 ET noArticle > 30)
```

Restriction et projection

Q7. Produire les noClient et dateCommande des Commandes dont la date est postérieure au 01/01/2004

```
SELECT      noClient,  dateCommande  
FROM        Commande  
WHERE       dateCommande > '01-JAN-04'
```

- Cette requête correspond à l'expression algébrique :

```
PROJECT (RESTRICT (Commande, dateCommande > '01-JAN-04'),  
        noClient, dateCommande)
```

Restriction d'une relation sur une condition

- Produits dont le prix est compris entre 50 et 100€

```
SELECT *  
FROM Produit  
WHERE prixUnitaire >= 50 AND prixUnitaire <= 100
```

```
SELECT *  
FROM Produit  
WHERE prixUnitaire BETWEEN 50 AND 100
```

- Produits dont le prix est inférieur à 50€ ou supérieur à 100 €

```
SELECT *  
FROM Produit  
WHERE prixUnitaire < 50 OR prixUnitaire > 100
```

Restriction : Opérateurs IS NULL et LIKE

- Commandes en quantité indéterminée (null)

```
SELECT      *
FROM        Commande
WHERE       quantité IS NULL
```

- Clients dont le nom commence par B, se termine par B et contient au moins 3 caractères

```
SELECT      *
FROM        Client
WHERE       nom LIKE 'B_%B'
```

LIKE recherche des chaînes de caractères correspondant à un *patron* où :

 % : désigne une suite de **zéro à n** caractères quelconques

 _ : désigne **un et un seul** caractère quelconque

Possibilité de définir un caractère d'échappement :

```
SELECT dname FROM dept WHERE dname LIKE 'A\%S%' ESCAPE '\'
```

Notion de valeur indéfinie (NULL) Et des tables de vérité traditionnelles

a	b	a AND b	a OR b	NOT a
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
NULL	TRUE	NULL	TRUE	NULL
NULL	NULL	NULL	NULL	NULL
NULL	FALSE	FALSE	NULL	NULL

a, b : expressions booléennes à valeurs dans {TRUE, FALSE, NULL}

Restriction : Opérateur IN

- Prénom des clients dont le nom est Dupont, Durant ou Martin

```
SELECT    prénom
```

```
FROM      Client
```

```
WHERE     nom IN ('Dupond', 'Durant', 'Martin')
```

- Prénom des clients dont le nom n'est pas dans l'ensemble {Dupont, Durant, Martin}

```
SELECT    prénom
```

```
FROM      Client
```

```
WHERE     nom NOT IN ('Dupond', 'Durant', 'Martin')
```

Expression de calcul sur les colonnes dans la condition (du WHERE)

- Une condition peut comporter une expression de calcul
*ex: Chercher les tuples de T pour lesquels $X < Y^{**}3+Z/40$*

```
SELECT *
FROM   T
WHERE  X  <  Y**3+Z/40
```

- Une expression peut aussi faire appel à des **fonctions**
ex: Liste des commandes de la journée

```
SELECT  *
FROM    Commande
WHERE   dateCommande = CURRENT_DATE
```

Liste des principales fonctions sous Oracle (1/2)

- ABS(n) :Valeur absolue
- CEIL(n) : plus petit entier $\geq n$
- FLOOR(n) : plus grand entier $\leq n$
- MOD(m,n) : Reste de la division euclidienne de m par n
- POWER(m,n) : m élevé à la puissance n
- SIGN(n) : Signe de n
- SQRT(n) : Racine carrée de n
- INITCAP(ch) :Première lettre en majuscule
- LOWER(ch) :Toutes les lettres en minuscules
- UPPER(ch) :Toutes les lettres en majuscules
- LTRIM(chaîne[,ens_car])) : Suppression de caractères par la gauche

- RTRIM(chaîne[,ens_car])) :Suppression de caractères par la droite
- REPLACE(ch,car[,ch]) : Remplacement de caractères
- SUBSTR(ch,position[,long]) : Extraction d'une chaîne
- TRANSLATE(ch,car1,car2) : Transcodage
- SOUNDEX(ch) : Comparaison phonétique
- LPAD(ch,lg[,car]) : Compléter par la gauche
- RPAD(ch ,lg[,car]) :Compléter par la droite
- ASCII(ch) : Donne la valeur ASCII
- INSTR(ch,sous_ch(position,n)) : Recherche la sous-chaîne dans la chaîne

Liste des principales fonctions sous Oracle (2/2)

- LENGTH(ch) : Longueur de chaîne
- ADD_MONTHS(TO_DATE(date),n) : Addition de mois
- LAST_DAY(TO_DATE(date)) : Dernier jour du mois
- MONTHS_BETWEEN(TO_DATE(date1), TO_DATE(date2)) : Nombre de mois entre deux dates
- NEXT_DAY(TO_DATE(date),jour) : Date du prochain jour
- SYSDATE : Date et heure système
- ROUND(TO_DATE(date)[,précision]) : Arrondi d'une date
- TRUNC(TO_CHAR(date)[,précision]) : Troncature d'une date
- TO_NUMBER(ch) : Conversion d'une chaîne en nombre

- TO_CHAR(exp[,format]) : Conversion d'une expression en chaîne
- TO_CHAR(date, 'mm') : extrait le mois de date ('dd' pour le jour, 'yyyy' l'année)
- TO_DATE(ch[,format]) : Conversion d'une chaîne en date
- NVL(expr,valeur) : Protège de la manipulation d'une valeur NULL
NVL (exp1, exp2) : retourne exp1 si exp1 n'est pas null et exp2 sinon. exp1 et exp2 doivent être des chaînes de caractères.
- DECODE(expr,valeur1,résultat1[,valeur2,résultat2 ...],default) : Incursion procédurale dans SQL ...
- GREATEST(n1,n2,...) : Le plus grand
- LEAST(n1,n2,...) : Le plus petit de la liste
- VSIZE(expression) : Nombre d'octets nécessaires pour le format interne
- UID :Identifiant numérique de l'utilisateur
- USER : Nom de l'utilisateur

Tri des résultats

- La clause **ORDER BY** permet de spécifier les colonnes définissant les critères de tri
- Le tri se fera d'abord selon la première colonne spécifiée, puis selon la deuxième colonne etc.

Q. Liste des produits en les triant par fournisseur puis par prix décroissant (pour les produits du même fournisseur)

```
SELECT      *
FROM        produit
ORDER BY    noFournisseur, prixUnitaire
```

- L'ordre du tri est précisé par **ASC (croissant)** ou **DESC (décroissant)** par **défaut ASC**

Expression de jointures : produit cartésien

- Le produit cartésien s'exprime en incluant plusieurs tables après la clause **FROM**.

```
SELECT      *
FROM relation1, relation2
```

Ex. Produire toutes les combinaisons possibles de Client et de Commande

```
SELECT      *
FROM        Client, Commande
```

- Cette requête correspond à l'expression algébrique :
 $\text{PRODUCT}(\text{Client}, \text{Commande})$

Jointure(s) implicite(s)

- La condition de jointure est exprimée dans la clause WHERE

```
SELECT attribut1 ...
FROM    relation1, relation2
WHERE   condition
```

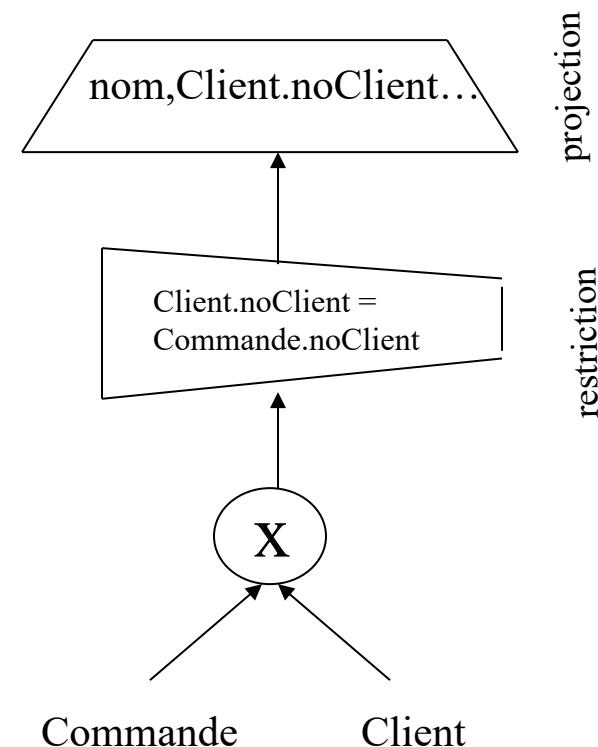
Cette commande SELECT combine **produit cartésien, restriction et projection**

N.B. Nécessité de préfixer le nom d'un attribut par sa relation en cas d'ambiguïté

Jointure implicite : exemple de requête

Q. Liste des commandes avec le nom du client

```
SELECT Cl.nom, Cl.noClient,
       Cd.noProduit, Cd.dateCommande,
       Cd.quantité
  FROM Commande Cd, Client Cl
 WHERE Cl.noClient = Cd.noClient
```

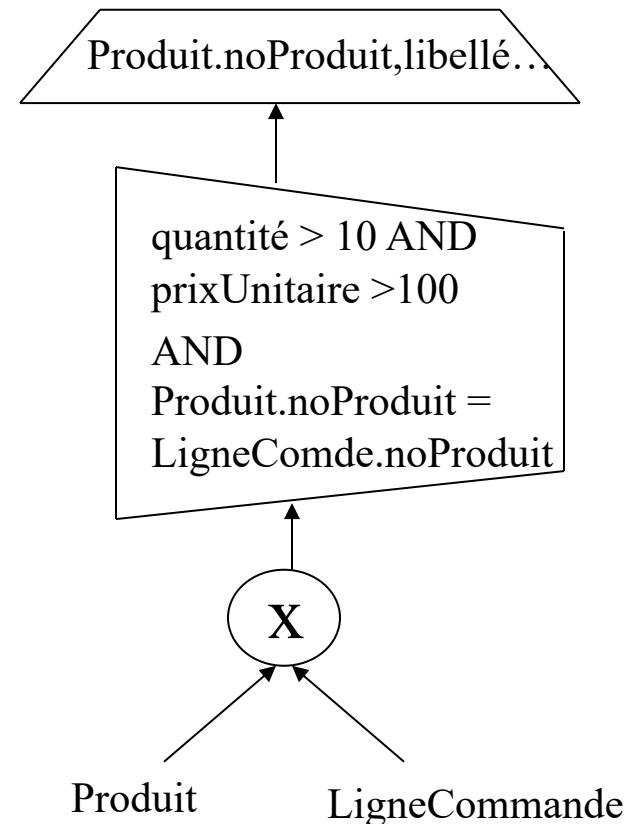


**Arbre algébrique
correspondant à l'expression
SQL**

Jointure implicite : autre exemple

ex. Produits commandés en quantité supérieure à 10 et dont le prix dépasse 100 €. Afficher les numéros de produit, leur libellé , leur prix unitaire ainsi que numéro de la commande.

```
SELECT P.noProduit, P.libellé,  
       P.prixUnitaire, C.noCommande  
FROM   Produit P, Commande C  
WHERE  quantité > 10  
AND    prixUnitaire > 100  
AND    P.noProduit = C.noProduit
```



Arbre algébrique

Utilisation d'alias et auto-jointure

Ex. Noms des clients de la même ville que DUPONT

```
SELECT      C2.nom  
FROM        Client C1 , Client C2  
WHERE       upper(C1.nom) = 'DUPONT' AND C1.ville = C2.ville
```

- Cet exemple utilise la jointure de la table « client » avec elle-même (auto-jointure)
- Pour pouvoir distinguer les attributs des 2 copies, il faut introduire 2 alias différents C₁ et C₂ pour la table Client

Jointures explicites

- Opérateur **JOIN** correspond à la combinaison produit cartésien/restriction (partie FROM de la commande SELECT)
- Opérateur **LEFT/RIGHT JOIN** correspond à la jointure externe
 - ✓ Permet de retenir lors d'une jointure les lignes d'une table qui n'ont pas de correspondant dans l'autre table, avec des valeurs NULL associées
 - ✓ On distingue **jointure externe gauche** ou **droite** selon que l'on retient les lignes de la première table ou de la deuxième

Jointures explicites

Jointure sur égalité d'attribut(s) de même nom

```
SELECT liste-projection  
FROM   Table1 JOIN Table2 USING (attr, attr...)  
       [JOIN Relation3 USING (attr, attr...) ... ]  
[WHERE condition]
```

Jointure sur condition (égalité d'attributs de noms différents)

```
SELECT liste-projection  
FROM   Table1 JOIN Table2 ON (condition1)  
       [JOIN relation3 ON (condition2) ... ]  
[WHERE condition]
```

Jointure externe (gauche ou droite)

```
SELECT liste-projection  
FROM   Table1 [LEFT|RIGHT] JOIN Table2  
       {ON (condition) | USING (attr, attr...) }  
[WHERE condition]
```

Exemples de Jointures explicites

- ```
SELECT * FROM Commande JOIN Client USING (noClient)
```
- ```
SELECT      *
FROM    Film   JOIN Programmation USING (titre)
        JOIN Salle USING (nom_cinema, noSalle)
        JOIN Cinema USING (nom_cinema)
WHERE realisateur like 'Barn%'
```
- ```
SELECT p1.nom, p1.prenom, p2.nom as "nom du père", p2.prenom
FROM Personne p1 JOIN Personne p2 ON (p1.id_pere = p2.id)
WHERE p1.statut = 'Etudiant'
```

# Fonctions d'agrégation

SQL fournit des fonctions de calcul opérant sur l'ensemble des valeurs d'une colonne d'une table et produit une valeur résultat unique (extension de l'algèbre relationnelle)

```
SELECT fctAgrégation
FROM relation(s) [WHERE condition]
```

fctAgrégation opère sur les lignes de la relation résultat (de la commande) où :

- **COUNT (\*)** : retourne le nombre de tuples de la relation résultat
- **COUNT (A)** : nombre de valeurs non NULL de la colonne A
- **COUNT (distinct A)** : nombre de valeurs non NULL distinctes de la colonne A
- **MAX (A)** : plus grande valeur de la colonne A
- **MIN (A)** : plus petite valeur de la colonne A

# Fonctions d'agrégation

SQL fournit des fonctions de calcul opérant sur l'ensemble des valeurs d'une colonne d'une table et produit une valeur résultat unique (extension de l'algèbre relationnelle)

```
SELECT fctAgrégation
FROM relation(s) [WHERE condition]
```

fctAgrégation opère sur les lignes de la relation résultat (de la commande) où :

- **SUM** (N) : somme des valeurs de la colonne N (ignore les valeurs NULL)
- **SUM** (distinct N) : somme des valeurs distinctes de la colonne N
- **AVG** (N) : moyenne des valeurs de la colonne N (ignore les valeurs NULL)

# Fonctions d'agrégation

- Nombre total d'articles et prix unitaire moyen

```
SELECT COUNT(*) AS nbArticles, AVG (prixUnitaire) AS prixMoyen
FROM Article
```

- Nombre de prixUnitaires (non NULL)

```
SELECT COUNT(prixUnitaire) AS nbPrix
FROM Article
```

- Nombre de prixUnitaires différents

```
SELECT COUNT(distinct prixUnitaire) AS nbPrixDiff
FROM Article
```

# Fonctions d'agrégation : Contraintes d'utilisation

- Une fonction d'agrégation doit être utilisée dans une clause SELECT sans résultats individuels

```
SELECT noProduit, max(prixUnitaire)
FROM Produit
```

} **Faux !!**

Requête **invalidé** puisque plusieurs noProduit et un seul maximum.

- Une fonction d'agrégation peut être utilisée dans une sous-requête  
    ⇒ sélection de résultats individuels dans la requête englobante

```
SELECT noProduit, libellé
FROM Produit
WHERE prixUnitaire = (SELECT max (prixUnitaire)
 FROM Produit)
```