Chapitre 6

Synthèses, calculs et requêtes imbriquées

6.1 Requêtes récapitulatives

6.1.1 Fonctions

Le langage SQL dispose de fonctions spéciales qui permettent d'effectuer des opérations statistiques sur des colonnes. Ces fonctions sont au nombre de cinq[1] :

`SUM()`	effectue la somme des valeurs comprises dans la colonne (uniquement avec des valeurs numériques).
`AVG()`	calcule la moyenne des valeurs contenues dans la colonne (uniquement avec des valeurs numériques).
`MAX()`	renvoie la valeur maximale (le nombre le plus grand ou le dernier après tri s'il s'agit d'une chaîne ou d'une date)
`MIN()`	renvoie la valeur minimale (le nombre le plus petit ou le premier après tri s'il s'agit d'une chaîne ou d'une date)
`COUNT()`	renvoie le nombres d'éléments dans la colonne spécifiée. Nous verrons que cette fonction comporte quelques variantes.

Oracle offre encore les deux fonctions `VARIANCE` (variance) et `STDDEV` (écart type) dont l'utilité ne sera évidente qu'après avoir fréquenté assidûment le cours de statistiques. Ces deux fonctions sont fréquemment implémentées. On trouve aussi parfois `RANGE` qui donne la différence entre les valeurs minimale et maximale.

Nous allons utiliser une partie de la table *Scott.Emp* pour illustrer l'emploi des fonctions.

[1] Access y ajoute `FIRST`, `LAST` pour ne retenir que les n premières ou dernières lignes. Notons que ces fonctions `FIRST` et `LAST` font souvent l'objet de l'envie des utilisateurs d'autres bases de données et qu'elles ne sont pas possibles à simuler d'une manière qui soit valable pour toutes les situations. La commande `ROWNUM` d'Oracle renvoie le numéro de la ligne en cours, on peut facilement l'utiliser pour limiter les affichages. mySQL comporte également une clause `LIMIT` qui permet également de limiter l'affichage à certaines lignes.

ENAME	DEPTNO	SAL	MGR
SMITH	20	800	7902
ALLEN	30	1600	7698
WARD	30	1250	7698
JONES	20	2975	7839
MARTIN	30	1250	7698
BLAKE	30	2850	7839
CLARK	10	2450	7839
SCOTT	20	3000	7566
KING	10	5000	
TURNER	30	1500	7698
ADAMS	20	1100	7788
JAMES	30	950	7698
FORD	20	3000	7566
MILLER	10	1300	7782

FIG. 6.1 – La célèbre table des collègues de Scott

```
SELECT Sum(SAL)
FROM Emp;
```

SUM(SAL)
29025

```
SELECT AVG(Sal)
FROM Emp ;
```

AVG(SAL)
2073.21429

```
SELECT MAX(Sal)
FROM Emp ;
```

MAX(SAL)
5000

```
SELECT MIN(Sal)
FROM Emp ;
```

MIN(SAL)
800

La fonction `COUNT()` peut avoir deux types d'arguments et voir son comportement modifié par l'ajout des mots clés `ALL` et `DISTINCT`.

Lorsqu'elle contient une astérisque, elle permet de compter les lignes de la table (en prenant en compte une éventuelle restriction introduite par `WHERE`).

```
SELECT Count(*)
FROM Emp;
```

COUNT(*)
14

Si la fonction comporte un nom de champ comme argument, elle compte le nombre de lignes où ce champ n'est pas `Null`. Par défaut, le mot clé `ALL` est sous-entendu et tous les champs sont pris en compte. Si on ajoute `DISTINCT`, on ne comptera que le nombre de champs différents. Essayons les différentes variantes de `COUNT()`.

Comptage des employés qui ont un chef de service

```
SELECT COUNT(Mgr)
FROM Emp ;
```

COUNT(MGR)
13

Variante de la requête précédente

```
SELECT COUNT(ALL Mgr)
FROM Emp ;
```

COUNT(ALLMGR)
13

Comptage des managers

```
SELECT COUNT(DISTINCT Mgr)
FROM Emp ;
```

COUNT(DISTINCTMGR)
6

Ces requêtes permettent de compter les 14 employés de la table, les 13 employés qui ont un chef de service (King n'en a pas) et les 6 employés qui sont chef d'un autre.

Remarque

L'une des caractéristiques de ces fonctions, qui surprend les débutants, est leur incapacité à coexister avec des noms de champs. Le moteur de la base de données refusera la requête suivante.

```
SELECT EName, Count(ALL Mgr) FROM Emp;
```

Il faudra toujours opérer des regroupements comme dans la section suivante si on veut mélanger fonctions statistiques et valeurs. Il est, par contre, autorisé, d'utiliser plusieurs fonctions dans une même requête :

```
SELECT MIN(Sal), MAX(Sal)
FROM Emp;
```

MIN(SAL)	MAX(SAL)
800	5000

6.1.2 Regroupements

L'utilisation des fonctions va souvent de pair avec l'utilisation de l'expression `GROUP BY`. Elle effectue les opérations après regroupement selon le critère voulu.

Compter le nombre d'employés par service.

```
SELECT DeptNo, Count(Ename)
FROM Emp
GROUP BY Deptno;
```

DEPTNO	COUNT(ENAME)
30	6
20	5
10	3

L'expression `GROUP BY` autorise le mélange de noms de champs avec des fonctions mais impose de fortes restrictions sur les champs listés après le mot `SELECT`. Seuls peuvent y figurer des champs qui sont repris dans les regroupements ou des champs calculés à l'aide d'une des cinq fonctions.

Si une restriction doit s'appliquer, elle vient *avant* l'expression `GROUP BY`.

Compter le nombre d'employés gagnant plus de 2000$ dans chaque service

```
SELECT DeptNo, Count(Ename)
FROM Emp
WHERE Sal > 2000
GROUP BY Deptno;
```

DEPTNO	COUNT(ENAME)
30	6
20	5
10	3

On peut envisager plusieurs niveaux de regroupement.

Compter les employés de chaque tranche de salaire dans chaque service.

```
SELECT DeptNo, floor(Sal/1000),Count(EName)
FROM Emp
GROUP BY DeptNo, floor(Sal/1000)
Order by 1,2 DESC;
```

DEPTNO	FLOOR(SAL/1000)	COUNT(ENAME)
10	5	1
10	2	1
10	1	1
20	3	2
20	2	1
20	1	1
20	0	1
30	2	1
30	1	4
30	0	1

J'ai arbitrairement défini une tranche de salaires en utilisant l'expression `floor(Sal/1000)`. Cela donne des résultats de 0 à 5 correspondant aux tranches 0-999, 1000-1999, ... 5000-5999. `Floor()` garde la partie entière d'un nombre flottant.

La requête ainsi obtenue ne permet pas cependant pas de compter le nombre d'employés par service. La possibilité de faire ces totaux et sous-totaux n'est pas définie dans le langage SQL. Il faut généralement utiliser des générateurs d'états, intégrés dans Access ou fournis comme programme annexe dans le cas de *Crystal Report*. Oracle se distingue encore en fournissant de telles possibilités au niveau de SQL. Elles seront présentées dans le chapitre suivant.

6.1.3 Restrictions sur les affichages

Le résultat d'une requête contenant des regroupements peut encore faire l'objet d'un traitement supplémentaire. Si on ne veut garder que les services comportant plus d'un nombre donné d'employés, on ajoutera l'expression `HAVING` suivie d'une condition.

Afficher les services qui ont au moins trois employés

```
SELECT DeptNo, Count(Ename) as NbreEmployes
FROM Emp Group by DeptNo
HAVING Count(Ename) >= 3;
```

DEPTNO	NBREEMPLOYES
30	6
20	5
10	3

Il convient de bien distinguer la portée des restrictions apportées par `WHERE` et par `HAVING`.

WHERE introduit un critère qui permet de sélectionner quelles lignes de la table vont être prises en compte (par exemple, celles qui concernent un employé ayant un salaire d'au moins 2000$). La condition introduite par `WHERE` portera toujours sur les valeurs des champs des tables intervenants dans la relation initiale.

Compter les employés gagnant au moins 2000$ dans chaque service

```
SELECT DeptNo, Count(Sal)
FROM Emp
WHERE Sal >= 2000
GROUP BY DeptNo;
```

DEPTNO	COUNT(SAL)
30	1
20	3
10	2

HAVING précise quelles lignes de la table finale seront retenues (par exemple, celles où le nombre d'employés est au moins de 3). La condition introduite par `HAVING` ne porte jamais sur les données originales des tables, mais sur les résultats des fonctions statistiques.

Afficher les services ayant au moins trois employés gagnant au moins 2000$

```
SELECT DeptNo, Count(Sal)
FROM Emp
WHERE Sal >= 2000
HAVING Count(Sal)>= 3
GROUP BY DeptNo;
```

DEPTNO	COUNT(SAL)
20	3

Remarque

Les regroupements nous autorisent à faire des imbrications de fonctions. Par exemple, la requête qui suit nous indique combien d'employés compte le service le plus peuplé de l'entreprise :

```
SELECT Max(Count(DeptNo))
FROM Emp
Group by DeptNo;
```

MAX(COUNT(DEPTNO))
6

Cela ne nous dit malheureusement pas lequel c'est.Toute tentative d'introduire le numéro du service (*DeptNo*) est rejetée. La section suivante nous permettra de résoudre le problème.

```
SELECT [DISTINCT]|[ALL] Champs ou Fonctions
FROM NomTable ou Jointure
WHERE Condition de sélection
GROUP BY Champs de groupement
HAVING Condition sur des fonctions
ORDER BY Champ | Numéro [ASC]|[DESC] [, ...]
```

6.2 Requêtes imbriquées

La norme SQL 2 permet d'imbriquer les requêtes. Sur les systèmes les plus permissifs, on peut introduire une requête secondaire (entourée de parenthèses) à la place d'une valeur quelconque dans une autre requête, à la place d'une liste de valeurs, voire même à la place d'un nom de table. Il faut évidemment que les données renvoyées par cette requête puissent s'intégrer dans le contexte donné. Nous examinerons successivement l'intégration d'une requête produisant les différents types de résultats :

- une seule cellule
- une colonne de cellules
- un ligne de cellules
- une table comportant plusieurs lignes et colonnes.

Si une requête ne produit pas de résultat, on obtient la valeur *Null* qui peut être diversement appréciée selon le contexte où figure la requête imbriquée. Il faudra toujours penser à cette possibilité.

On pourra également distinguer les requêtes imbriquées qui sont indépendantes de la requête principale (elles peuvent ou non utiliser la même table) de celles qui utilisent les informations de la requête principale. On parle alors de requêtes *synchronisées*.

6.2.1 Requêtes imbriquées renvoyant une seule cellule

On utilisera les requêtes imbriquées renvoyant une seule cellule pour remplacer une valeur unique, soit dans un test, soit même dans une insertion. On ne mettra donc qu'une seule colonne dans la clause `SELECT`. Ce sont les plus délicates à manipuler puisque si la requête renvoie plusieurs lignes, on déclenche une erreur.

La requête suivante nous renvoie le numéro du producteur *Georges* :

```
SELECT idProducteur FROM Producteurs
WHERE PrenomProd='Georges';
```

IDPRODUCTEUR
1

Nous pouvons l'utiliser pour obtenir la liste de ses émissions ou lui attribuer une nouvelle émission :

```
SELECT TitreSerie FROM Series
WHERE idProducteur=(SELECT idProducteur FROM Producteurs
                    WHERE PrenomProd='Georges');

INSERT INTO Series(idSerie,TitreSerie,idProducteur)
VALUES(5,'Qui veut gagner des bidons',
(SELECT idProducteur FROM Producteurs WHERE PrenomProd='Georges'));
```

La requête imbriquée nous dispense de connaître le numéro de producteur de Georges, ce qui est bien utile pour l'insertion de la nouvelle série d'émissions. On peut remarquer que les données de la requêtes imbriquée ne figureront pas dans la résultat de la première requête. Cela rappelle les semi-jointures[2].

Les requêtes imbriquées renvoyant une seule valeur sont souvent employées avec des fonctions statistiques.

Compter combien d'étudiants ont des points supérieurs à la moyenne

```
SELECT Count(*) FROM Delibe
WHERE Total>(SELECT Avg(Total) FROM Delibe);
```

6.2.2 Requêtes synchronisées renvoyant une seule cellule

Dans certains cas, la valeur renvoyée par la requête secondaire est dépendante des valeurs de la requête principale. Considérons la table *Resultat*[3] proposant des critères de réussite en première session :

MIN	MAX	VERDICT
0	14	Echec
15	17	2me session
18	30	Réussite

Nous voudrions faire figurer à côté de chaque note, le verdict final obtenu[4]. Il suffit tout simplement de comparer la note de l'étudiant avec les valeurs `MIN` et `MAX` de la table.

Afficher le nom, le total des points et la décision du jury

[2] On aurait pu obtenir précisément la liste des émissions de Georges à l'aide de la semi-jointure suivante :

```
SELECT TitreSerie
FROM Series INNER JOIN Producteurs USING(idProducteur)
WHERE PrenomProd='Georges';
```

[3] La table *Delibe* est en fait une vue qui ajoute le champ calculé *Total* à la table des étudiants. On peut, si nécessaire, la créer à l'aide de la commande suivante :

```
CREATE VIEW DELIBE AS
SELECT N,Nom,Prenom, Section, DateNaiss, Interro1,
Interro2, Examen,  Interro1+Interro2+Examen as Total
FROM Etudiants;
```

[4] Ici encore, on pourrait obtenir un résultat similaire à l'aide d'une jointure, dont la condition de jointure mérite d'être observée :

```
SELECT Delibe.*, Verdict
FROM DELIBE INNER JOIN Resultat
ON Total BETWEEN MIN AND MAX;
```

```
SELECT Nom,Total,
  (SELECT Verdict
   FROM Resultat WHERE Total BETWEEN Min AND MAX)
Decision
FROM Delibe;
```

NOM	TOTAL	DECISION
DECRAMER	14	Echec
AENDEKERK	20	Réussite
BEAUVAIS	11	Echec
TOMIZZI	20	Réussite
DELVAUX	17	2me session
MONET	23	Réussite
LEBLOND	20	Réussite
DUPONT	16	2me session
MIKLATZKI	20	Réussite

On constate que le champ *Total* de la requête imbriquée fait partie de la table de la requête principale. La requête secondaire produit donc une valeur originale pour chaque ligne de la table principale.

Dans l'exemple relatif à la moyenne des points, nous avons calculé la moyenne sur toute l'école. Il serait plus intéressant de comparer les points obtenus par l'étudiant avec la moyenne de *sa* classe. La référence aux valeurs de la requête principale dans la requête imbriquée se complique maintenant du fait que les deux requêtes utilisent la même table. Comme dans le cas des jointures réflexives, nous serons obligés d'employer un alias.

Afficher le nombre d'étudiants ayant plus que la moyenne de leur classe

```
SELECT Count(*) FROM Delibe D
WHERE Total> (
   SELECT Avg(Total) FROM Delibe
   WHERE D.Section=Section);
```

6.2.3 Requêtes renvoyant plusieurs lignes

Lorsqu'une requête imbriquée renvoie plusieurs lignes, ses possibilités d'emploi se réduisent. On pourra l'employer dans une clause `WHERE`, à condition d'employer des opérateurs qui s'accomodent de plusieurs valeurs. Le plus employé est `IN`.

a) IN

Nous voulons savoir l'incidence des sections surpeuplées sur les résultats. Nous mettrons la barre à 35 étudiants, ce qui concerne trois sections.

Quelles sont les sections qui comportent au moins 35 étudiants ?

```
SELECT Section FROM Delibe
Group by Section having count(nom)>=35;
```

SECTION
Italien 1
Secrétariat 2
Allemand 1

Combien y a-t-il d'échecs dans les sections d'au moins 35 étudiants ?

```
SELECT Section, Count(*) FROM Delibe
WHERE Total < 15 AND Section IN
    (SELECT Section FROM Delibe
     GROUP BY Section HAVING Count(nom)>=35)
GROUP BY Section;
```

SECTION	COUNT(*)
Italien 1	3
Secrétariat 2	4
Allemand 1	3

b) EXISTS

Cet opérateur permet simplement de vérifier que la sous-requête renvoie au moins une valeur.

Afficher les voitures qui sont possédées par un employé[5]

```
SELECT * FROM Autos A
WHERE EXISTS
(SELECT * FROM Employes WHERE A.NVoiture = NVoiture);
```

NVOITURE	MARQUE	MODELE
1	Opel	Corsa
3	Opel	Astra
4	Ford	Fiesta

c) comparateur + ANY

Lorsque la requête renvoie plusieurs valeurs, on peut vérifier que la comparaison donne un résultat vrai pour au moins une des valeurs à l'aide de ANY.

Afficher les étudiants qui ont obtenu moins de points au total que certains étudiants sur les seules interrogations

```
SELECT Nom, Total From DELIBE
WHERE Total < ANY (
   SELECT Interro1+Interro2 as Eval
   FROM Etudiants);
```

NOM	TOTAL
BEAUVAIS	11
PAJUELO GALLEGE	11
PARTHOENS	11
DE KERPEL	11
VAN GHELUWE	11

[5]La semi-jointure suivante donne le même résultat :

```
Select NVoiture, Marque, Modele
FROM Autos A
INNER JOIN Employes USING(NVoiture);
```

mais il faut reconnaître que c'est trop facile !

d) comparateur + ALL

On peut aussi exiger que la comparaison soit vraie pour toutes les valeurs. Dans un tel cas, il faut évidemment veiller à ce que les « candidats » ne soient pas comparés avec eux-mêmes.

Afficher la liste des élèves qui ont plus de points que tous leurs condisciples (premiers sans ex-aequo).

```
SELECT Section, Nom, Total
FROM Delibe E
WHERE Total >
   ALL (SELECT Total
        FROM Delibe
        WHERE  E.Section = Section
        AND E.Nom <> Nom AND Total IS NOT NULL)
ORDER BY 1,2
```

SECTION	NOM	TOTAL
Allemand 3	HEPTIA	22
Anglais 2	MUKOLONGA	23
Comptabilité 1	BURY	22
Comptabilité 2	GRIGOROVITCH	23
Espagnol	MBELO LOSE LOSE	11
Espagnol 1	FRANKINET	23
Espagnol 2	BOUQUETTE	21

Afficher la liste des meilleurs résultats de chaque section

```
SELECT Section, Nom, Total,
FROM Delibe E
WHERE Total >=
   ALL (SELECT Total
        FROM Delibe
        WHERE  E.Section = Section
        and Total IS NOT NULL)
ORDER BY 1,2
```

SECTION	NOM	TOTAL
Allemand 1	GRAVAGNO	23
Allemand 1	LEONARD	23
Allemand 2	BOELEN	22
Allemand 2	BUSUITO	22
Allemand 2	HANSEN	22
Allemand 3	HEPTIA	22
Anglais 1	ARNOULD	22
Anglais 1	AYARI	22

6.2.4 Requêtes renvoyant plusieurs colonnes

Oracle implémente la norme SQL2 qui autorise l'utilisation de plusieurs valeurs dans une comparaison d'égalité. Beaucoup d'autres systèmes n'autorisent pas cette syntaxe. Il est toujours possible de s'en tirer autrement

Nous allons examiner un exemple :

Liste des meilleurs résulats par section

```
SELECT Section,Max(Total)
FROM Delibe GROUP BY Section
ORDER BY 1;
```

SECTION	MAX(TOTAL)
Allemand 1	23
Allemand 2	22
Allemand 3	22
Anglais 1	22
Anglais 2	23
Anglais 3	22
Comptabilité 1	22
Comptabilité 2	23
Espagnol	11

Liste des étudiants obtenant le meilleur résulat de leur section

```
SELECT Section,Nom,Total
FROM Delibe
WHERE (Section,Total) IN (
        SELECT Section,Max(Total)
        FROM Delibe GROUP BY Section);
```

SECTION	NOM	TOTAL
Allemand 1	GRAVAGNO	23
Allemand 1	LEONARD	23
Allemand 2	BOELEN	22
Allemand 2	BUSUITO	22
Allemand 2	HANSEN	22
Allemand 3	HEPTIA	22
Anglais 1	ARNOULD	22
Anglais 1	AYARI	22
Anglais 1	SPIRLET	22

Nous avons vu plus haut une autre manière d'obtenir cette liste sans employer un comparateur portant sur plusieurs valeurs.

6.2.5 Requêtes renvoyant une table

Si la requête renvoie une table, on peut bien entendu combiner les techniques vues pour la gestion des requêtes imbriquées multilignes et multicolonnes. Il reste un emploi particulier de ce genre de requête qui consiste à placer la requête imbriquée à la place d'une table dans une jointure.

Afficher les étudiants avec leurs points et la moyenne de leur classe

```
SELECT Nom,Total,Moyenne
FROM Delibe INNER JOIN
(  SELECT Section,AVG(Total)Moyenne
   FROM Delibe
   GROUP BY Section) M
USING(Section);
```

SECTION	NOM	TOTAL	MOYENNE
Allemand 1	ALBESA	15	17.6111111
Allemand 1	BOLL	15	17.6111111
Allemand 1	BRISY	17	17.6111111
Allemand 1	CAPETO TORRES	19	17.6111111
Allemand 1	CHAUMONT	21	17.6111111
Allemand 1	COKAIKO	13	17.6111111
Allemand 1	DELAVA	17	17.6111111
Allemand 1	DETILLOUX	16	17.6111111
Allemand 1	EL KAROUNI	19	17.6111111
Allemand 1	ENGELEN	16	17.6111111
Allemand 1	ERBESFELD	15	17.6111111
Allemand 1	FARNIR	15	17.6111111
Allemand 1	FELD	15	17.6111111
Allemand 1	VOSSEN	20	17.6111111
Allemand 1	WALRAVENS	18	17.6111111
Allemand 2	BASTOGNE	17	17.8571429
Allemand 2	BOELEN	22	17.8571429
Allemand 2	BUSUITO	22	17.8571429
Allemand 2	CUIGNESSE	18	17.8571429