-Le modèle de la base de données:
Les tableaux emp contient les employés, le tableau dept contient les départements.

La clé primaire de la table emp est empno.
La clé primaire de la table dept est deptno.

Il y a une clé étrangère dans la table emp => deptno, pointant vers la clé primaire de la table dept (depnto).

Et il y a une autre clé étrangère, le champ mgr, qui pointe vers la clé primaire empno, c'est une relation interne de la table emp à elle-même. En effet chaque employé, sauf le président, a son manager qui est un autre employé de la table emp. Il s'agit d'une hiérarchie.

a) Les jointures

Il existe au moins cinq façons d'effectuer une jointure interne dans Oracle.

La clé étrangère du premier ensemble est jointe à la clé primaire de la seconde, (en vert les résultats sur la diapositive), les lignes correspondant à ce critère sont extraites des deux tables.

Nous voyons donc que chaque fois, nous obtenons les mêmes résultats.
Pourquoi y a t-il 5 façons de le faire?

La première façon, avec la clause where, est une syntaxe ancienne d'Oracle, mais elle fonctionne toujours et reste compatible.

La deuxième façon, avec le mot-clé «inner join » et le mot-clé «on» est la manière ANSI de le faire.

En effet, dans ANSI SQL, on n'utilise pas la clause where pour écrire des jointures. On peut utiliser la clause where pour écrire d'autres conditions, comme « where sal> 1000 » ou « where com is null »

La troisième solution est une jointure naturelle. Ici, c'est possible parce que les deux tables ont le même champ «deptno».

Une autre façon est simplement de taper le mot-clé «join»,et si vous ne précisez pas le type de jointure, par défaut il s' agira de la jointure interne.

Le dernier exemple montre comment le faire avec l'utilisation d'alias. On peut mettre un alias pour chaque table dans la clause from, puis le réutiliser dans la clause where. De cette façon, on évite de taper les noms de tables entières.
Bien sûr, ici nous avons besoin de taper tous les noms de table afin de faire une différence de deptno dans la table emp et le deptno dans la table dept. Si le champ de clé étrangère a été nommé différemment du champ de clé primaire, il n'y a pas besoin d'aliases.

Le deuxième exemple est la jointure externe.

Ainsi, nous pouvons avoir une jointure externe gauche, une jointure externe droite et la jointure externe complète.

Nous voulons donc récupérer tous les employés, même ceux qui ne sont pas affectés à un département.
Ou nous voulons récupérer tous les départements, même ceux qui sont vides.
Ou nous voulons les deux.
Dans la syntaxe du standard ANSI SQL, on utilise les mots clés , left outer join, right outer join et full outer join.

Dans la syntaxe Oracle, il faut utiliser le signe (+).
Attention. Le signe + est placé du côté opposé du mot-clé «gauche» ou «droit» de l'exemple précédent.
Donc, pour la jointure externe gauche, le + est sur le côté droit.
Pour la jointure externe droite, le + est sur le côté gauche.
Vous pouvez vous rappeler cette syntaxe par exemple si vous vous rappelez que vous avez mis le signe + sur le côté où il ya une donnée manquante. (Donc il y a plus de données de l'autre côté ...)
Vous ne pouvez pas mettre le signe + sur les deux côtés.
Pour cela, vous devez utiliser les »unions ».

Les tests d’existence

Ils s’effectuent avec les sous requêtes + les mots clés « in » et « exists »
Il est essentiel de comprendre que in et exists sont équivalents dans les exemples suivants :

Select * from emp where depnto in (select deptno from dept where dept.deptno=emp.deptno)

Select * from emp where exists (select 1 from dept where dept.depnto=emp.deptno)

A cause de l’existence de la valeur « null » dans la table emp pour le champ deptno, (il existe un salarié qui n’est pas affecté à un département) not in et not exists ne donnent pas le même résultat

Select * from emp where depnto not in (select deptno from dept where dept.deptno=emp.deptno)

Select * from emp where not exists (select 1 from dept where dept.depnto=emp.deptno)

Si, à la place du deptno=null on met une valeur inexistante dans la table dept,(ce qui est possible si n’avons pas déclaré la clé étrangère) alors not in et not exists donnent le même résultat.

Fonctions d’agrégation

Les fonctions d’agrégation dans Oracle sont celles qui effectuent une sorte d'agrégation (count, avg, sum, stddev) ou celles qui recherchent des minimums, maximums.

Si nous utilisons count (*), nous comptons le nombre de lignes (tuples) dans la table. Si nous utilisons count (field_name), nous comptons le nombre de lignes qui contiennent le champ en question qui n’est pas null.

Par exemple count (sal) donnera un résultat plus élevé que count (comm), car tous les employés n'ont pas de prime. Si nous utilisons le mot clé group by, il est possible de créer les sous-totaux, par exemple au niveau du département, au niveau du job. Chaque colonne qui est mentionnée dans l'instruction select à côté de la fonction de regroupement (max, min, count etc.) doit également être mentionnée dans la clause group by, sinon la requête ne fonctionne pas et le message d'erreur est affiché. On peut, de l'autre côté, avoir plus de colonnes dans la clause group by, que dans la clause select, mais habituellement ce n’est pas utile, puisque vous regroupez des choses sans les voir.

Exemple

select job, max (sal) de emp group by rollup(job)

pour afficher le grand total.

Le cumul avec 2 paramètres (job, deptno) affiche les sous-totaux pour le premier champ (job) avec les informations associées pour le second (deptno), mais ne regroupe pas les résultats sur le second champ. Il va bien sûr donner le grand total.

Group by cube (job, deptno) affiche les deux sous-totaux, au niveau du job, au niveau du département et du grand total.

ALL, ANY and MIN, MAX

A)Select * from emp where sal > all (select sal from emp where deptno=10)

est equivalent à

Select * from emp where sal > (select max(sal) from emp where deptno=10)

B) Select * from emp where sal > any (select sal from emp where deptno=10)

est equivalent à

Select * from emp where sal > (select min(sal) from emp where deptno=10)

C) Select * from emp where sal < all (select sal from emp where deptno=10)

est equivalent à

Select * from emp where sal < (select min(sal) from emp where deptno=10)

et

d) Select * from emp where sal < any (select sal from emp where deptno=10)

est équivalent à

Select * from emp where sal < (select max(sal) from emp where deptno=10)

Les analytics

Il s'agit d'une extension du SQL standard, contenant le mot clé over. Dans quel cas les analytics sont utiles ?

Par exemple, on souhaite parfois afficher le nom du salarié, son salaire, son département et le salaire moyen de son département.

Mais avec la requête suivante, ce n'est pas possible

```
Select ename, sal, deptno ,avg(sal)
From
Emp
Group by deptno
```

Car il faudrait faire le group by par ename et sal aussi, et le résultat affiché ne correspondra plus au salaire moyen du département.

Avec les analytics, la solution est simple :

Select ename,sal,deptno,avg(sal) over (partition by deptno) from emp
Comme le header de la fonction contenant over est long dans l'affichage de sqlplus, on met souvent l'alias, par exemple

Select ename,sal,deptno,avg(sal) over (partition by deptno) avg_sal from emp

Il possible d'exécuter toutes les fonctions d'agrégation avec les analytics :

```
Select ename,sal,deptno,
avg(sal) over (partition by deptno) avg_sal ,
max(sal) over (partition by deptno) max_sal,
min(sal) over (partition by deptno) min_sal ,
sum(sal) over (partition by deptno) sum_sal
from emp
```

Dans le contexte des analytics, il est important de comprendre la notion de la fenêtre (window).

Chaque ligne sélectionnée fait partie d'une fenêtre et contient le pointeur current row qui pointe vers elle-même.

La valeur par défaut du début de la fenêtre est « unbounded preceding » , l'intervalle ouverte plus grand ou égal (>=).

L'utilisation de l'opération du (order by) tri permet d'exploiter les possibilités des fenêtres.

Dans ce sens, il revient au même d'écrire

Select ename,sal,deptno ,sum(sal) over (order by sal)

ou

Select ename,sal,deptno ,sum(sal) over (order by sal range unbounded preceding) running_total

Dans les 2 cas, la fenêtre commence à la première ligne.

Comme chaque nouvelle ligne additionne son salaire, cela donne l'effet du cumul (on l'appelle souvent running total dans les rapports informatiques) de tous les salaires précédents jusqu'à début de la fenêtre.

Si on ajoute le mot-clé partition by, avec chaque nouveau département, une nouvelle fenêtre commence.

Select ename,sal,deptno ,sum(sal) over (partition by deptno order by sal)

ou

Select ename,sal,deptno ,sum(sal) over (partition by deptno order by sal range unbounded preceding)

```
SQL> Select ename,sal,deptno,sum(sal) over(partition by deptno order by sal) run
ning_sal from emp;

ENAME          SAL     DEPTNO RUNNING_SAL
---------- ---------- ---------- -----------
MILLER         1300         10        1300<====start of the window
CLARK          2450         10        3750
KING           5000         10        8750<=====end of the window

SMITH           800         20         800 <====start of the window
ADAMS          1100         20        1900
JONES          2975         20        4875
SCOTT          3000         20       10875
FORD           3000         20       10875<====end of the window

JAMES           950         30         950<====start of the window
MARTIN          250         30        3450
WARD           1250         30        3450
TURNER         1500         30        4950
ALLEN          1600         30        6550
BLAKE          2850         30        9400<====end  of the window

JOHN          10000                  10000<=====start and the end of the window

15 rows selected.
```

Les fonctions Lead et Lag

Parfois nous avons besoin d'afficher le nom de l'employé, son salaire, le salaire précédent et le salaire suivant, trié en ordre croissant ou décroissant.

Avec les analytics, c'est facile

- Select ename,sal,deptno,lead(sal) over (order by sal) next_sal,lag(sal) over (order by sal) prev_sal from emp;
- Select ename,sal,deptno,lead(sal,2) over (order by sal) next_sal,lag(sal,2) over (order by sal) prev_sal from emp;

Le second exemple montre que lead et lag acceptant aussi un autre paramètre, pour demander la distance entre la ligne courante et la ligne demandée. Dans notre cas, cela donne le salaire qui précède le précédent et qui suit le suivant.

Avant l'existence des Analytics dans Oracle, pour afficher les 3 salaires sur la même ligne il a fallu développer un programme en PL/SQL , par exemple

- declare
- salary number;
- prev_salary number;
- prev_prev_salary number;
- next_salary number;
- cnt number;
- cursor c1 is select empno,ename,sal from emp order by deptno;
- begin
- cnt:=0;
- for x in c1 loop
- cnt:=cnt+1;
- if cnt=1 then
- prev_salary:=x.sal;
- elsif cnt=2 then
- salary:=x.sal;
- elsif cnt=3 then
- next_salary:=x.sal;
- dbms_output.put_line('empno ename prev_sal sal next_sal');
- dbms_output.put_line(x.empno||','||x.ename||','||prev_salary||','||salary||','||next_salary);
- cnt:=0;
- end if;
- end loop;
- end;
- /

Ce programme utilise un curseur, qui est une boucle, et qui sélectionne les lignes une par une.

Après avoir passé 3 fois dans la boucle, on peut afficher la première ligne etc.

Rank, dense_rank et row_number

- Select ename,sal,deptno,rank() over (order by sal) rrank,dense_rank() over (order by sal) drank, row_number() over (order by sal) rn from emp;

Rank affiche le même numéro pour les salaires équivalents, mais saute <n> numéros après, par exemple

1

1

3

Dense_rank affiche le même numéro pour les salaires équivalents, mais ne saute pas de numéros après, par exemple

1

1

2

Row_number affiche tout simplement le numéro de la ligne

1

2

3

Encore un peu sur les analytics. Dans les exemples suivants, observez où se trouve la début de la fenêtre et où est la ligne courante.

```
SQL> Select ename,deptno,hiredate,count(*) over (order by hiredate range 100 preceding) cnt_range, ename,count(*) over (order by hiredate rows 2 preceding)
cnt_rows from emp;

ENAME          DEPTNO HIREDATE            CNT_RANGE ENAME        CNT_ROWS
---------- ---------- ------------------- ---------- ---------- ----------
SMITH              20 1980-12-17 00:00:00          1 SMITH               1
ALLEN              30 1981-02-20 00:00:00          2 ALLEN               2
WARD               30 1981-02-22 00:00:00          3 WARD                3
JONES              20 1981-04-02 00:00:00          3 JONES               3
BLAKE              30 1981-05-01 00:00:00          4 BLAKE               3
CLARK              10 1981-06-09 00:00:00          3 CLARK               3
TURNER             30 1981-09-08 00:00:00          2 TURNER              3
MARTIN             30 1981-09-28 00:00:00          2 MARTIN              3
KING               10 1981-11-17 00:00:00          3 KING                3
JAMES              30 1981-12-03 00:00:00          5 JAMES               3
FORD               20 1981-12-03 00:00:00          5 FORD                3

ENAME          DEPTNO HIREDATE            CNT_RANGE ENAME        CNT_ROWS
---------- ---------- ------------------- ---------- ---------- ----------
SCOTT              20 1982-12-09 00:00:00          1 SCOTT               3
JOHN                  1982-12-23 00:00:00          3 JOHN                3
MILLER             10 1982-12-23 00:00:00          3 MILLER              3
ADAMS              20 1983-01-12 00:00:00          4 ADAMS               3

15 rows selected.
```

```
SQL> Select ename,deptno,hiredate,count(*) over (partition by deptno order by hiredate range 100 preceding) cnt_range, ename,
count(*) over (partition by deptno order by hiredate rows 2 preceding) cnt_rows from emp;

ENAME          DEPTNO HIREDATE            CNT_RANGE ENAME        CNT_ROWS
---------- ---------- ------------------- ---------- ---------- ----------
CLARK              10 1981-06-09 00:00:00          1 CLARK               1
KING               10 1981-11-17 00:00:00          1 KING                2
MILLER             10 1982-12-23 00:00:00          1 MILLER              3
SMITH              20 1980-12-17 00:00:00          1 SMITH               1
JONES              20 1981-04-02 00:00:00          1 JONES               2
FORD               20 1981-12-03 00:00:00          1 FORD                3
SCOTT              20 1982-12-09 00:00:00          1 SCOTT               3
ADAMS              20 1983-01-12 00:00:00          2 ADAMS               3
ALLEN              30 1981-02-20 00:00:00          1 ALLEN               1
WARD               30 1981-02-22 00:00:00          2 WARD                2
BLAKE              30 1981-05-01 00:00:00          3 BLAKE               3

ENAME          DEPTNO HIREDATE            CNT_RANGE ENAME        CNT_ROWS
---------- ---------- ------------------- ---------- ---------- ----------
TURNER             30 1981-09-08 00:00:00          1 TURNER              3
MARTIN             30 1981-09-28 00:00:00          2 MARTIN              3
JAMES              30 1981-12-03 00:00:00          3 JAMES               3
JOHN                  1982-12-23 00:00:00          1 JOHN                1
```

```
Select ename,deptno,hiredate,first_value(ename) over (partition by deptno order by hiredate rows 2 preceding),
last_value(ename) over (partition by deptno order by hiredate rows 2 preceding) from emp;

ENAME          DEPTNO HIREDATE            FIRST_VALU LAST_VALUE
---------- ---------- ------------------- ---------- ----------
CLARK              10 1981-06-09 00:00:00 CLARK      CLARK
KING               10 1981-11-17 00:00:00 CLARK      KING
MILLER             10 1982-12-23 00:00:00 CLARK      MILLER
SMITH              20 1980-12-17 00:00:00 SMITH      SMITH
JONES              20 1981-04-02 00:00:00 SMITH      JONES
FORD               20 1981-12-03 00:00:00 SMITH      FORD
SCOTT              20 1982-12-09 00:00:00 JONES      SCOTT
ADAMS              20 1983-01-12 00:00:00 FORD       ADAMS
ALLEN              30 1981-02-20 00:00:00 ALLEN      ALLEN
WARD               30 1981-02-22 00:00:00 ALLEN      WARD
BLAKE              30 1981-05-01 00:00:00 ALLEN      BLAKE

ENAME          DEPTNO HIREDATE            FIRST_VALU LAST_VALUE
---------- ---------- ------------------- ---------- ----------
TURNER             30 1981-09-08 00:00:00 WARD       TURNER
MARTIN             30 1981-09-28 00:00:00 BLAKE      MARTIN
JAMES              30 1981-12-03 00:00:00 TURNER     JAMES
JOHN                  1982-12-23 00:00:00 JOHN       JOHN
```

```
SQL> r
  1  Select ename,deptno,hiredate,
  2  count(*) over (order by hiredate range between 100 preceding and 100 Following)  cnt_range,
  3  ename,
  4  count(*) over (order by hiredate rows between 2 preceding and 2 following) cnt_rows
  5* from emp

ENAME           DEPTNO HIREDATE             CNT_RANGE ENAME         CNT_ROWS
---------- ---------- ------------------- ---------- ---------- ----------
SMITH               20 1980-12-17 00:00:00          3 SMITH               3
ALLEN               30 1981-02-20 00:00:00          5 ALLEN               4
WARD                30 1981-02-22 00:00:00          5 WARD                5
JONES               20 1981-04-02 00:00:00          5 JONES               5
BLAKE               30 1981-05-01 00:00:00          5 BLAKE               5
CLARK               10 1981-06-09 00:00:00          4 CLARK               5
TURNER              30 1981-09-08 00:00:00          6 TURNER              5
MARTIN              30 1981-09-28 00:00:00          5 MARTIN              5
KING                10 1981-11-17 00:00:00          5 KING                5
JAMES               30 1981-12-03 00:00:00          5 JAMES               5
FORD                20 1981-12-03 00:00:00          5 FORD                5

ENAME           DEPTNO HIREDATE             CNT_RANGE ENAME         CNT_ROWS
---------- ---------- ------------------- ---------- ---------- ----------
SCOTT               20 1982-12-09 00:00:00          4 SCOTT               5
JOHN                   1982-12-23 00:00:00          4 JOHN                5
MILLER              10 1982-12-23 00:00:00          4 MILLER              4
ADAMS               20 1983-01-12 00:00:00          4 ADAMS               3

15 rows selected.
```

Avec la clause partition by, la nouvelle fenêtre commence avec le nouveau département.

10 ←= unbounded preceding

← n rows or n range preceding

Clark ← current row

←n rows or n range following

20

30

Range (ou rows) between <N> preceding and <N> following est pratique pour trouver les moyennes mobiles !

MongoDB

Mongo a ses collections, qui correspondent aux tables d'une SGBDR, et ses documents,

qui correspondent aux lignes.

Tous les documents n'ont pas nécessairement la même structure.

Il n'est pas indispensable de créer l'utilisateur test (le schéma test) comme avec Oracle

Tout simplement, on peut faire :

"Use test"

Test sera créé implicitement.

Pas besoin de créer explicitement une collection, car elle sera créée implicitement quand on insère un premier document.

```
db.emp.insert({empno:1,ename: "Bob", sal: 100000})
WriteResult({ "nInserted" : 1 })
```

Pour vérifier le résultat de notre insert:

```
db.emp.find()
{ "_id" : ObjectId("583981276543762b35a7a6a1"), "empno" : 1, "ename" : "Bob", "sal"
: 100000 }
```

Nous pouvons voir que MongoDB a associé une clé interne (ObjectId)

Puis, nous insérons un deuxième employé, qui a une prime:

```
db.emp.insert({empno:1,ename: "John", sal: 9000, comm:100})
WriteResult({ "nInserted" : 1 })
```

Et même un troisième, qui a un champ additionel, qui n'est pas du même type comme le champ « comm » du document précedent, nous allons l'appeler « remark »:

```
db.emp.insert({empno:1,ename: "Tom", sal: 9000, remark:"half time"})
WriteResult({ "nInserted" : 1 })
```

Encore une fois, dans une collection MongoDB, tous les documents n'ont pas nécessairement la même structure.

Maintenant, quelques requêtes essentielles.

Nous voulons trouver les salaries qui gagnent 100000 ou plus.

db.emp.find({"sal": {$gte:100000}})
{ "_id" : ObjectId("583981276543762b35a7a6a1"), "empno" : 1, "ename" : "Bob", "sal" : 100000 }

On peut facilement imaginer et tester d'autres cas, avec les clauses suivantes:

$gt greater than

$lt lower than

$gte greater on equal than

$lte lower or equal than

Et un autre exemple, dans lequel nous insérons directement le nom du département dans le document EMP.

db.emp.insert({empno:2,ename: "Eric", sal: 5000,dname: "DALLAS"})
WriteResult({ "nInserted" : 1 })
> db.emp.insert({empno:2,ename: "Michel", dname: "DALLAS", comm: 90})
WriteResult({ "nInserted" : 1 })

Ceci est possible, encore une fois, les documents n'ont pas nécessairement la même structure.

Si nous cherchons tous les salarié de DALLAS , nous avons:

db.emp.find({“dname”: “DALLAS”})
{ “_id” : ObjectId(“583ab7626543762b35a7a6a9”), “empno” : 2, “ename” : “Eric”, “sal” : 5000, “dname” : “DALLAS” }
{ “_id” : ObjectId(“583ab77a6543762b35a7a6aa”), “empno” : 2, “ename” : “Michel”, “dname” : “DALLAS”, “comm” : 90 }

Retenez:
Il n y a pas de “CREATE USER” (ou “CREATE DATABASE”) command, car USE <dbname> est suffisante pour créer une base, qui correspnd à un schéma dans Oracle.

Il n y a pas de commande “CREATE COLLECTION” (DDL CREATE TABLE dans SGBDR), parce que la commande INSERT crée une collection, si celle-ci n’existe pas, et y insére un premier document.