Base de données

Séance 6

Algèbre relationnelle et SQL :

- LMD : SELECT
- Fonctions de calcul
- Agrégats
- Sous-requêtes

SELECT : extraction de données

- Récupérer les informations stockées dans la base de manière structurée
- Syntaxe générale :

```
SELECT
   { <col>[, <col>… ] | *}
FROM
   <nom_table>
[WHERE <expr>]
```

Forme simple du SELECT

- Récupérer **tous** les enregistrements de la table 'ma_table'

```
SELECT * FROM ma_table;
```

- Algèbre relationnelle :
  toutes les occurrences de ‘ma_table’

Projection avec SELECT

- Récupérer seulement les colonnes c1 et c2 :

```
SELECT c1, c2 FROM ma_table;
```

- Garder les éventuels doublons (non ensembliste)

Unicité avec DISTINCT

- Projection en éliminant les doublons:

```
SELECT DISTINCT c1, c2
FROM ma_table;
```

- Algèbre relationnelle :

$\Pi_{C1,C2}$ (ma_table)

Restriction avec la clause WHERE

- Ajout d'un prédicat avec WHERE :

```
SELECT *
FROM ma_table
WHERE <predicat>;
```

- Exemple :

```
SELECT *
FROM ma_table
WHERE c1 = 600;
```

- Algèbre relationnelle : $\sigma_{C1=600}(ma\_table)$

Les formes de la clause WHERE

- Opérateurs de comparaison :

  = , <> , <= , >= , < , >

- Opérateurs logiques

  AND, OR, NOT

- Exemple :

```
SELECT *
FROM ma_table
WHERE c1 <> 12 AND c2 = 15;
```

Le cas particulier de la nullité

- Cas particulier des champs 'NULL' :

  Il sont testés avec le prédicat IS NULL ou IS NOT NULL.

- Exemple :

```
SELECT *
FROM ma_table
WHERE c2 IS NULL;
```

L'opérateur LIKE

- L'opérateur LIKE permet de tester les chaînes de caractères avec l'expression d'un motif (pattern)

- Les méta-caractères sont :

  _ : un seul caractère

  % : zéro ou plusieurs caractères

L'opérateur LIKE : exemples

- Toutes les personnes dont le nom commence par 'A' :

```
SELECT *
FROM personne
WHERE nom LIKE 'A%' ;
```

- Tous les prénoms dont le nom contient un 'X' en deuxième position.

```
SELECT prenom
FROM personne
WHERE nom LIKE '_X%' ;
```

Autres opérateurs de caractères

- Concaténation : || ou CONCAT

```
SELECT first_name || ' ' || last_name
FROM …
SELECT CONCAT(first_name,' ',last_name)
FROM …
```

- Sous-chaînes :

```
SUBSTR(chaîne, pos, long)
```

ATTENTION : En SQL les chaînes de caractères sont indicées à partir de 1

Conversion de caractères

- En majuscule : UPPER(chaîne)
- En minuscule : LOWER(chaîne)
- En nombre : TO_NUMBER(chaîne)

Conversion nombre/texte

- Nombre => chaîne :

  TO_CHAR(nb, masque)

- **Le masque** définit des règles de transformation :
  - 9 : chiffre représenté sauf 0 non significatif
  - 0 : chiffre représenté
  - D : séparateur décimal apparent (, ou .)
  - G : séparateur de groupe (. ou ,)
  - C : symbole ISO de monnaie locale

- Exemple : TO_CHAR(1555444.2, '999G999G999D99C')

  => 1.555.444,20EUR

Conversion de date

- chaîne => date : TO_DATE(chaîne, format)
- date => chaîne : TO_CHAR(date, format)
- Le **format** définit les parties utilisées et leur ordre
- Exemple : lundi 17 novembre 2008
  - Année : YYYY, YY, YEAR
    - 2008, 08, TWO THOUSAND EIGHT (non traduit)
  - Mois : MM, Mon, Month
    - 11, Nov, Novembre
  - Semaine : WW (année), W (mois)
    - 46, 3
  - Jour : DDD (année), DD (mois), D (semaine), DAY, DY
    - 322, 17, 1, Lundi, LU

Conversion de date

- Exemple : 14H53’45
  - Heures : HH, HH12, AM, PM
    - 14, 2, pm, pm
  - Minutes : MM
    - 53
  - Secondes : SS
    - 45
- Séparateurs
  - - / . , ; : espace
  - "a" : entre quotes pour les autres caractères
  - Exemple : 'HH"h"MM:SS' => 14h53:45

Tri avec ORDER BY

- Syntaxe :

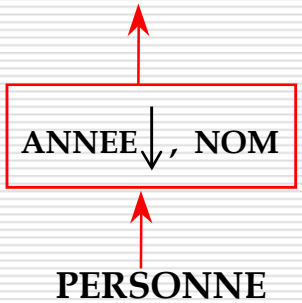
```
SELECT ...
FROM ma_table
[WHERE ...]
ORDER BY COL1 [ASC|DESC]
         [, COL2 [ASC|DESC]*
```

Tri avec ORDER BY : exemple

- Tri de personne par année décroissante puis par nom dans l'ordre alphabétique :

```
SELECT *
FROM PERSONNE
ORDER BY ANNEE DESC, NOM
```

- Algèbre relationnelle :



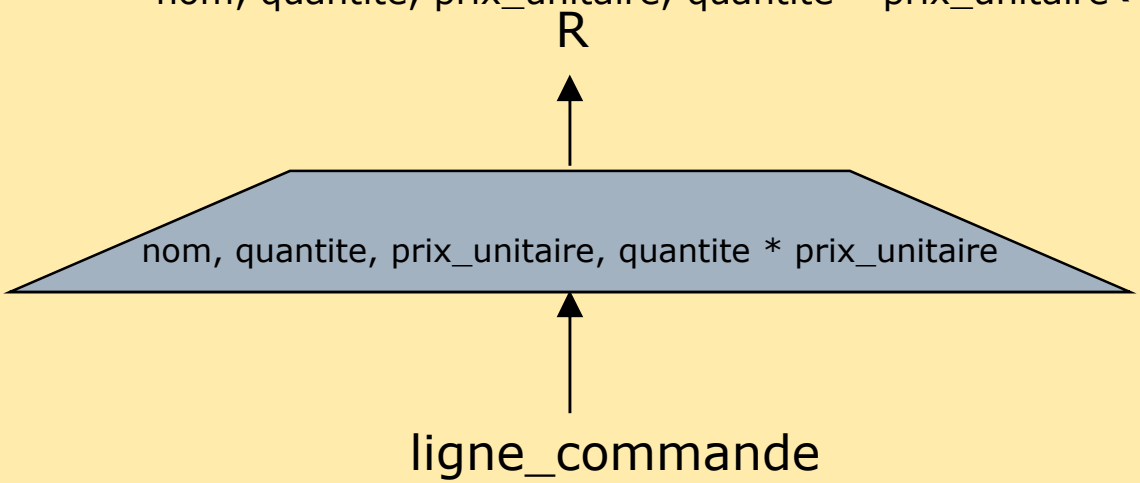
Fonction sur un n-uplet

- Utilisation d'une fonction pendant :
  - une projection : champs calculé
    => extension de l'opérateur Π
  - une restriction : dans un prédicat
- Fonctions disponibles
  - mathématiques : +, -, * , /
  - chaînes de caractères
  - dates

Exemple de calcul avec projection

- Montant total d'une ligne de commande (chaque ligne de commande corresponde à un produit) :

$R = \Pi_{nom,\ quantite,\ prix\_unitaire,\ quantite * prix\_unitaire}(ligne\_commande)$



algèbre relationnelle

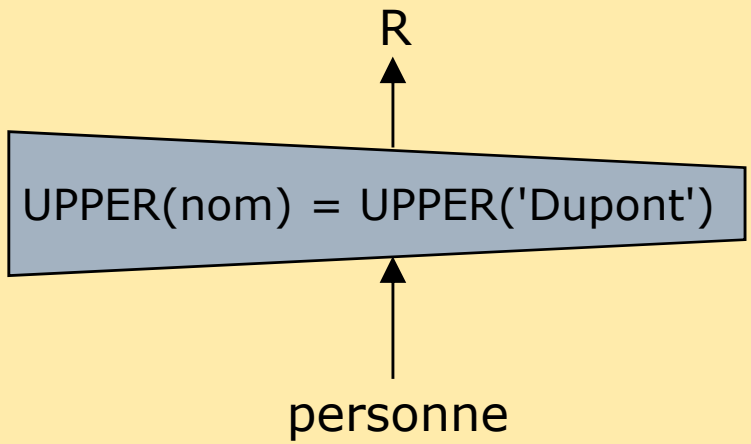
```
SELECT nom, quantite, prix_unitaire, quantite * prix_unitaire
FROM ligne_commande;
```

SQL

Exemple de calcul avec restriction

- Recherche d'après un critère saisi dans un formulaire indépendamment de la casse :

$R = \sigma_{UPPER(nom) = UPPER('Dupont')}(personne)$



algèbre relationnelle

SQL

```
SELECT * FROM personne WHERE UPPER(nom) = UPPER('Dupont');
```

Fonction sur une colonne

- Calcul sur toutes les occurrences d'un attribut (une colonne) :
  - AVG() : moyenne
  - SUM() : somme
  - MIN() : minimum
  - MAX() : maximum
  - COUNT() : dénombrement
  - VARIANCE() : variance
  - STDDEV() : écart-type

Fonction sur une colonne

- Syntaxe algèbre relationnelle :

$$R2 = F_{fonction([attribut])}(R1)$$

- Renvoie une relation R2 avec une seule ligne contenant le résultat de la fonction appliquée à toutes les occurrences de l'attribut dans R1.
- Les valeurs nulles sont ignorées

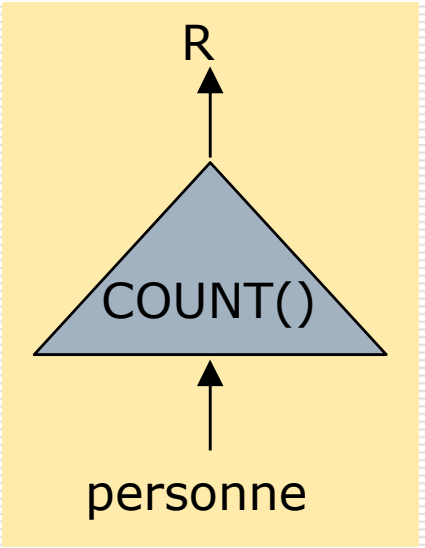
- Equivalent SQL :

```
SELECT fonction(un_attribut) FROM ma_table;
```

Fonction sur une colonne : exemple 1

$R = F_{count()}(personne)$

```
SELECT COUNT(*) FROM personne;
```



Résultat :

personne	id	Nom	Prenom
	1	DERAY	Odile
	2	DUPONT	Emile
	3	DURAND	Norbert

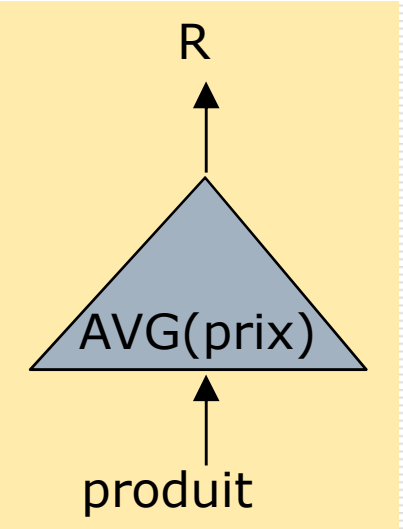
→

R	count
	3

Fonction sur une colonne : exemple 2

$R = F_{AVG(prix)}(produit)$

SELECT AVG(prix) FROM produit;



Résultat :

produit	libelle	prix
	chaise	25
	table	225
	lampe	20

→

R	avg
	90

Renommage d'attributs

Soit une relation R(A1,A2,...,An).

- But :
  - Lever une ambigüité avec un attribut d'une autre relation
  - Renommer le résultat d'un calcul
- Solutions :
  - Préfixer un attribut par le nom de la relation : R.A1
  - Renommer un attribut avec :
    - opérateur $\rho_{(B1,B2,...,Bn)}R$ en algèbre relationnelle
    - le mot-clé AS en SQL

Exemple de renommage

$\rho_{(nom,\ quantite,\ prix\_unitaire,\ prix\_total)}($
$\quad \Pi_{nom,\ quantite,\ prix\_unitaire,\ quantite*prix\_unitaire}($
$\quad\quad$ ligne_commande
$\quad)$
$)$

```
SELECT nom, quantite, prix_unitaire,
    quantite*prix_unitaire AS prix_total
FROM ligne_commande;
```

Renommage de relation

Soit une relation R(A1,...,An).

- Renommer la relation R en une relation E : $\rho_E(R)$
- Renommer la relation R et ses attributs :

  $\rho_{E(B1,..., Bn)}(R)$

- En SQL :

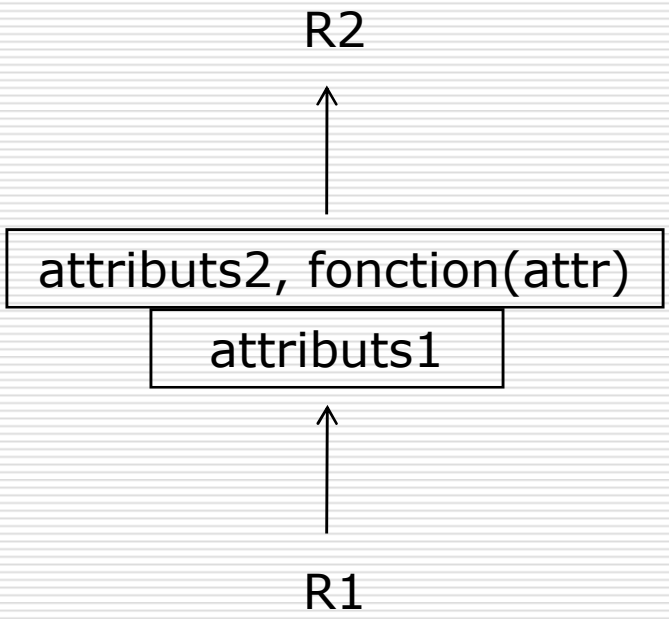
```
SELECT p.nom FROM personne p;
```

Agrégats

- Partitionnement horizontal d'une relation selon les valeurs d'un groupe d'attributs appelés facteurs de groupage
- Suivi d'un regroupement par une fonction de calcul en colonne (SUM, MIN, MAX, AVG, COUNT, …)
- La relation résultat contient une ligne par partition avec :
  - des facteurs de groupage (valeur unique par partition)
  - le résultat du calcul
- $R2 = {}_{attributs1}F_{attributs2,fonction(attr)}(R1)$
  (attributs2 est un sous-ensemble d'attributs1)

Agrégat : arbre algèbrique

- $R2 = {}_{attributs1}F_{attributs2,fonction(attr)}(R1)$



Exemple 1 : calculer le degré moyen de vins par cru

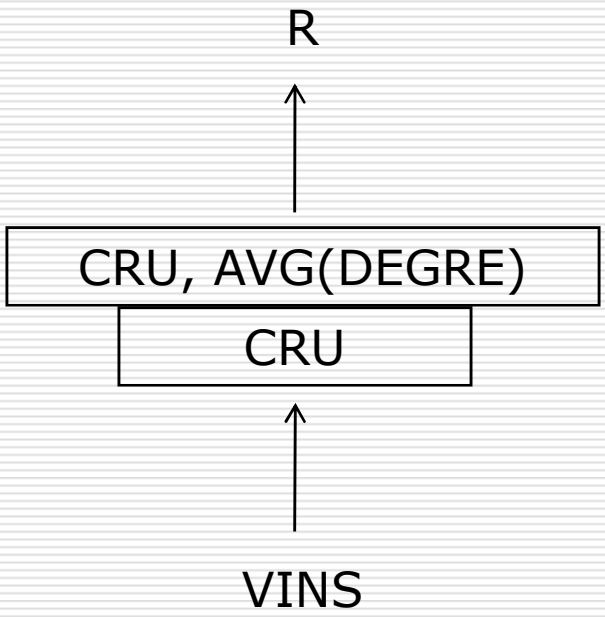
VINS	CRU	MILL	DEGRE	QUANTITE
	CHABLIS	1977	10.9	100
	CHABLIS	1987	11.9	250
	VOLNAY	1977	10.8	400
	VOLNAY	1986	11.2	300
	MEDOC	1985	11.2	200

$$_{cru}F_{CRU,AVG(DEGRE)}(VINS)$$

AVG	CRU	AVG(DEGRE)
	CHABLIS	11.4
	VOLNAY	11.0
	MEDOC	11.2

Exemple 1 : arbre algèbrique

- $R = {}_{CRU}F_{CRU,AVG(DEGRE)}(VINS)$



Exemple 2 : calculer la somme de quantité pour chaque cru

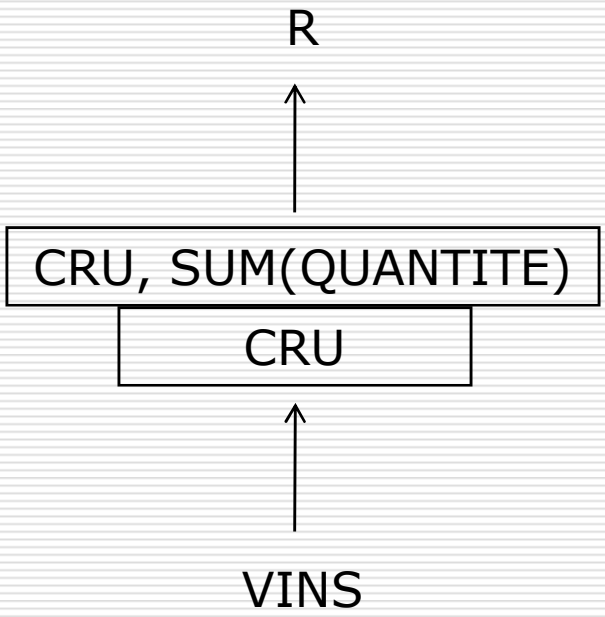
VINS	CRU	MILL	DEGRE	QUANTITE
	CHABLIS	1977	10.9	100
	CHABLIS	1987	11.9	250
	VOLNAY	1977	10.8	400
	VOLNAY	1986	11.2	300
	MEDOC	1985	11.2	200

$$_{CRU}F_{CRU,SUM(QUANTITE)}(VINS)$$

SUM	CRU	SUM(QUANTITE)
	CHABLIS	350
	VOLNAY	700
	MEDOC	200

Exemple 2 : arbre algèbrique

- $R = {}_{CRU}F_{CRU,SUM(QUANTITE)}(VINS)$



Syntaxe SQL de groupage

- Pour regrouper des données, il faut utiliser la clause GROUP BY suivi du facteur de groupage.

```
SELECT
    <col1>, <col2>, <fonction_agrégat>(<col3>)
FROM
    ...
WHERE
    ...
GROUP BY
    <col1>, <col2>;
```

Groupage : exemple 1

- Calculer les quantités récoltés pour chaque producteur.

```
SELECT
   numprod, SUM(quantite)
FROM
   recolte
GROUP BY
   numprod ;
```

Groupeur

Groupage : exemple 2

- Calculer le nombre de films dramatiques réalisés par chaque réalisateur.

Film(num_film, *num_ind*, titre, genre, annee)

```
SELECT num_ind, COUNT(*)
FROM Film
WHERE genre = 'Drame'
GROUP BY num_ind;
```

Groupage : la clause HAVING

- Elle permet d'opposer un prédicat à un groupe.

```
SELECT
   num_client,
   SUM(montant)
FROM
   commandes
GROUP BY
   num_client
HAVING
   SUM(montant) > 1000;
```

La clause **HAVING** élimine des groupes comme la clause **WHERE** élimine des lignes.

- HAVING s'applique sur le résultat de la requête.

WHERE ou HAVING ?

- Question 1 : Calculer pour chaque cinéma le nombre de projections dont la date est antérieure à '2000-01-01'.

- Question 2 : Quels sont les cinémas dont le nombre de projections est supérieur ou égal à 4 ?

  Projection(*num_cine, num_film*, pdate)

WHERE ou HAVING ?

- Réponse 1 :

```
SELECT num_cine, COUNT(*)
FROM Projection
WHERE pdate < DATE '2000-01-01'
GROUP BY num_cine;
```

- Réponse 2 :

```
SELECT num_cine, COUNT(*)
FROM Projection
GROUP BY num_cine
HAVING COUNT(*) >= 4;
```

Sous-requêtes SQL

- Une sous-requête permet de faire une requête sur la base du résultat d'une autre requête.
- Le résultat peut être utilisé à la place d'une constante dans un calcul ou un prédicat
- Le résultat de la sous-requête peut renvoyer :
  - une ligne contenant :
    - une valeur
    - un n-uplet
  - une liste de lignes
  - aucun résultat

Sous-requêtes renvoyant une valeur

- Sélection par comparaison avec une constante :

```
WHERE poste = 'Manager'
```

- Est équivalent à :

```
WHERE poste = (SELECT poste from ...)
```

- Accepte tous les opérateurs tels que <, >, <=, >=, <>

Sous-requêtes renvoyant une valeur

```
SELECT
   nom
FROM
   employe
WHERE
   poste = (SELECT poste
               FROM employe
               WHERE nom = 'Martin') ;
```

Renvoie tous les employés qui ont le même poste que Martin

Note : On considère que nom est unique

Sous-requêtes sur colonnes multiples

```
SELECT
    nom
FROM
    employe
WHERE (poste, salaire) =
          (     SELECT poste, salaire
                FROM employe
                WHERE nom = 'Martin') ;
```

Renvoie tous les employés qui ont le même poste et le même salaire que Martin

Note : On considère que nom est unique

Sous-requêtes renvoyant une liste

- …**WHERE** poste **IN** (**SELECT poste FROM ..**);
  - La valeur doit être trouvée dans le résultat de la sous requête
  - On peut utiliser **NOT IN**
- …**WHERE** numero **> ALL** (**SELECT numero FROM ..**);
  - La valeur testée doit être supérieure à toutes les valeurs ramenées par la sous requête
- …**WHERE** poste **> ANY** (**SELECT poste FROM ..**);
  - La valeur testée doit être supérieure à au moins une valeur obtenue par la sous requête

Sous-requêtes renvoyant une liste

- opérateur **IN**, **NOT IN**

- opérateur simple **=, <>, <, >, <=, >=** suivi de **ALL** ou **ANY.**
  - = ANY est équivalent à IN
  - != ALL est équivalent à NOT IN

Sous-requêtes renvoyant une liste

- Exemple

```
SELECT
   nom
FROM
   client
WHERE num_client
IN (SELECT num_client
   FROM commande
   WHERE date_commande = '05-JUN-98');
```

Les sous-requêtes et requêtes imbriquées

- Plusieurs sous-requêtes peuvent être imbriquées
- Une sous-requête peut être dépendante ou indépendante de la requête principale
  - Une sous-requête indépendante si elle n'utilise aucun attribut de la requête principale (attention aux ambigüités)
  - Les requêtes dépendantes seront vues ultérieurement (cours 8)