LIF4 - Initiation aux Bases de données : SQL - 2

E.Coquery

emmanuel.coquery@liris.cnrs.fr

http ://liris.cnrs.fr/~ecoquery

Opérations ensemblistes

- Permettent de combiner les résultats de plusieurs SELECT.
- Opérateur :
  - $\cup$: UNION
  - $\cap$: INTERSECTION
  - $-$: MINUS
- Pas de doubles (DISTINCT implicite).
- Les SELECT doivent contenir le même nombre d'attributs.
- Les noms des attributs sont ceux du premier SELECT.
  - C'est l'ordre des attributs qui compte.
- Seul le dernier SELECT peut contenir un ORDER BY.
  - Les colones à utiliser pour le tri sont précisées par leur numéro et pas par leur attribut.

Exemple

Schéma :
Employe1(Nom, Num, Fonction, NumSup, Embauche, Salaire, NumDept)
Employe2(Nom, Num, Fonction, Numsup, Embauche, Salaire, NumDept)

Liste des département ayant des employé dans 2 filiales dont les employés sont donnés par Employe1 et Employe2 :

```
(SELECT NumDept FROM Employe1)
INTERSECT
(SELECT NumDept FROM Employe2) ;
```


Expressions

Il est possible d'utiliser des expressions plus complexes que simples attributs.

Entre autres :

- Fonctions et expressions arithmétiques
- Fonctions sur les chaînes de caractères
- Fonctions sur les dates
- Fonctions de conversion

Il existe également des fonctions de groupes permettant de traiter plusieurs lignes à la fois.

Expressions - 2

Ces expressions sont utilisables :

- Dans le SELECT :
  - le nom dans la relation résultat est en général l'expression elle-même
    $\Rightarrow$ utiliser le renommage.
- Dans le WHERE :
  - permet d'exprimer des conditions plus complexes
- Dans le ORDER BY :
  - il est ainsi possible de trier les lignes selon des valeur plus complexes que de simples attributs

Quelques fonctions numériques

- $+$: unaire et binaire ;
- $-$: unaire et binaire ;
- $*$: multiplication et $/$: division ;
- $ABS(e)$: valeur absolue de e ;
- $COS(e)$: cosinus de e avec e en radians ;
- $SQRT(e)$: racine carrée de e ;
- $MOD(m, n)$: reste de la division entière de m par n, vaut 0 si $n = 0$;
- $ROUND(e, n)$: valeur arrondie de e à n chiffres après la virgule, n optionnel et vaut 0 par défaut ;
- $TRUNC(e, n)$: valeur tronquée de e à n chiffres après la virgule, n optionnel et vaut 0 par défaut.

Pour ROUND et TRUNC, si n est négatif cela indique des chiffres avant la virgule.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche,
Salaire, Commission, Num_Dept)

Donner pour chaque commercial son revenu (salaire + commission) :

```
SELECT Nom, (Salaire + Commission) Revenu
FROM Employe
WHERE Fonction = 'commercial' ;
```


Exemple - 2

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche,
Salaire, Commission, Num_Dept)

Donner la liste des commerciaux classée par rapport commission/salaire décroissant.

```
SELECT Nom, (Commission/Salaire) Rapport
FROM Employe
WHERE Fonction = 'commercial'
ORDER BY Commission/Salaire ;
```


Exemple - 3

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche,
Salaire, Commission, Num_Dept)

Donner, avec leur salaire journalier arrondi au centime près, la liste des employés dont la commission est inférieure à 50% du salaire.

```
SELECT Nom, ROUND(Salaire/(22*12), 2) SJournalier
FROM Employe
WHERE Commission <= Salaire * 0.5 ;
```


Fonctions sur les chaînes de caractères

- *CONCAT*(e_1, e_2) : concaténation de e_1 et e_2
  - Dans certains systèmes, *CONCAT* peut prendre plus de deux arguments.
  - Dans certains systèmes, *CONCAT* est représenté par l'opérateur binaire ||.
- *REPLACE*(e, old, new) : Renvoie e dans laquelle les occurrences de *old* on été remplacées par *new*.
- *UPPER*(e) : convertit e en majuscules.
- *LENGTH*(e) : longueur de e.
- *INSTR*(e, s) : donne la position de la première occurrence s dans e.
- *SUBSTR*(e, n, l) ou *SUBSTRING*(e, n, l) : renvoie la sous-chaîne de e commençant au caractère n et de longueur l
  - si l n'est pas précisé, on prend la sous-chaîne du caractère n jusqu'à la fin de e.

Fonctions sur les dates

Oracle :

- $d + n$ ou $d - n$: d est une date, le résultat est $d \pm n$ jours.
- *ADD_MONTHS*(d, n) : ajoute n mois à d.
- $d_1 - d_2$: nombre de jours entre d_1 et d_2.
- *SYSDATE* : date courante.

MySQL :

- *ADDDATE*$(d, INTERVAL\ n\ DAY)$: ajoute n jours à d.
  - *DAY* peut être remplacé par *SECOND*, *MINUTE*, *HOUR*, *MONTH*, ou *YEAR*.
- *SUBDATE*$(d, INTERVAL\ n\ DAY)$: similaire à *ADDDATE*, mais effectue une soustraction.
- *DATEDIFF*(d_1, d_2) : nombre de jour entre d_1 et d_2.
- *SYSDATE*() : date courante.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche,
Salaire, Commission, Num_Dept)

Donner nombre de jours depuis l'embauche de chaque employé.

```
SELECT Nom, DATEDIFF(SYSDATE(),Embauche)
FROM Employe ;
```

Fonctions de conversion

- *ASCII*(*e*) : renvoie le code ASCII du premier caractère de *e*.

Oracle :

- *CHR*(*e*) : renvoie le caractère dont le code ASCII est *e*.
- *TO_NUMBER*(*e*) convertit la chaîne *e* en nombre.
- *TO_CHAR*(*e*, *format*) convertit *e* en chaîne de caractères.
  - *e* peut être un nombre ou une date ;
  - *format* indique la forme que doit avoir le résultat.
- *TO_DATE*(*e*, *format*) convertit une chaîne de caractères en date.
  - *format* est un chaîne de caractères contenant une indication sur la représentation de la date.
  - ex : TO_DATE('12122003','ddmmyyyy') donne la date '2003-12-12'

Fonction de conversion - 2

MySQL :

- *CAST*(*e AS type*) ou *CONVERT*(*e*, *type*) : convertit *e* en *type*.
  - *type* peut être BINARY, CHAR, DATE, TIME, DATETIME, SIGNED, UNSIGNED

Exécution naïve

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Récupération des données dans le FROM
  → on obtient un produit cartésien $table_1 \times table_2 \times \ldots$
- Filtrage des n-uplets obtenus en utilisant la condition du WHERE
- Tri des n-uplets restant suivant l'ordre spécifié par ORDER BY
- Calcul des n-uplets indiqué dans le SELECT à partir des restant n-uplets triés.

Exécution naïve - 2

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Les requêtes imbriquées dans le FROM sont exécutées juste avant la création du produit cartésien.
- Les requêtes imbriquées dans le WHERE sont exécutées pour chaque n-uplet à tester.

En réalité, le SGBD optimise l'exécution des requêtes.

- Par exemple, les sous-requêtes dans le WHERE qui ne dépendent pas de la requête principale ne seront exécutées qu'une seule fois.

Exemple

Schéma :
Departement(Num_dept, Nom_dept, Num_bat)
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

```
SELECT Nom_dept, Batiment.Nom_bat
FROM Departement, Batiment
WHERE Departement.Num_bat = Batiment.Num_bat
ORDER BY Nom_dept ;
```


Exemple - 2

Departement			Batiment			
Num_dept	Nom_dept	Num_bat	Num_bat	Nom_bat	Ent_princ	Ent_Sec
10	Marketing	1	1	Turing	Nord	Ouest
20	Developpement	2	1	Turing	Nord	Ouest
30	Direction	3	1	Turing	Nord	Ouest
10	Marketing	1	2	Einstein	Ouest	NULL
20	Developpement	2	2	Einstein	Ouest	NULL
30	Direction	3	2	Einstein	Ouest	NULL
10	Marketing	1	3	Newton	Sud	Nord
20	Developpement	2	3	Newton	Sud	Nord
30	Direction	3	3	Newton	Sud	Nord
10	Marketing	1	4	Pointcarre	Est	NULL
20	Developpement	2	4	Pointcarre	Est	NULL
30	Direction	3	4	Pointcarre	Est	NULL

```
FROM Departement, Batiment
```

Exemple - 2

Departement			Batiment			
Num_dept	Nom_dept	Num_bat	Num_bat	Nom_bat	Ent_princ	Ent_Sec
10	Marketing	1	1	Turing	Nord	Ouest
20	Developpement	2	1	Turing	Nord	Ouest
30	Direction	3	1	Turing	Nord	Ouest
10	Marketing	1	2	Einstein	Ouest	NULL
20	Developpement	2	2	Einstein	Ouest	NULL
30	Direction	3	2	Einstein	Ouest	NULL
10	Marketing	1	3	Newton	Sud	Nord
20	Developpement	2	3	Newton	Sud	Nord
30	Direction	3	3	Newton	Sud	Nord
10	Marketing	1	4	Pointcarre	Est	NULL
20	Developpement	2	4	Pointcarre	Est	NULL
30	Direction	3	4	Pointcarre	Est	NULL

```
WHERE Departement.Num_bat = Batiment.Num_bat
```


Exemple - 3

Departement			Batiment			
Num_dept	Nom_dept	Num_bat	Num_bat	Nom_bat	Ent_princ	Ent_Sec
20	Developpement	2	2	Einstein	Ouest	NULL
30	Direction	3	3	Newton	Sud	Nord
10	Marketing	1	1	Turing	Nord	Ouest

```
ORDER BY Nom_dept
```

Nom_dept	Num_bat
Developpement	Einstein
Direction	Newton
Marketing	Turing

```
SELECT Nom_dept, Batiment.Nom_bat
```


Regroupements

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
GROUP BY att_k, att_l, ...
ORDER BY att_i, att_j, ...

Le GROUP BY, exécuté après le WHERE, indique de procéder à une répartition du résultat en groupes de n-uplets :

- Deux n-uplets sont dans un groupe s'il ont mêmes valeurs sur les attributs att_k, att_l, ...
- Si deux n-uplets sont dans deux groupes, alors il y a au moins un attribut parmi att_k, att_l, ... pour lequel ils ont une valeur différente.

Conséquences du regroupement

- La requête ne renvoie qu'un seul n-uplet par groupe.
- Le SELECT et le ORDER BY ne peuvent utiliser que des attributs présents dans le GROUP BY.
  - Dans un groupe, la valeur pour les attributs du GROUP BY est fixe, on peut donc l'utiliser.
  - En revanche, la valeur pour les autres attributs peut varier, ce qui rend leur utilisation directe impossible.
    (On ne saurait pas quelle valeur utiliser.)

Exemple

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

```
SELECT Fonction,Num_Dept
FROM Employe
GROUP BY Fonction, Num_Dept
ORDER BY Num_Dept ;
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Dupuis	14028	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 2

```
SELECT Fonction
FROM Employe
GROUP BY Fonction, Num_Dept
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Lambert	25012	directeur	30000	20
Soule	28963	directeur	25000	10

Exemple - 3

```
ORDER BY Num_Dept
```

Nom	Num	Fonction	Salaire	Num_dept
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
LambertJr	15630	stagiaire	6000	20
Lambert	25012	directeur	30000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30

Exemple - 4

```
SELECT Fonction, Num_Dept
```

Fonction	Num_dept
commercial	10
directeur	10
ingenieur	20
stagiaire	20
directeur	20
directeur	30
gestionnaire	30

Fonctions d'agrégation

- Fonctions agissant sur un ensemble de valeurs atomiques.
- Utilisables en conjonction avec un GROUP BY pour combiner les valeurs des attributs qui ne font pas partie du GROUP BY.
- Utilisées dans le SELECT et dans le ORDER BY.
- On ne peut *pas* les utiliser dans le WHERE.
  (Le where a lieu *avant* regroupement.)
- Par exemple, $AVG(e)$ donne la moyenne de l'expression e pour le groupe considéré.

Exemple

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner le salaire moyen pour chaque fonction :

```
SELECT Fonction, AVG(Salaire) SalaireMoyen
FROM Employe
GROUP BY Fonction ;
```


Fonctions d'agrégation - 2

- $COUNT(e)$: Le nombre d'occurrences de e dans le groupe.
  - Les n-uplets pour lesquels e vaut NULL ne sont pas comptés.
  - * peut remplacer e. Compte alors le nombre de n-uplets du groupe.
- $MAX(e)$: La valeur maximale de e pour le groupe.
- $MIN(e)$: La valeur minimale de e pour le groupe.
- $SUM(e)$: La somme des valeurs de e pour le groupe.
- $AVG(e)$: La moyenne de l'évaluation de e sur le groupe.
- $STDDEV(e)$: L'écart-type de e pour le groupe.
- $VARIANCE(e)$: La variance de e pour le groupe.

e peut être précédé du mot clé DISTINCT : dans ce cas, on élimine les doublons.

- Important pour COUNT, SUM, AVG, STDDEV et VARIANCE.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Salaire, Num_Dept)
Departement(Num_dept, Nom_dept, Num_bat)

Donner pour chaque département le nombre de fonction différentes occupée dans ce département :

```
SELECT Nom_dept, COUNT(DISTINCT Fonction) NbFonctions
FROM Employe, Departement
WHERE Employe.Num_dept = Departement.Num_dept
GROUP BY Departement.Num_dept, Nom_dept ;
```

Exemple - 2

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner pour chaque département le ou les employés qui ont le plus haut salaire :

```
SELECT Num_dept, Nom, Salaire
FROM Employe
WHERE (Num_dept, Salaire) IN
   (SELECT Num_dept, MAX(Salaire)
    FROM Employe
    GROUP BY Num_dept) ;
```


Sélection des groupes

SELECT att_1, att_2, ...
FROM $table_1$, $table_2$, ...
WHERE *condition*
GROUP BY att_k, att_l, ...
HAVING *condition_groupe*
ORDER BY att_i, att_j, ...

- Le WHERE ne pour que sur les n-uplets individuels, avant regroupement.
- La condition du HAVING pour sur les groupes et pas sur les n-uplets individuels :
  - Utilisation directe des attributs du GROUP BY possible.
  - Utilisation des autres attributs à travers les fonctions d'agrégation.
  - Exécuté entre le GROUP BY et le ORDER BY.

Exemple

```
SELECT Num_Dept, COUNT(DISTINCT Fonction) NbFonctions
FROM Employe
WHERE Salaire > 15000
GROUP BY Num_Dept
HAVING COUNT(*) > 2 ;
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Dupuis	14028	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 2

```
FROM Employe WHERE Salaire > 15000
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Dupuis	14028	commercial	20000	10
LambertJr	15630	stagiaire	6000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 3

```
GROUP BY Num_Dept
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 4

```
HAVING COUNT(*) > 2
```

Nom	Num	Fonction	Salaire	Num_dept
Bellot	13021	ingenieur	25000	20
Jones	19563	ingenieur	20000	20
Brown	20663	ingenieur	20000	20
Lambert	25012	directeur	30000	20
Martin	16712	directeur	40000	30
Dupont	17574	gestionnaire	30000	30
Dupuis	14028	commercial	20000	10
Fildou	25631	commercial	20000	10
Soule	28963	directeur	25000	10

Exemple - 5

```
SELECT Num_Dept, COUNT(DISTINCT Fonction) NbFonctions
```

Num_dept	NbFonctions
10	2
20	2

Tout regrouper

Utilisation d'une fonction d'agrégation sans GROUP BY :

- Provoque la création d'un groupe englobant tous les n-uplets sélectionnés.
- Le SELECT ne peut alors contenir que des fonctions d'agrégation.
- Utile pour obtenir des informations sur l'ensemble des lignes sélectionnées.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner le total des salaires du département 10 :

```
SELECT SUM(Salaire)
FROM Employe
WHERE Num_dept = 10 ;
```


Double regroupement

Utilisation d'une fonction d'agrégation au résultat d'une fonction d'agrégation dans un SELECT :

- Possible uniquement dans une requête avec un GROUP BY.
- Cette utilisation provoque deux regroupements :
  - Premier regroupement classique par le GROUP BY
  - Deuxième regroupement implicite dû à la fonction d'agrégation dans le SELECT

Remarque : non implémenté dans MySQL, mais possibilité d'imiter ce comportement à l'aide d'une requête imbriquée.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Donner la taille du plus gros département en termes de nombre d'employés.

```
SELECT MAX(COUNT(*))
FROM Employe
GROUP BY Num_dept ;
```

```
SELECT MAX(NbEmp)
FROM (  SELECT COUNT(*) NbEmp
        FROM Employe
        GROUP BY Num_dept)
                    CountEmp ;
```