

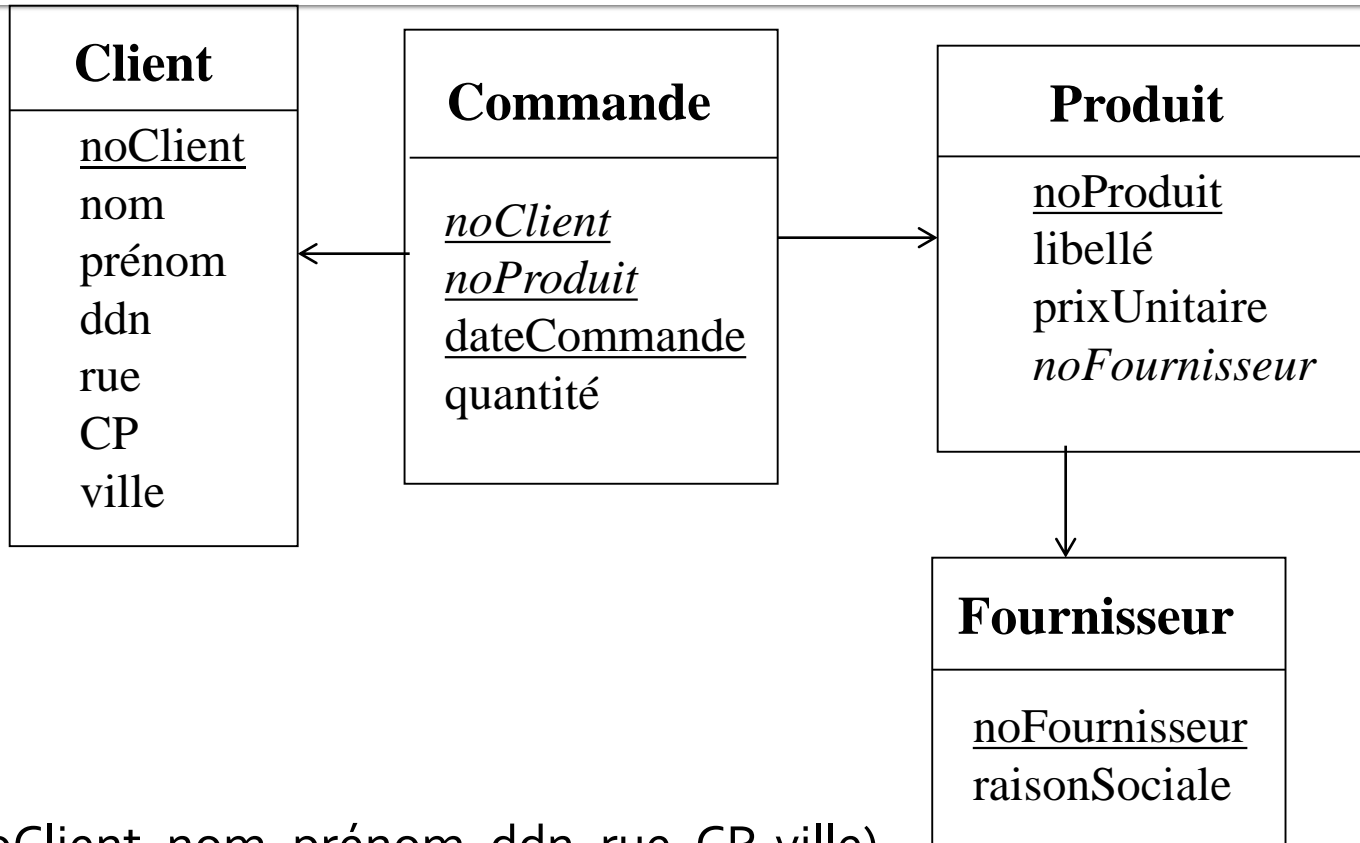
Introduction aux bases de données

Langage SQL Partie 2

Le langage SQL (*SGBD ORACLE*)

- I. Interrogation des données (sous-requêtes, GROUP BY)
- II. Mise à jour des données (INSERT, UPDATE, DELETE)

Base de données Produits



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

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

Fournisseur (noFournisseur, raisonSociale)

Commande (noClient, noProduit, dateCommande, quantité)

Sous-requêtes en SQL (1)

La commande SELECT permet l'imbrication de sous-requêtes au niveau de la clause WHERE

- Les sous-requêtes sont utilisées avec :
 - un prédicat de comparaison (=, <>, <, <=, >, >=)
 - le prédicat IN ou NOT IN
 - le prédicat EXISTS ou NOT EXISTS
 - le prédicat ALL ou ANY

Sous-requêtes en SQL (2)

- Une sous-requête dans un **prédicat de comparaison** (=, <>, <, <=, >, >=) doit renvoyer **une seule valeur** (un tuple réduit à un attribut)

Ex. Liste des produits ayant le prix le plus élevé

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

Sous-requêtes en SQL (3)

- Une **sous-requête** avec le prédicat **IN (NOT IN)**, **ALL** ou **ANY** doit renvoyer une **table à une colonne**

Ex. Liste des produits jamais commandés

```
SELECT    noProduit, libellé
FROM      Produit
WHERE     noProduit NOT IN
          (SELECT DISTINCT noProduit
           FROM      Commande)
```

Sous-requêtes en SQL (4)

Le Prédicat **ALL** ou **ANY** permet de tester si un prédicat de comparaison est vrai pour tous (ALL) ou au moins un (ANY) des résultats d'une sous-requête

Ex. Donner les noms des clients ayant acheté un produit en quantité supérieure à chacune des quantités de produits achetées par le client 'c1'

```
SELECT  C.noClient
FROM    Commande C
WHERE   C.quantité >= ALL
        (SELECT W.quantité
         FROM   Commande W
         WHERE  W.noclient = 'C1')
```

Requêtes quantifiées : Prédicat EXISTS

Le prédicat **EXISTS** permet de tester si le résultat d'une sous-requête est non vide.

Ex. Donner les noms des produits qui n'ont pas été achetés

Produits non achetés = DIFFERENCE(tous les produits, produits achetés)

```
SELECT noProduit, libellé  
FROM   produit P  
WHERE  NOT EXISTS  
      ( SELECT *  
        FROM commande  
        WHERE Commande.noProduit = P.noProduit )
```


Requêtes quantifiées : Prédicat EXISTS

Les prédicats **ALL** et **ANY** sont redondants, ils peuvent s'exprimer avec EXISTS

```
X * ANY  
(SELECT Y  
  FROM   T  
  WHERE  p)
```

⇔

```
EXISTS  
(SELECT *  
  FROM   T  
  WHERE  p AND x * T.Y )
```

```
X * ALL  
(SELECT Y  
  FROM   T  
  WHERE  p)
```

⇔

```
NOT EXISTS  
(SELECT *  
  FROM   T  
  WHERE  p AND NOT (x * T.Y) )
```

Où * est un prédicat de comparaison (=, <>, <, <=, >, >=)

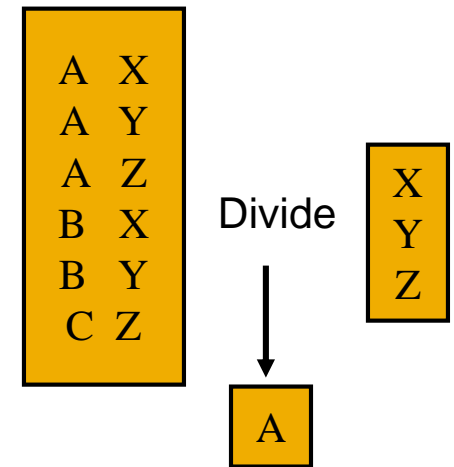
Opération de DIVISION

- **Exemple** : Liste des produits qui ont été achetée **par tous** les clients de Metz => $\forall x \mid P(x)$
Sur le schéma: X, Y, Z sont les clients de Metz et A, B, C sont les produits.

$$\forall x \mid P(x) \Leftrightarrow \neg (\exists x \mid \neg P(x))$$

Dans le résultat : un produit est sélectionné **s'il n'existe pas de client de Metz qui n'a pas acheté ce produit.**

```
SELECT noProduit
FROM   Produit P
WHERE  NOT EXISTS (SELECT *
                    FROM CLIENT
                    WHERE adresse = 'METZ' AND noClient NOT IN
                        (SELECT noClient
                         FROM Commande C, Ligne_Com LC
                         WHERE C.numCom = LC.numCom
                              AND LC.noProduit = P.noProduit))
```



Opération ensembliste d'UNION

SQL permet d'exprimer l'opération d'union en connectant des SELECT par des **UNION**

Ex. Donner les noms des produits du fournisseur IBM ou ceux achetés par le client no 'c1'

```
SELECT P.noProduit
FROM Produit P
WHERE P.noFournisseur = 'IBM'
UNION
SELECT LC.noProduit
FROM Commande C, Ligne_Commande LC
WHERE C.noProduit=LC.noProduit AND C.noClient = 'c1'
```

L'union élimine les doublons

Pour garder les doublons il faut spécifier ALL après UNION :

X UNION ALL Y

Opérations ensemblistes : INTERSECT, EXCEPT (DIFFERENCE)

- Sous ORACLE, **MINUS** au lieu de **EXCEPT**

Ex. Trouver les noms et prénoms des employés qui sont aussi des passagers

Employé

| noEmployé | nomEmp | prénomEmp |
|-----------|---------|-----------|
| 10 | Henry | John |
| 15 | Conrad | James |
| 35 | Jenqua | Jessica |
| 46 | Leconte | Jean |

Passager

| noPassager | nomPass | prénomPass |
|------------|---------|------------|
| 4 | Harry | Peter |
| 78 | Conrad | James |
| 9 | Land | Robert |
| 466 | Leconte | Jean |

```
(SELECT nomEmp as nom, prénomEmp as prénom  
FROM Employé)
```

INTERSECT

```
(SELECT nomPass as nom, prénomPass as prénom  
FROM Passager)
```

| nom | prénom |
|---------|--------|
| Conrad | James |
| Leconte | Jean |

Grouper les relations (GROUP BY)

- Il est possible de partitionner une relation en groupes selon les valeurs de une ou plusieurs colonnes (attributs) : **GROUP BY**
- Toutes les lignes d'un même groupe ont la même valeur pour la liste des attributs de partitionnement spécifiés après GROUP BY

Exemple. *Nombre de produits commandés par client*

```
SELECT  noClient, COUNT(*) AS totalProduits
FROM    Commande
GROUP BY noClient
```

- 1) Les commandes sont groupées par numéro de client
- 2) pour chaque groupe, afficher le numéro du client concerné par le groupe et le nombre de commandes.

Attention : Chaque expression du SELECT doit avoir une valeur **unique** par **groupe**

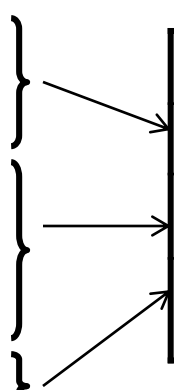
Grouperment de relations (GROUP BY)

ex. Nombre de produits commandés par client

```
SELECT  noClient, COUNT(*) AS totalProduits
FROM    Commande
GROUP BY noClient
```

| noProduit | dateCommande | noClient |
|-----------|--------------|----------|
| 4 | 05-JAN-03 | 10 |
| 5 | 05-JAN-03 | 10 |
| 20 | 14-MAY-03 | 12 |
| 28 | 15-AUG-03 | 12 |
| 68 | 15-AUG-03 | 12 |
| 59 | 20-SEP-03 | 15 |

| noClient | totalProduits |
|----------|---------------|
| 10 | 2 |
| 12 | 3 |
| 15 | 1 |



Groupement de relations (GROUP BY)

Ex. *Quantité totale de produits commandés par client en dehors du produit F565*

```
SELECT    noClient, SUM(quantité)
FROM      Commande
WHERE     noProduit <> 'F565'
GROUP BY  noClient
```

- 1) Les tuples de Commande ne vérifiant pas la condition sont exclus
- 2) Les commandes restantes sont groupées par numéro de client
- 3) pour chaque groupe, afficher le numéro du client concerné par le groupe et la somme des quantités.

Une clause **HAVING** permet de spécifier une condition de restriction des groupes

Grouperement de relations (GROUP BY)

Ex. *Quantité moyenne commandée par produit pour les produits ayant fait l'objet de plus de 3 commandes. Ignorer les commandes concernant le client C47.*

```
SELECT      noProduit, AVG(quantité)
FROM        Commande
WHERE      noClient != 'C47'
GROUP BY  noProduit
HAVING    COUNT(*) > 3
```

- 1) Les tuples de Commande ne vérifiant pas la condition WHERE sont exclus
- 2) Les commandes restantes sont groupées par numéro de produit
- 3) pour chaque groupe, compter le nombre d'éléments et éliminer les groupes à moins de 3 éléments.
- 4) pour les groupes restants, afficher le numéro de produit et la quantité moyenne.

N.B. : La clause HAVING ne s'utilise qu'avec un GROUP BY.

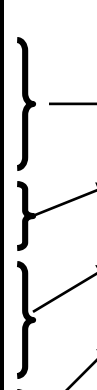
Grouper les relations (GROUP BY)

- Il est possible de partitionner sur plusieurs colonnes (attributs)

ex. *Nombre de produits commandés par client et par date*

```
SELECT    noClient, dateCommande, COUNT(noProduit) AS
nbProduits
FROM      Commande
GROUP BY  noClient, dateCommande
```

| noProduit | dateCommande | noClient |
|-----------|--------------|----------|
| 4 | 05-JAN-03 | 10 |
| 5 | 05-JAN-03 | 10 |
| 20 | 14-MAY-03 | 12 |
| 28 | 15-AUG-03 | 12 |
| 68 | 15-AUG-03 | 12 |
| 59 | 20-SEP-03 | 15 |



| noClient | dateCommande | nbProduits |
|----------|--------------|------------|
| 10 | 05-JAN-03 | 2 |
| 12 | 14-MAY-03 | 1 |
| 12 | 15-AUG-03 | 2 |
| 15 | 20-SEP-03 | 1 |

Ordonner les résultats d'une requête (ORDER BY)

- Possibilité de trier les résultats d'une requête par rapport à une ou plusieurs de ses colonnes

```
SELECT colonne(s)
FROM relation(s) [WHERE condition]
ORDER BY colonne1 ASC, colonne2 ASC ...
```

Où

ASC : ordre ascendant (par défaut)

DESC : ordre descendant

ex. liste des commandes par ordre croissant du numéro de client et par ordre chronologique inverse de la date

```
SELECT      *
FROM        Commande
ORDER BY   noClient, dateCommande desc
```

Forme générale de SELECT

| | |
|-------------------|----------------------------------|
| SELECT [DISTINCT] | Liste des attributs, expressions |
| FROM relation(s) | liste des tables |
| WHERE condition] | qualification |
| GROUP BY | attributs de partitionnement |
| HAVING | qualification de groupe |
| ORDER BY | liste de colonnes de tri |

Exemple. Donner les noms, les prix, les marques et la quantité maximum vendue de tous les produits HP, Apple ou Dell dont la quantité totale vendue est supérieure à 500 et dont les quantités vendues sont > 10

| | |
|----------|--|
| SELECT | P.IdPro, P.prix, P.marque, 'Qte max vendue = ', MAX (V.qte) |
| FROM | produit P , vente V |
| WHERE | P.IdPro = V.IdPro |
| AND | P.marque IN ('IBM', 'Apple', 'Dell') AND V.qte > 10 |
| GROUP BY | P.IdPro, P.prix, P.marque |
| HAVING | SUM (V.qte) > 500 |

Forme générale de SELECT

Le résultat d'un SELECT est construit suivant les étapes :

1. **FROM** : la clause FROM est évaluée de manière à produire une nouvelle table, produit cartésien ou jointure des tables dont le nom figure après FROM

2. **WHERE** : le résultat de l'étape 1 est réduit par élimination de toutes les lignes qui ne satisfont pas à la clause WHERE

3. **GROUP BY** : le résultat de l'étape 2 est partitionné selon les valeurs des colonnes dont le nom figure dans la clause GROUP BY

Dans le dernier exemple, les colonnes sont P.IdPro, P.prix et P.marque ; en théorie il suffirait de prendre uniquement P.IdPro comme colonne définissant les groupes (puisque le prix et la marque sont déterminés par le no de produit) SQL oblige de faire apparaître dans la clause GROUP BY toutes les colonnes qui sont mentionnées dans la clause SELECT

4. **HAVING** : les groupes ne satisfaisant pas la condition HAVING sont éliminés du résultat de l'étape 3

5. **SELECT** : chacun des groupes génère une seule ligne du résultat

Le langage SQL (*SGBD ORACLE*)

- I. Interrogation des données (SELECT/ sous-requêtes, GROUP BY)
- II. **Mise à jour des données (INSERT, UPDATE, DELETE)**

Mise à jour des données

a. Insertion de tuples (INSERT)

```
INSERT INTO nom-relation [(attribut1, attribut2...)]  
{VALUES (valeur1[, valeur2, ...] )} | {commande-SELECT}
```

Ajout d'un produit

```
INSERT INTO Produit (nop, nomp, prix)  
  VALUES (430, 'lecteur DVD', 9.99);
```

Ajout du résultat d'un SELECT

```
INSERT INTO Commande (noClient, noProduit)  
  SELECT noClient, noProduit FROM Produit, Client;
```

- Les attributs non mentionnés sont positionnés à NULL
(si pas de valeur par défaut)

Mise à jour des données

b. Modification de tuples (UPDATE) (1/2)

```
UPDATE nom_relation
SET attribut1 = {expression1 | NULL | clause-SELECT1}
    [, attribut2 = {expression2 | NULL | clause-SELECT2}
    ...]
[WHERE condition ]
```

- *Positionner à 'Durand' le nom du client n°3*

```
UPDATE Client SET nom = 'Durand' WHERE noClient=3
```

- *Augmenter de 5% le prix des produits dont le libellé est dans une liste*

```
UPDATE Produit
SET prixUnitaire = prixUnitaire*1.05
WHERE libellé IN ('CD-ROM', 'DVD', 'ZIP')
```

Mise à jour des données

b. Modification de tuples (UPDATE) (2/2)

- *Augmenter de 10% les prix des produits fournis par le fournisseur 'FFF'*

```
UPDATE Produit p SET      prixUnitaire = prixUnitaire*1.1
WHERE   p.noFournisseur IN
          (SELECT      f.noFournisseur
           FROM        Fournisseur f
           WHERE       f.raisonSociale = 'FFF');
```

- *Positionner le libellé du produit n° 99 à 'produitTest' et son prix au prix moyen*

```
UPDATE   Produit
SET
libellé = 'produitTest' ,
prixUnitaire = (SELECT AVG(prixUnitaire) FROM Produit)
WHERE noProduit = 99;
```


Mise à jour des données

c. Suppression de tuples (DELETE)

DELETE [**FROM**] relation [**WHERE** condition]

- *Supprimer les clients de Metz :*

```
DELETE Client WHERE ville = 'Metz'
```

- *Supprimer tous les tuples de la table Client :*

```
DELETE Client
```

- ✓ suppression de **tous** les tuples de la table Client
- ✓ le **schéma** de la table Client existe toujours (dans le dictionnaire)