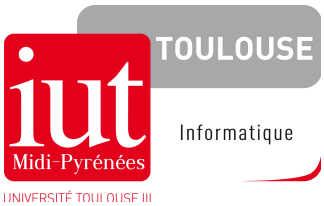
M2106 : Programmation et administration des bases de données

Cours 4/6 – Fonctions SQL internes et PL/SQL

Guillaume Cabanac

guillaume.cabanac@univ-tlse3.fr



version du 18 février 2018

Fonctions SQL internes et PL/SQL

Fonctions d'agrégation orientées colonne

Fonction d'agrégation (rappel du cours 3/5)

Une fonction d'agrégation synthétise les valeurs d'une colonne (en entrée) en une seule valeur (en sortie).

Parallèle avec les maths : $3+6=9$ $\min(3,4,5)=3$

Fonction	Description
`count(*)`	Nombre total de lignes
`count(col)`	Nombre de lignes dont la valeur de `col` n'est pas vide
`count(distinct col)`	Nombre de valeurs non nulles distinctes dans `col`
`min(col)`	Valeur minimale dans `col`
`max(col)`	Valeur maximale dans `col`
`sum(col)`	Somme des valeurs de `col`
`avg(col)`	Moyenne des valeurs de `col`
`stddev(col)`	Écart-type des valeurs de `col`
`median(col)`	Valeur médiane de `col`

Fonctions d'agrégation orientées colonnes et `null`

```
select * from participation ;

--   TATOUAGE         IDC CLASSEMENT      PRIME
-- ---------- ---------- ---------- ----------
--         666         10          2
--         666         11          5
--        1664         11          1       1234
```

⚠ `null` ≠ 0. Les fonctions d'agrégation ignorent les `null` (sauf `count(*)`).

```
select count(prime), sum(prime), avg(prime) from participation ;

-- COUNT(PRIME) SUM(PRIME) AVG(PRIME)
-- ------------ ---------- ----------
--            1       1234       1234
```

Conversion : la fonction interne `nvl(x, r)` $= \begin{cases} x & \text{si x } \texttt{is not null} \\ r & \text{si x } \texttt{is null} \end{cases}$

```
select count(nvl(prime, 0)), sum(nvl(prime, 0)), avg(nvl(prime, 0))
from participation ;

-- COUNT(NVL(PRIME,0)) SUM(NVL(PRIME,0)) AVG(NVL(PRIME,0))
-- ------------------- ------------------ ------------------
--                   3               1234       411.3333333
```

Fonctions orientées lignes

Aiguillage avec decode

$$\texttt{decode(x, e1, y1, [..., eN, yN], r)} = \begin{cases} \texttt{y1} & \text{si } \texttt{x = e1} \\ \ldots & \ldots \\ \texttt{yN} & \text{si } \texttt{x = eN} \\ \texttt{r} & \text{sinon} \end{cases}$$

```
select classement, decode(classement, 1, 'or',
                                      2, 'argent',
                                      3, 'bronze', 'nada') as resultat
from participation ;

-- CLASSEMENT RESULTAT
-- ---------- --------
--          2 argent
--          5 nada
--          1 or
```

NB : decode est une fonction spécifique Oracle.

Fonctions orientées lignes

Aiguillage avec case

```
-- Case similaire à decode
case x
  when e1 then y1
  [
    ...
    when eN then yN
  ]
  else n
end
```

```
-- Case dont les conditions sont libres
case
  when condition1 then res1
  [
    ...
    when conditionN then res N
  ]
  else n
end
```

```
select classement, case classement
                     when 1 then 'or'
                     when 2 then 'argent'
                     when 3 then 'bronze'
                     else 'nada'
                   end as resultat
from participation ;

-- CLASSEMENT RESULTAT
-- ---------- --------
--          2 argent
--          5 nada
--          1 or
```

NB : case est une fonction de la norme SQL-92 (portable).

Fonctions orientées lignes

Nombres (1/2)

Fonction	Description
`abs(x)`	Valeur absolue de x
`ceil(x)`	Valeur plafond de x : plus petit entier ⩾ x
`floor(x)`	Valeur plancher de x : plus grand entier ⩽ x
`round(x[, n])`	Arrondit x à n décimales (n = 0 par défaut)
`trunc(x[, n])`	Tronque x à n décimales (n = 0 par défaut)
`to_number(s[, f])`	Convertit en nombre la chaîne s exprimé en format f

```
select to_number('1.664') a, abs(-1.69) b, ceil(1.69) c, floor(1.69) d,
       trunc(1.69) e, trunc(1.69, 1) f, round(1.69) g, round(1.69, 1) h
from dual ;
```

```
--          A          B          C          D          E          F          G          H
-- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
--      1.664       1.69          2          1          1        1.6          2        1.7
```

La table `dual` appartient à l'utilisateur `sys`. Elle est consultable par tout utilisateur ; elle contient 1 colonne 'dummy' et une ligne 'X'.

Fonctions orientées lignes

Nombres (2/2)

Fonction	Description
`least(a, b)`	Plus petit nombre entre a et b
`greatest(a, b)`	Plus grand nombre entre a et b
`power(x, n)`	Puissance x^n
`sqrt(x)`	Racine carrée $\sqrt{x}$
`exp(x)`	Exponentielle e^x
`log(x, b)`	Logarithme $\log_b(x)$
`ln(x)`	Logarithme népérien $\log_e(x)$
`cos(x)`	Cosinus de x
`...`	...

```
select least(42, 51) a, greatest(42, 51) b, power(2, 10) c, exp(ln(42)) d
from dual ;
```

```
--          A          B          C          D
-- ---------- ---------- ---------- ----------
--         42         51       1024         42
```

Comme en mathématiques, les fonctions SQL sont composables : $e^{\ln(x)}$.

Fonctions orientées lignes

Chaînes de caractères (1/3)

Fonction	Description
`ascii(c)`	Code ASCII correspondant au caractère c
`chr(n)`	Caractère correspondant au code ASCII n
`initcap(s)`	Transforme l'initiale de chaque mot de s en lettre capitale
`length(s)`	Longueur de la chaîne s
`lower(s)`	Convertit une chaîne s en lettres minuscules
`upper(s)`	Convertit une chaîne s en lettres majuscules
`soundex(s)`	Valeur représentant s avec l'algorithme Soundex
`trim(s)`	Enlève les espaces en début et en fin de s

```
select chr(38) a, ascii(65) b, initcap('bob dylan') c, length('wtf') d,
       upper('wtf') e, soundex('lynda') f, soundex('linda') g, trim(' iut ') h
from dual ;

-- A          B  C                   D E   F    G    H
-- - ---------- --------- ---------- --- ---- ---- ---
-- &         54 Bob Dylan           3 WTF L530 L530 iut
```

NB : Soundex est un algorithme phonétique d'indexation de noms par leur prononciation en anglais (1918).

Fonctions orientées lignes

Chaînes de caractères (2/3)

Fonction	Description
`a || b`	Concatène les chaînes a et b
`concat(a, b)`	Concatène les chaînes a et b
`substr(s, d[, f])`	Extrait f caractères de s à partir de l'index d ⩾ 1
`instr(s, c)`	Retourne la position de c dans s ou 0 si c ∉ s
`replace(s, a, b)`	Remplace a par b dans la chaîne s
`translate(s, a, b)`	Convertit les caractères de a par ceux de b dans s
`...`	...

```
select 'bob'||'éponge' a, substr('bobby', 3, 2) b, instr('wtf', 't') c,
       instr('wtf', 'z') d, replace('wtf', 'f', 'b') e,
       translate('wtf', 'ftw', 'abc') f
from dual ;

-- A         B            C          D E   F
-- --------- -- ---------- ---------- --- ---
-- bobéponge bb           2          0 wtb cba
```

NB : Certains paramètres des fonctions sont volontairement omis ici.
Consulter la documentation pour connaître toutes les possibilités offertes.

Fonctions orientées lignes
Chaînes de caractères (3/3)

Opérateur `like`

La condition « `a` **`like`** `b` » est vraie si la chaîne `a` correspond au motif (*pattern*) décrit dans la chaîne `b`. La négation s'écrit « `a` **`not like`** `b` ».

Le motif est composé de caractères, dont des caractères d'échappement :

- `_` représente un seul caractère
- `%` représente zéro ou plusieurs caractères

```
select ville
from adherent
where ville like 'T%e' ; -- ville commence par 'T' et finit par 'e'
```

☠ Ne pas utiliser **`like`** si le motif ne contient aucun caractère d'échappement ! Utiliser l'opérateur d'égalité entre chaînes (=) à la place.

Fonctions orientées lignes
Dates

Fonction	Description
`sysdate`	Date-heure système
`d + n`	Ajoute $n \in \mathbb{Z}$ jours à la date `d`
`extract``(m from d)`	Extrait une composante `m` de la date `d` $m \in$ {`year, month, day, hour, minute, second`}
`months_between``(d1, d2)`	Nombre $n \in \mathbb{Z}$ de mois écoulés entre `d1` et `d2`
`to_char``(d[, f])`	Convertit en chaîne de format `f` la date `d`
`to_date``(s[, f])`	Convertit en date la chaîne `s` exprimée en format `f`
`...`	...

```
select to_char(sysdate, 'YYYY-MM-DD HH24:MI') a, extract(month from sysdate) b,
       to_date('08/03/1982', 'DD/MM/YYYY') + 100 c
from dual ;

-- A                          B C
-- ---------------- ---------- ---------
-- 2013-09-30 23:10           9 16-JUN-82
```

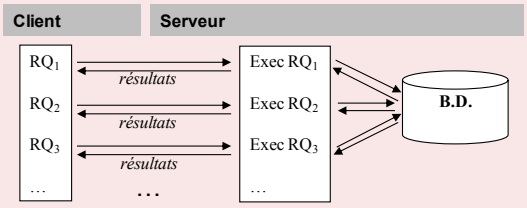
NB : Tous les formats `f` sont détaillés dans la documentation Oracle en ligne.

Apports de la programmation procédurale

Intérêt des sous-programmes stockés : fonctions et procédures PL/SQL

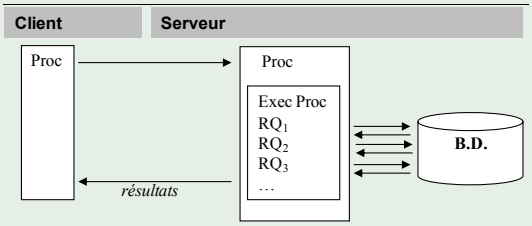
Sans sous-programme

Les ordres sont envoyés **un par un**.



Avec sous-programme

Les ordres sont envoyés **en bloc**.



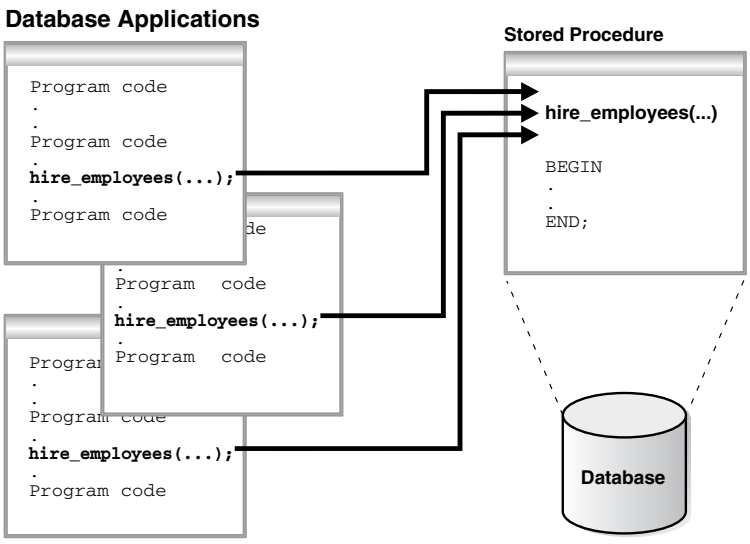
Éléments de comparaison :

- réutilisabilité du code,
- intensité du trafic réseau lors de l'exécution,
- résistance aux pannes (RQ1, RQ2, mais RQ3 oublié. . .).

Sous-programmes PL/SQL stockés

Illustration issue de la documentation *Oracle*

Figure 8–1 Calling a PL/SQL Stored Procedure



Fonctions PL/SQL stockées

Définition

Une fonction PL/SQL peut être définie par un utilisation pour encapsuler des instructions dont l'exécution dépend d'éventuels paramètres en entrée, sortie ou mise-à-jour. Un seul résultat est fourni à l'appelant, qui doit l'utiliser.

```
create function nomFonction[(param1 [{in | out | in out}] type1
                    [, ..., paramN [{in | out | in out}] typeM])]
return typeR {as | is}
  -- Définition des constantes et affectation obligatoire d'une valeur
  con1  constant typeCon1 {:= | default} valCon1 ;
  ...
  conL  constant typeCon1 {:= | default} valConL ;
  -- Définition des variables et affectation facultative d'une valeur
  var1  typeVar1 [not null] [{:= | default} valVar1] ;
  ...
  var0  typeVar0 [not null] [{:= | default} valVar0] ;
begin
  -- Code de la fonction
  instructions ;
  -- Toutes les branches du code doivent aboutir à un 'return'
  -- ou à une levée d'exception
  return valRetour ;
end [nomFonction] ;
/

-- Utiliser 'create or replace' pour éviter d'avoir à faire 'drop function'
-- lors de la mise au point du code
```


Fonctions PL/SQL stockées

Exemple : la fonction `getAgeEnJours`

```
-- Définition de la fonction : en-tête et corps
create function getAgeEnJours(pDateNaissance in date) return number is
begin
  return sysdate - pDateNaissance ;
end getAgeEnJours ;
/

-- Utilisation de la fonction dans une requête SQL
select tatouage, ddn, trunc(getAgeEnJours(ddn)) nbJours
from chien ;


-- Résultat
--   TATOUAGE DDN          NBJOURS
-- ---------- --------- ----------
--         51 15-FEB-99       5342
--         42 01-MAY-09       1614
--          3 04-MAY-10       1246
--        666 08-MAR-11        938
--       1664 12-DEC-12        293
```

Fonctions PL/SQL stockées

Exemple : la fonction récursive `factorielle`

```
-- Définition de la fonction : en-tête et corps
create function factorielle(pN in number) return number is
begin
  if pN < 0 then
    return null ;  -- @FIXME À traiter avec une exception plus tard
  elsif pN = 0 then
    return 1 ;
  else
    return pN * factorielle(pN - 1) ;  -- Appel récursif
  end if ;
end factorielle ;
/

-- Utilisation de la fonction dans une requête SQL
select factorielle(-42), factorielle(0), factorielle(6)
from dual ;


-- Résultat
-- FACTORIELLE(-42) FACTORIELLE(0) FACTORIELLE(6)
-- ---------------- -------------- --------------
--                                1            720

-- Octroi du privilège en exécution à tout utilisateur
grant execute on factorielle to public ;

-- Utilisation depuis un autre schéma que celui du créateur en préfixant
select cabanac.factorielle(6) from dual ;
```


Syntaxe du PL/SQL : similaire à ADA

Constantes et variables

Déclaration des constantes

```
nomConstante constant typeConstante {:= | default} valeur ;

-- Exemple
pi constant number := 3.1416 ;
```

Déclaration des variables

```
nomVariable typeVariable [not null] [{:= | default} valeur] ;

-- Exemple
vNomGagnant varchar(15 char) := 'Inconnu' ;
vNomGagnant chien.nom%type   := 'Inconnu' ;
```

Syntaxe du PL/SQL : similaire à ADA

Paramètres de sous-programmes

Déclaration des paramètres

```
nomParamètre {in | out | in out} typeParamètre
```

Avec les possibilités suivantes pour `typeParamètre` :

- `char`, `date`, `number`, `varchar2...` ☠ ne pas préciser la longueur !
- `boolean`
- `table.colonne%type` ⚠ à utiliser autant que possible !

```
create function totalPrimes(tatouage in chien.tatouage%type, pAnnee in number)
return number is
...

create function chienPresent(pNom in chien.nom%type, pDdn in chien.ddn%type)
return boolean is
...
```

NB1 : Convention de nommage → préfixes : `pParamètres` et `vVariable`.

NB2 : Pour débogguer, utiliser la commande `show errors` (abrégé `sho err`).

Syntaxe du PL/SQL : similaire à ADA

Aiguillage : `if` et `case`

Syntaxes

```
if cond1 then
  instructions1 ;
[
elsif cond2 then
  instructions2 ;
elsif cond3 then
  ...
]
[
else
  instructionsN ;
]
end if ;
```

```
case var1
  -- si var1 = exp1
  when exp1 then
    instructions1 ;
  [
  when exp2 then
    instructions2 ;
  when exp3 then
    ...
  ]
  [
  else
    instructionsN ;
  ]
end case ;
```

```
case
  when cond1 then
    instructions1 ;
  [
  when cond2 then
    instructions2 ;
  when cond3 then
    ...
  ]
  [
  else
    instructionsN ;
  ]
end case ;
```

```
-- Calcul de mention avec un 'if'
   if note >= 16 then return 'TB' ;
elsif note >= 14 then return 'B' ;
elsif note >= 12 then return 'AB' ;
elsif note >= 10 then return 'P' ;
else                  return null ;
end if ;
```

```
-- Calcul de mention avec un 'case'
case
  when note >= 16 then return 'TB' ;
  when note >= 14 then return 'B' ;
  when note >= 12 then return 'AB' ;
  when note >= 10 then return 'P' ;
  else                 return null ;
end case ;
```

Syntaxe du PL/SQL : similaire à ADA

Répétition : `while` et `loop`

Syntaxes

```
-- tantque ... faire
--   ...
-- fin tantque ;
while condition
loop
  instructions ;
end loop ;
```

```
-- faire
-- ...
-- tantque ;
loop
  instructions ;
  exit [when condition] ;
end loop ;
```

```
-- pour i allant de valInf à valSup faire
-- ...
-- fin pour
for compteur in [reverse] valInf .. valSup
loop
  instructions ;
end loop ;
```

```
-- Exécution d'un bloc PL/SQL anonyme
declare
  n     number := 6 ; -- n >= 0
  fact number := 1 ;
begin
  while n > 1 loop
    fact := fact * n ;
    n := n - 1 ;
  end loop ;
  dbms_output.put_line(fact) ;
end ;
/
```

```
-- Exécution d'un bloc PL/SQL anonyme
declare
  n     number := 6 ; -- n >= 0
  fact number := 1 ;
begin
  for i in 2 .. n loop
    fact := fact * i ;
  end loop ;
  dbms_output.put_line(fact) ;
end ;
/
```

NB : `dbms_output.put_line` affiche une chaîne (cf. diapositive 32).

Procédures PL/SQL stockées

Définition

Une procédure PL/SQL peut être définie par un utilisateur pour encapsuler des instructions dont l'exécution dépend d'éventuels paramètres en entrée, sortie ou mise-à-jour. Aucune valeur n'est retournée à l'appelant.

```
create procedure nomProcedure[(param1 [{in | out | in out}] type1
                          [, ..., paramN [{in | out | in out}] typeM])]
{as | is}
  -- Définition des constantes et affectation obligatoire d'une valeur
  con1  constant typeCon1 {:= | default} valCon1 ;
  ...
  conL  constant typeCon1 {:= | default} valConL ;
  -- Définition des variables et affectation facultative d'une valeur
  var1  typeVar1 [not null] [{:= | default} valVar1] ;
  ...
  varO  typeVarO [not null] [{:= | default} valVarO] ;
begin
  -- Code de la fonction
  instructions ;
end [nomProcedure] ;
/

-- Utiliser 'create or replace' pour éviter d'avoir à faire 'drop procedure'
-- lors de la mise au point du code
```

Procédures PL/SQL stockées

Exemple : la procédure enregistrerChien

```
-- Enregistre un chien, envoie un mail de confirmation à son propriétaire,
-- envoie une notification au vétérinaire de la Société Canine et
-- inscrit automatiquement le chien au prochain concours de Toulouse
-- organisé par la Société
create procedure enregistrerChien(pTatouage in chien.tatouage%type,
                                  pNomChien in chien.nom%type,
                                  pDdn in chien.ddn%type,
                                  pSexe in chien.sexe%type,
                                  pIdR in race.idR%type,
                                  pIdA in adherent.idA%type) is
begin
  -- Inscription du chien
  insert into chien(tatouage, nom, ddn, sexe, idA, idR)
  values(pTatouage, pNomChien, pDdn, pSexe, pIdR, pIdA) ;
  -- Envoi du mail de confirmation à l'adhérent
  envoyerMailAdhérent(pIdA, pTatouage) ;
  -- Envoi d'une notification au vétérinaire de la Société Canine
  envoyerMailVétérinaire(pIdA, pTatouage) ;
  -- Inscription au concours de Toulouse organisé par la Société Canine
  inscrireAuConcoursDeToulouse(pTatouage) ;
end enregistrerChien ;
/
```

```
-- Appel de la procédure : call, exec ou execute
call enregistrerChien('3615ALLU', 'Derrick', to_date('13/04/2012', 'DD/MM/YYYY'), 'M', 'CA', 51) ;
exec enregistrerChien('ZEROC00L', 'Rex', to_date('09/12/2011'), 'M', null, 999) ;
execute enregistrerChien('CAFEBABE', 'Rexona', to_date('09/12/2011'), 'F', null, 999) ;
```


Paquetages PL/SQL

Définition

Un paquetage PL/SQL est une bibliothèque logicielle qui rassemble des constantes et des sous-programmes (notamment). La création d'un paquetage est réalisée en deux étapes :

1. `create package nomPaquetage as ...`
2. `create package nomPaquetage body as ...`

La suppression est réalisée *via* `drop package [body] nomPaquetage`.

```
create package nomPaquetage {as | is}
  constante1 ;
  constante2 ;
  en-tête de sous-programme1 ;
  en-tête de sous-programme2 ;
  ...
  en-tête de sous-programmeN ;
end [nomPaquetage] ;
/
```

```
create package body nomPaquetage {as | is}
  -- partie publique
  en-tête et corps de sous-programme1 ;
  en-tête et corps de sous-programme2 ;
  ...
  -- partie privée
  en-tête et corps de sous-programmeN ;
  ...
  en-tête et corps de sous-programmePrivé1 ;
  en-tête et corps de sous-programmePrivéM ;
end [nomPaquetage] ;
/
```

Paquetages PL/SQL

Illustration issue de la documentation *Oracle*

Figure 8–2 Calling Subprograms in a PL/SQL Package

Paquetages PL/SQL

Exemple 1 : le paquetage interne `dbms_output`

Définition

Le paquetage `dbms_output` rassemble des sous-programmes dédiés à l'affichage d'informations sur la console du serveur de BD. Les sorties se font *via* un tampon (*buffer*) qui est désactivé par défaut.

Quelques procédures de `dbms_output` parmi les plus fréquemment utilisées :

- `enable()` Active l'affichage du tampon (*buffer*)
- `put(s)` Affiche la chaîne `s`
- `put_line(s)` Affiche la chaîne `s` et retourne à la ligne
- `new_line()` Retourne à la ligne

```
-- Exemple d'utilisation dans un bloc PL/SQL anonyme
begin
  dbms_output.enable() ;  -- équivalent de "set serveroutput on" dans sqlplus
  dbms_output.put('Nous sommes le '||sysdate||'. ') ;
  dbms_output.put_line('Utilisateur connecté '||user) ;
end ;
/

-- Nous sommes le 01-OCT-13. Utilisateur connecté CABANAC
```

Paquetages PL/SQL

Exemple 2 : le paquetage interne dbms_random

Définition

Le paquetage dbms_random rassemble des sous-programmes dédiés à la génération de valeurs aléatoires.

Quelques procédures de dbms_random parmi les plus fréquemment utilisées :

`initialize(g)`	Procédure initialisant le générateur avec une graine `g` (*seed*)
`normal()`	Fonction générant une valeur aléatoire suivant la loi normale
`random()`	Fonction générant une valeur aléatoire $v \in \left[2^{-31}; 2^{31}\right[$
`value()`	Fonction générant une valeur aléatoire $v \in [0; 1[$
`value(a, b)`	Fonction générant une valeur aléatoire $v \in [a; b[$
`string(c, l)`	Fonction générant une chaîne aléatoire de longueur `l` selon une configuration `c` (cf. documentation du paquetage).

```
select dbms_random.value() aleas, dbms_random.string('a', 8) mdp from dual ;

--       ALEAS MDP
-- ---------- ----------------
-- 0.8980590597 cNlyacRQ
```


Sources

Les illustrations sont reproduites à partir des documents suivants :

- Oracle Database Documentation Library 11.1
  (http://www.oracle.com/pls/db111/homepage)
  - Oracle® Database : Concepts 11g Release 1 (11.1) B28318-06
- Oracle Database Documentation Library 11.2
  (http://www.oracle.com/pls/db112/homepage)
  - Oracle® Database : Concepts 11g Release 2 (11.2) E40540-01
- SQL pour Oracle – 3e édition, C. Soutou, Eyrolles (2008)