Deuxième Partie

MODELE ET OPERATIONS

Partie II
Sommaire

- Algèbre relationnelle
  - Le modèle relationnel
  - Opérations ensemblistes
  - Opérations spécifiques
  - Opérations dérivées
  - Expressions relationnelles et requêtes

- Requêtes SQL
  - Syntaxe
  - Conditions
  - Expressions et fonctions
  - Jointures
  - Opérations ensemblistes
  - Tri
  - Groupements
  - Exercice

Algèbre relationnelle

- deux familles d'opérations (ensemblistes et unaire)
- Le modèle relationnel impose la forme des données stockées
- Les notions de base sont : relations (tables), attributs (colonnes), domaines (types)
- Les requêtes sont des opérations sur ces objets : union, intersection, projection, etc...
- Objets et opérations forment une algèbre : un système de calcul, avec certaines propriétés
- SQL est un langage de description de ces objets et de ces opérations, de cet algèbre.

Algèbre relationnelle
Le modèle relationnel

- Le modèle relationnel décrit :
  - des objets élémentaires : données, relations, attributs
  - des opérations sur ces objets : union, jointure, etc.

- Une relation est :
  - définie par un schéma : un ensemble d'attributs typés
  - constituée d'un ensemble de n-uplets (ou tuple)

Par exemple :

```
Personne(nom,sexe,ddn, dept) = {
(Jean,M,01/04/1965,68),
(Anne,F,10/09/1968,67),
(Paul,M,22/03/1965,90)}
```

Algèbre relationnelle
Le modèle relationnel (suite)

- Chaque attribut a un domaine : l'ensemble des valeurs qu'il peut prendre

  Exemple :

  Domaine(nom) = A80

  Domaine(sexe) = {M,F}

  Domaine(ddn) = D

  Domaine(dept) = {1-19,2A,2B,21-95,971-974}

- Les domaines sont finis, ils ont un nombre fixé de valeurs possibles (éventuellement très grand), donc
  - une relation a une taille maximale connue

Algèbre relationnelle
Opérations ensemblistes

- Les opérations ensemblistes sur les relations sont la traduction des opérations sur des ensembles de n-uplets

- Dans tous les cas, les deux relations doivent avoir le même domaine

$$Domaine(R1 \cup R2)$$
$$Domaine(R1 \cap R2) \quad => Domaine(R1)=Domaine(R2)$$
$$Domaine(R1 - R2)$$

- L'union consiste à fusionner deux relations

$$R1 \cup R2 = \{t \mid t \in R1 \text{ ou } t \in R2\}$$

(note : pas de doublon dans le résultat)

Algèbre relationnelle
Opérations ensemblistes (suite)

- La différence consiste à conserver les n-uplets de la première relation qui ne sont pas dans la seconde

$$R1 - R2 = \{ t \mid t \in R1 \text{ et } t \notin R2\}$$

- L'intersection consiste à ne garder que les n-uplets communs

$$R1 \cap R2 = \{t \mid t \in R1 \text{ et } t \in R2\}$$

- On peut écrire l'intersection à l'aide de l'union et de la différence :

Algèbre relationnelle
Opérations ensemblistes (suite)

- Le produit cartésien combine deux relations de domaine éventuellement différents

$$R1 \times R2 = \{t1 \,.\, t2 \mid t1 \in R1 \text{ et } t2 \in R2\}$$

(ou *t1. t2* est la concaténation des tuplets *t1* et *t2*)

Plat
Viande
Poisson

x

Vin
Rouge
Blanc

=

Plat	***Vin***
Viande	*Rouge*
Viande	*Blanc*
Poisson	*Rouge*
Poisson	*Blanc*

- Le produit cartésien est fondamental pour les requêtes

Algèbre relationnelle
Opérations spécifiques

- Pour le traitement de données, on a besoin de quelques opérations spécifiques pour :
  - filtrer (sélection)
  - réduire (projection)
  - associer (jointure)

- La sélection permet de ne conserver que les n-uplets qui répondent à une condition booléenne

$$\sum_{Cond}(R) = \{t \in R \mid Cond(t) \text{ est vraie}\}$$

- La condition porte sur les attributs

Algèbre relationnelle
Opérations spécifiques (suite)

- Exemple de sélection :

$\sum_{\text{sexe=M}}$ (

Nom	Sexe	Dept
Jean	M	68
Anne	F	67
Paul	M	90

) =

Nom	Sexe	Dept
Jean	M	68
Paul	M	90

- La projection consiste à supprimer certains attributs

$\Pi_{\text{\{nom,dept\}}}$ (

Nom	Sexe	Dept
Jean	M	68
Anne	F	67
Paul	M	90

) =

Nom	Dept
Jean	68
Anne	67
Paul	90

- La sélection conserve le domaine, pas la projection

Algèbre relationnelle
Opérations spécifiques (suite)

- La jointure permet de combiner deux relations, puis de sélectionner certains n-uplets

$$R1 \underset{Cond}{\Diamond\Diamond} R2 = \underset{Cond}{\sum} R1 \times R2$$

- Souvent, la condition est une égalité sur deux attributs pris dans chacune des relations (equi-jointure)

Nom_1	Sexe
Jean	M
Anne	F
Paul	M

$\underset{Nom_1=nom_2}{\Diamond\Diamond}$

Nom_2	Sport
Jean	Foot
Anne	Velo

=

Nom	Sexe	Nom	Sport
Jean	M	Jean	Foot
Anne	F	Anne	Velo

Algèbre relationnelle
Opérations spécifiques (suite)

- La jointure permet de parcourir les données d'une relation à l'autre

Nom	Num
Jean	42
Anne	43
Paul	44

◊◊
num=net

Net	Cid
42	BDD
43	POO
43	BDD
44	POO

◊◊
cid=code

Titre	Code
Bases..	BDD
Prog..	POO

=

Nom	Num	Net	Cid	Code	Titre
Jean	42	42	BDD	BDD	Bases..
Anne	43	43	POO	POO	Prog..
Anne	43	43	BDD	BDD	Bases..
Paul	44	44	POO	POO	Prog..

Algèbre relationnelle
Opérations spécifiques (suite)

- La condition d'une jointure peut cependant être quelconque

Nom	Note
Jean	11
Anne	15
Paul	12

◊◊

note>=min et note<max

Min	Max	Mt
10	12	P
12	14	AB
14	16	B
16	20	TB

=

Nom	Note	…	Mt
Jean	11	…	P
Anne	15	…	B
Paul	12	…	AB

Algèbre relationnelle
Opérations spécifiques (suite)

- Une auto-jointure est une jointure d'une relation avec elle-même

Nom_1	$Sexe_1$
Jean	*M*
Anne	*F*
Paul	*M*

◊◊ $sexe_1 = sexe_2$

Nom_2	$Sexe_2$
Jean	*M*
Anne	*F*
Paul	*M*

=

Nom_1	$Sexe_1$	Nom_2	$Sexe_2$
Jean	*M*	*Jean*	*M*
Jean	*M*	*Paul*	*M*
Anne	*F*	*Anne*	*F*
Paul	*M*	*Jean*	*M*
Paul	*M*	*Paul*	*M*

Algèbre relationnelle
Opérations dérivées

- Les opérations de base permettent de définir des opérations dérivées, plus complexes/utiles/... (?)
- La jointure externe permet de conserver les n-uplets d'une relation même s'ils n'ont pas de correspondant dans l'autre relation
- La semi-jointure est une jointure suivie d'une projection sur les attributs de la première relation

Algèbre relationnelle
Opérations dérivée (suite)

- La division est « inverse » de la jointure (répond à la question quelque soit x trouver y)

Exemple

R

etudiant	*cours*	*reussi*
Francois	***BDD***	***Oui***
Francois	***Prog***	***Oui***
Jacques	*BDD*	*Oui*
Jacques	*Math*	*Oui*
Pierre	*Prog*	*Oui*
Pierre	*BDD*	*non*

V

Cours	*réussi*
Prog	*Oui*
BDD	*Oui*

R division V

etudiant
Francois

Algèbre relationnelle
Expressions relationnelles et requêtes

- Les requêtes possibles sont toutes les expressions relationnelles correctes

- Une expression relationnelle :
  - est soit le nom d'une relation existante
  - est soit une opération sur une ou deux expressions relationnelles
  - utilise les opérateurs relationnels (=,<,>,<=,>=,<>,…)

- Exemple : on a les relations

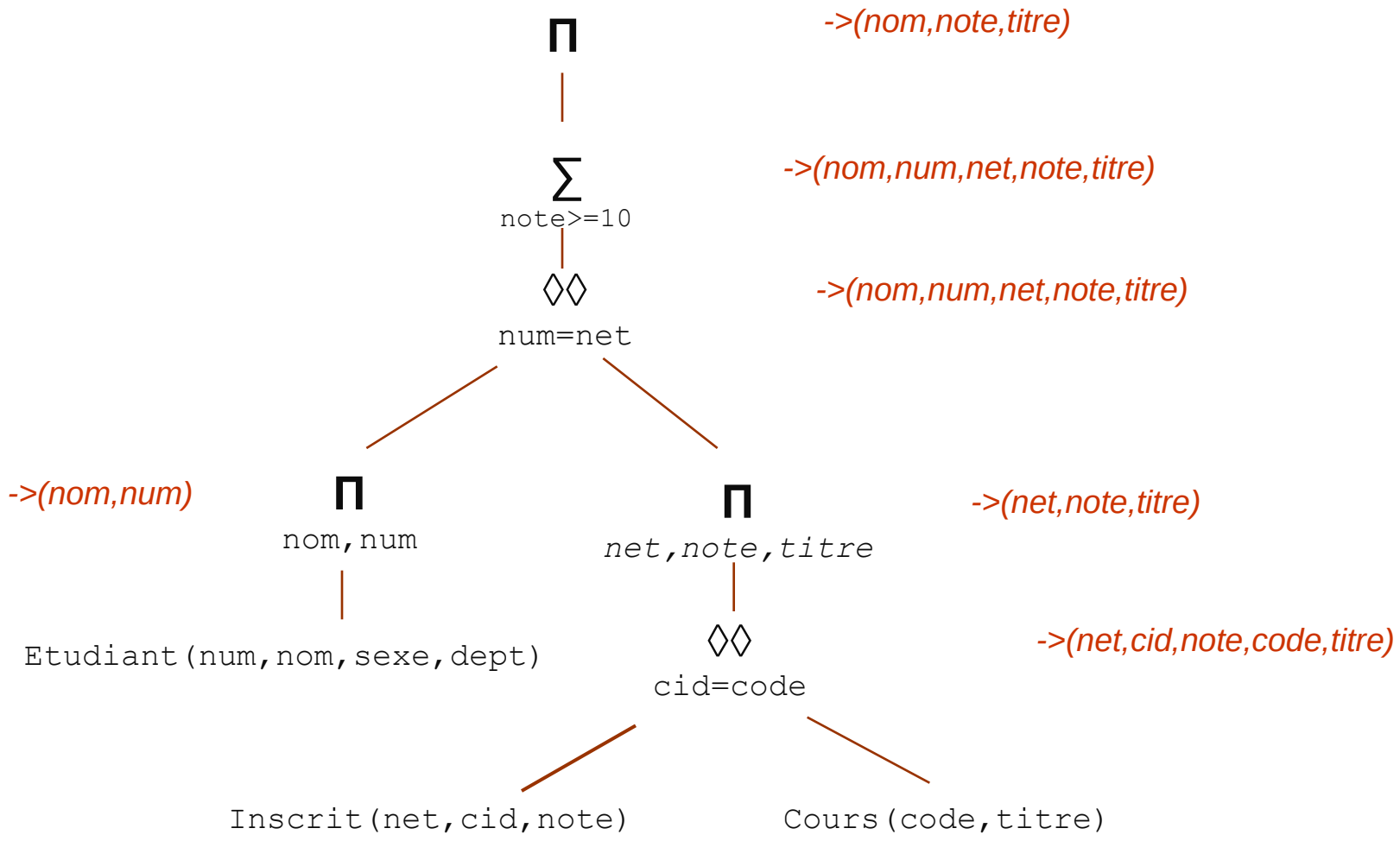
  Etudiant (num, nom, sexe, dept)
  Cours (code, titre)
  Inscrit (net, cid, note)

On veut : étudiant et cours lorsque la note est >= 10.
donne l'expression relationnelle suivante:

$$\Pi\ (\sum_{Note>=10}\ (\Pi(Etudiant)\ \lozenge\lozenge_{num=net}\ (\Pi(Inscrit\ \lozenge\lozenge_{Code=cid}\ Cours))))$$

Algèbre relationnelle
Expressions relationnelles et requêtes (suite)

Π (∑ (Π(Etudiant) ◊◊ (Π(Inscrit ◊◊ Cours))))



Requêtes SQL
Historique

- Le modèle relationnel a été inventé par E. F. Codd en 1970, suite à quoi de nombreux langages ont fait leur apparition pour traduire en code informatique le modèle relationnel:
  - IBM Sequel (Structured English Query Language) en 1977
  - IBM Sequel/2
  - IBM System/R
  - IBM DB2

- Ce sont ces langages qui ont donné naissance au standard SQL (**S**tructured **Q**uery **L**anguage), normalisé en 1986 par l'ANSI pour donner SQL/86.

- SQL est un langage fourni avec tout SGBD relationnel commercialisé.

Requêtes SQL
Historique des Normes

Année	Nom	Appellation	Commentaires
1986	ISO/CEI 9075:1986	SQL-86 ou SQL-87	Édité par l'ANSI puis adopté par l'ISO en 1987.
1989	ISO/CEI 9075:1989	SQL-89 ou SQL-1	Révision mineure.
1992	ISO/CEI 9075:199	SQL-92 ou SQL2	Révision mineure.
1999	ISO/CEI 9075:1999	SQL-99 ou SQL3	Expression Rationnelles, Requêtes récursives Déclencheurs, Quelques fonctions OO.
2003	ISO/CEI 9075:2003	SQL-2003	Fonctions Manipulation XML
2008	ISO/CEI 9075:2008	SQL-2008	Fonctions de fenêtrage(first value, last value) Limitation nb ligne Mécanisme d'auto incréments.

Requêtes SQL
Rôle

- SQL est un langage pour :
  - définir les relations (tables), leurs attributs (colonnes) et leurs liens éventuels (DDL)
  - insérer, modifier et supprimer des n-uplets (lignes) (DML)
  - écrire des expressions relationnelles (requêtes) pour extraire des données (DQL)
  - contrôler les opérations et organiser des transactions (DCL)
  - créer des utilisateurs et paramétrer leurs privilèges

- L'instruction de base permettant d'écrire des requêtes est :

```
SELECT ...
FROM ...
[ WHERE ... ]
[ ORDER BY ... ]
[ GROUP BY ... [ HAVING ... ] ]
```

- SQL n'élimine pas les doublons à moins que cela ne soit explicitement précisé par le mot clé DISTINCT.

Requêtes SQL

Syntaxe

- `SELECT` est l'unique instruction permettant d'écrire des requêtes. La syntaxe (simplifiée) est :

```
SELECT expression [ , expression ]*
    FROM table [ , table ]*
    | [ WHERE condition ]
```

- Notation :
  - tous les mots en majuscule désignent des mots-clés SQL
  - [`element`] désigne un élément optionnel
  - [`element`]* désigne un élément répété 0 ou N fois
  - `element` désigne un élément décrit plus loin

- Le résultat de `SELECT` est une pseudo-table ; selon l'outil utilisée, elle sera affichée, accessible ligne par ligne, etc.

Requêtes SQL
Syntaxe (suite)

- Tous les noms de tables et de colonnes sont fixés

- Si on a les tables suivantes :
  - `departement (numdept,nomdept)`
  - `personne (nom,prenom,age,dept)`
- Une projection simple s'écrit :

```
SELECT nom, prenom
    FROM personne
```

- Une sélection simple (et une projection) s'écrit :

```
SELECT nom, prenom              <= Projection
    FROM personne
    WHERE age < 60              <= séléction
```

- Une jointure simple s'écrit :

```
SELECT nom, prenom, nomdept
    FROM personne, departement
    WHERE dept = numdept    <= jointure
```

Requêtes SQL
Syntaxe (suite)

- La liste des tables donne l'ensemble des noms de colonnes utilisables dans les conditions et les expressions

- Le nom de colonne peut être préfixé par le nom de la table :

```
SELECT personne.nom, personne.prenom,
   departement.nomdept
FROM personne, departement
WHERE departement.numdept = personne.dept
```

(permet de lever les ambiguïtés éventuelles)

- L'expression spéciale * sélectionne toutes les colonnes

```
SELECT * FROM personne WHERE age < 60
```

Requêtes SQL
Syntaxe (suite)

- En cas d'*auto-jointure*, les tables peuvent être « *aliasées* »

```
SELECT p1.prenom, p2.prenom
   FROM personne p1, personne p2
   WHERE p1.age = p2.age          => cond. auto-jointure
```

(utile aussi pour simplifier les conditions)

- La requête ci-dessus renvoie tous les couples de personnes ayant le même age : (`Jean,Paul`) `et` `(Paul,Jean)` par exemple, et aussi (`Jean,Jean`)

```
SELECT p1.prenom, p2.prenom
    FROM personne p1, personne p2
    WHERE p1.age = p2.age
    AND p1.prenom < p2.prenom   =>élimine prénoms identiques
```

Requêtes SQL
Conditions

- Les conditions (clause `WHERE`) sont :
  - soit des conditions simples liant des expressions par un opérateur (`=, <>, <, <=, >, >=, etc`.). Les expressions sont formées à partir de noms de colonnes et/ou de constantes ;
  - soit des combinaisons logiques de conditions, utilisant les connecteurs `OR, AND et NOT`, éventuellement parenthésées.

- Les constantes s'écrivent :
  - `42,-17,12.34,etc.` pour les nombres
  - `'Hello', 'l''apostrophe' etc`. pour les chaînes

- Il y a quelques opérateurs originaux :
  - `expr BETWEEN expr AND expr`
  - `expr IS [ NOT ] NULL`

Requêtes SQL
Conditions (suite)

- `expr [ NOT ] LIKE motif [ ESCAPE car ]` où motif est une chaîne de caractères contenant les caractères spéciaux :
  - `%` qui désigne un nombre quelconque de caractères
  - `_` désigne un unique caractère et *`car`* désigne un caractère d'échappement

  Exemple :
  - `nomdep LIKE '%marketing%'`
  - `note LIKE '__\%' ESCAPE '\'`

    `% n'est plus un Joker`

- Attention aux chaînes de caractères en cas d'apostrophe !

Requêtes SQL
Conditions (suite)

- Il est également possible de manipuler des listes de valeurs, avec `[ NOT ] IN :`
  - `prenom IN ('Jean','Paul','Jacques')`
  - `age IN (40,50,60)`

- Les opérateurs de comparaison `(=, <>, <, > etc.)` peuvent être associés à `ANY ou ALL` suivi d'une liste
  - `op ANY liste` : compare vraie pour une des valeurs
  - `op ALL liste` : compare vraie pour toutes les valeurs

  Exemples :
  - `prenom = ANY ('Jean','Paul') (idem IN)`
  - `prenom = ALL ('Jean','Paul')` n'a pas de sens
  - `age < ANY (...)` (âge inférieur au maximum)
  - `age > ANY (...)` (âge supérieur au minimum)
  - `age < ALL (...)` (âge inférieur au minimum)
  - `age > ALL (...)` (âge supérieur au maximum)

Requêtes SQL
Conditions (suite)

- Avec `IN, ANY et ALL`, la liste peut être remplacée par une requête, placée entre parenthèses

```
SELECT nom, prenom
   FROM personne
   WHERE cp
    IN (SELECT codpost
           FROM ville
           WHERE codpost IN (67,68)
       )
```

- A utiliser avec modération : deux requêtes imbriquées

Requêtes SQL
Expressions et fonctions

- Les expressions placées après `SELECT` peuvent combiner plusieurs colonnes et/ou utiliser des fonctions SQL :

```
SELECT INITCAP(prenom)||' '||UPPER(nom)
    FROM personne
```

- Fonctions pour les chaînes de caractères :
  - `||` est l'opérateur de concaténation (oracle)
  - `LENGTH(s), LOWER(s), UPPER(s), INITCAP(s), SUBSTR(s,deb,long), INSTR(S,s)...`

- Fonctions pour les nombres :
  - `+, -, *, /, ...` <= operateurs math.
  - `ABS, MOD, ROUND, TRUNC, SIGN,CEIL, FLOOR` <=fct numérique
  - `COS, SIN, TAN, COSH, SINH, TANH` <= fct trigonométrique
  - `EXP, LOG, LN, POWER, SQRT...`

Requêtes SQL
Expressions et fonctions (suite)

- Fonction pour les dates :
  - `date + nombre` (ajout de jours à une date, idem pour -)
  - `date - date` (différence de deux dates, en jours)
  - `SYSDATE` (date courante)
  - `MONTHS_BETWEEN(d1,d2), ADD_MONTHS(d1,n)...`
- Des fonctions de conversion :
  - `TO_CHAR` (pour date et nombre)
  - `TO_DATE` (pour chaîne)
  - `TO_NUMBER` (pour chaîne)
- Une fonction de transformation de NULL
  - `NVL(e,v) vaut v si e est NULL, e sinon`

Requêtes SQL
Jointures

- Si `SELECT` liste plusieurs tables après `FROM`, les conditions portent sur le produit cartésien des tables

```
SELECT nom, nomdep
   FROM personne, departement
   WHERE dept = numdept
```

Ici, c'est la condition qui « fait » la jointure

- Alternativement, on peut écrire :

```
SELECT nom, nomdept
   FROM personne JOIN departement
   ON dept = numdept
```

(peut aider le SGBD à optimiser le plan de requête)

Requêtes SQL
Jointures (suite)

- De la même façon les jointures externes ont leur propre syntaxe

```
Personne(nom, dept)
Departement(numdept, nomdept)

SELECT nom, nomdept
   FROM personne
        LEFT OUTER JOIN
        departement
        ON dept = numdept
```

(et `RIGHT OUTER JOIN` respectivement)

Requêtes SQL
Jointures (suite)

- On retrouve dans une requête les opérations de base algèbre relationnelle :

```
1. SELECT nomdept
2.         FROM personne, departement
3.         WHERE dept = numdept
4.                 AND age > 50
```

  - Projection : ligne 1
  - Sélection : ligne 4
  - Jointure : lignes 2 et 3

- Il est quelquefois difficile de comprendre une requête :

```
SELECT nom, note, titre
   FROM etudiant, cours, inscrit
   WHERE num = net
    AND cid = code
    AND note >= 10
```

Requêtes SQL
Opérations ensemblistes

- Les opérations ensemblistes sont :
  - `requete UNION [ ALL ] requete` `<= R1 u R2`
  - `requete INTERSECT requete` `<= R1 n R2`
  - `requete MINUS requete` `<= R1 - R2`

  L'opération UNION ALL produira éventuellement des doublons

- Rarement utilisées dans la pratique, parce que, si :

  - `r1` est : `SELECT e FROM t WHERE cond1`
  - `r2` est : `SELECT e FROM t WHERE cond2`

  Alors :

  - `r1 UNION r2` est la requête :
    `SELECT e FROM t WHERE (cond1) OR (cond2)`
  - etc.

  (elles restent utiles si les requêtes font des jointures distinctes, ou si UNION ALL est nécessaire)

Requêtes SQL
Tri

- La clause ORDER BY permet de trier les lignes du résultat :

```
SELECT nom, prenom
   FROM personne
   ORDER BY age
```

- On peut préciser le sens de tri :

```
ORDER BY age DESC
```

- On peut préciser plusieurs critères :

```
ORDER BY age DESC, nom ASC, prenom ASC
```

- Un tri, c'est toujours coûteux...
- Rappel : il n'y a aucun ordre garanti sans ORDER BY !
- Il est également possible de trier une colonne de type date sans plus de difficulté.

Requêtes SQL
Groupements

- SQL prévoit également le groupement de lignes, et la possibilité de faire des calculs par groupe.

Dans cette partie, on utilise :

```
personne (nom,prenom,age,salaire,dept)
departement(numdept,nomdept)
```

- Exemple : calculer la moyenne d'âge par département

```
SELECT AVG(age), dept
FROM personne
GROUP BY dept
```

- Déroulement du calcul :
  - ✓ 1) calcul du résultat de la requête (ici, la table personne)
  - ✓ 2) formation des groupes selon GROUP BY
  - ✓ 3) calcul d'un (unique) résultat pour chaque groupe

Le résultat contient autant de lignes qu'il y a de groupes

Requêtes SQL
Groupements (suite)

- La liste d'expressions suivant `SELECT` peut contenir :
  - des colonnes apparaissant aussi dans `GROUP BY`
  - des « fonctions d'agrégation » appliquées aux autres colonnes

  Les fonctions d'agrégation sont : `COUNT, MIN, MAX, SUM, AVG, VARIANCE et STDDEV`

- `COUNT` a des variantes :
  - `COUNT(*)` : nombre de lignes
  - `COUNT(expr)` : nombre de lignes pour lesquelles `expr` n'est pas `NULL`
  - `COUNT(DISTINCT expr)` : nombre de valeurs (`non NULL`) de `expr` distinctes

Requêtes SQL
Groupements (suite)

- Il est possible de ne conserver que certains groupes

```
SELECT AVG(salaire), dept
    FROM personne
    GROUP BY dept
    HAVING COUNT(*) >= 10
```

- La clause HAVING s'applique à un groupe, donc la condition :
  - ne peut pas utiliser de colonnes non groupante
  - peut utiliser des fonctions d'agrégation

- On peut utiliser `GROUP BY` et `ORDER BY` dans le même requête (même restriction de tri si le même attribut est utilise)

Requêtes SQL
Exercice

- On dispose de la base de données suivante :
  - `departement (numd,nomd)`
  - `employe (nume,nome,numd)`
  - `projet (nump,nomp,cout)`
  - `participe (nume,nump,role)`

- Détail des colonnes :

`numd` numéro du département
`nomd` nom du département
`nume` numéro de l'employé
`nome` nom de l'employé
`nump` numéro du projet
`nomp` nom du projet
`cout` coût en euros
`role` rôle de l'employé au sein du projet

Requêtes SQL
Exercice (suite)

- Donnez les requêtes SQL pour obtenir :
  1. projets dont le coût est supérieur à 1MC
  2. employés travaillant dans les services 1, 5 ou 8
  3. différents rôles tenus par les employés
  4. employés travaillant dans le service « Comptabilité »
  5. noms des employés travaillant pour le projet 42
  6. noms des employés travaillant pour le projet « Barrage »
  7. noms des départements dont des employés travaillent pour le projet « Barrage »
  8. noms des employés du service « IT » travaillant pour le projet « Barrage »
  9. nombre d'employé ayant le rôle « chef » par service
  10. coût total des projets auxquels participent les employés du service 17

Troisième Partie

Schémas