Mémento SQL

Offert par sql spot

et ORSYS formation

Frédéric Brouard

Synoptique ordre SELECT
Langage SQL
Norme : ISO/IEC 9075 (2015)

ordre SELECT : extraction des données

SELECT	données retournées
FROM	source des données
WHERE	filtre sur les données des tables
GROUP BY	regroupement (sous-ensembles)
HAVING	filtre sur les résultats agrégés
ORDER BY	tri du résultat

Clause SELECT, syntaxe

```
SELECT { [ALL] | DISTINCT }
  élément1 [ [AS] aliasColonne1 ] ,
    élément2 [ [AS] aliasColonne2 ], ...
      élémentN [ [AS] aliasColonneN ]
<élément i > ::=
  {       [ { nomTable | aliasTable } . ] colonne |
          constante |
          fonction |
          expression |
          [ { nomTable | aliasTable } . ] *    }
```

REMARQUES

ALL (par défaut) laisse d'éventuels doublons.
DISTINCT élimine les doublons.
Oracle : ne permet pas l'utilisation de * avec d'autres expressions sauf si * est préfixé d'un alias.

L'ordre **SELECT** minimal est constitué des clauses **SELECT** et **FROM**. Certains **SGBDR** permettent d'utiliser la clause **SELECT** seule ne rapportant qu'une seule ligne. D'autres utilisent une pseudo-table de nom **DUAL** (**Oracle**).

SELECT, exemple

```
SELECT DISTINCT
       NOM      AS PATRONYME,
       PRENOM,
       CLIENTS.*,
       'FRANCE' AS PAYS,
       CURRENT_DATE
                AS JOUR,
       CONCAT(LEFT(NOM,1), LEFT(PRENOM,1))
                AS INITIALES,
       DATE_NAIS jour,
       12.5     AS REDUCTION_POURCENT
FROM...
```

Clause FROM, syntaxe

Jointure généralisée :

```
FROM  table1 [ [ AS ] aliasTable1 ]
    { [ INNER ] |
        { LEFT | RIGHT | FULL } [ OUTER ] }
            JOIN  table2  [ [ AS ] aliasTable2 ]
    ON <prédicat_jointure>
```

Jointure "naturelle" :

```
  FROM  table1 [ [ AS ] aliasTable1 ]
     NATURAL { [ INNER ] |
       { LEFT | RIGHT | FULL } [ OUTER ] }
           JOIN  table2  [ [ AS ] aliasTable2 ]
    [ USING  <liste_colonne_jointure> ]
```

Produit cartésien :

```
  FROM  table1 [ [ AS ] aliasTable1 ]
     CROSS  JOIN  table2  [ [ AS ] aliasTable2 ]
```

Application d'une fonction table :

{ **CROSS** | **OUTER** } **APPLY** <fonction_table>(...)

LATERAL <fonction_table>(...)

<table> [[AS] alias] **TABLESAMPLE** (n { **PERCENT** | **ROWS** })

REMARQUES

NATURAL JOIN : le prédicat de jointure n'est pas nécessaire (dans ce cas, la jointure s'opère sur toutes les colonnes de même nom dans les tables - si vous utilisez USING, vous devrez énumérer les colonnes à joindre).

NATURAL JOIN à éviter. Problématiques diverses (jointures triangulaires, évolution de la base dangereuse...).

Une même table peut être citée plusieurs fois à condition d'être représentée par des alias distincts.

CROSS JOIN n'a pas de prédicat de jointure car il effectue le produit cartésien («multiplication» relationnelle).

APPLY permet «d'appliquer» la fonction table dont le résultat dépend de paramètres pour lesquels les arguments sont ceux de colonne(s) ou d'expression(s) venant de tables précitées.
CROSS APPLY pour le produit cartésien et **OUTER APPLY** en produit cartésien «externe».

LATERAL est un équivalent de **CROSS APPLY**.

TABLESAMPLE permet d'obtenir un échantillon aléatoire de cardinalité approximative.

Oracle utilise **SAMPLE** proche du **TABLESAMPLE**. **MySQL** n'implémente pas **TABLESAMPLE**.

REMARQUES (suite)

SQL Server ne permet pas **NATURAL JOIN**.
Oracle ne permet pas **AS** pour définir un alias de table.
PostGreSQL et **MySQL** ne connaissent pas **APPLY**.
Pour **PostGreSQL**, utilisez **LATERAL** au lieu de **CROSS APPLY**.

FROM, exemples

```
FROM T_CLIENT

FROM T_FACTURE AS F
     JOIN T_CLIENT AS C
          ON F.CLI_ID = C.CLI_ID

FROM T_FACTURE AS F TABLESAMPLE (1 PERCENT)
     INNER JOIN T_CLIENT AS C
           ON F.CLI_ID = C.CLI_ID

FROM T_FACTURE AS F
     NATURAL JOIN T_CLIENT AS C

FROM T_FACTURE AS F
     NATURAL JOIN T_CLIENT AS C
          USING (CLI_ID)

FROM T_CLIENT AS C
     LEFT OUTER JOIN T_EMAIL AS E
          ON C.CLI_ID = E.CLI_ID

FROM T_CLIENT AS C
     LEFT JOIN T_EMAIL AS E
           ON C.CLI_ID = E.CLI_ID

FROM T_EMAIL AS E
     RIGHT OUTER JOIN T_CLIENT AS C
           ON E.CLI_ID = C.CLI_ID

FROM T_CLIENT AS C
     FULL OUTER JOIN T_PROSPECT AS P
          ON E.CLI_ID = C.PSP_ID

FROM T_EMPLOYE AS E
     LEFT OUTER JOIN T_EMPLOYE AS C
          ON E.EMP_ID_CHEF = C.EMP_ID

FROM T_CLIENT AS C
     LEFT OUTER JOIN T_EMAIL AS E
          ON C.CLI_ID = E.CLI_ID
     LEFT OUTER JOIN T_TELEPHONE AS T
          ON C.CLI_ID = T.CLI_ID
     INNER JOIN T_FACTURE AS F
          ON C.CLI_ID = F.CLI_ID
        INNER JOIN T_DETAIL_FACTURE AS DF
              ON F.FAC_ID = DF.FAC_ID
        LEFT  OUTER JOIN T_TVA AS TV
              ON DF.TVA_ID = TV.TAV_ID
```

FROM, exemples (suite)

```
FROM T_MOIS AS M
     CROSS JOIN T_ANNEE AS A

FROM T_FACTURE AS F
     CROSS APPLY F_ECHEANCE(F.FAC_TYPE,F.FAC_DATE)
```

Clause WHERE, syntaxe

WHERE <prédicat_WHERE>

Opérateurs de comparaison

(e1 et e2 sont deux expressions)

e1 = e2	est égal à
e1 > e2	est supérieur à
e1 >= e2	est supérieur ou égal à
e1 < e2	est inférieur à
e1 <= e2	est inférieur ou égal à
e1 <> e2	est différent de

e1 **BETWEEN** valeur_debut
AND valeur_fin
e1 **IN** (valeur1, valeur2, … valeurN)
e1 **LIKE** <motif> [**ESCAPE** 'c']
<motif> ::= <chaine_caractère_like>
<caractère_like> ::= { caractère | <joker> }
<joker> ::= { % | _ }

REMARQUES

Certains SGBDR utilisent aussi != pour différent de.
BETWEEN est une fourchette de valeurs inclusives.
ALL et **ANY** complètent **IN** pour des comparaisons « non équi ».
Jokers du **LIKE** : _ un et un seul caractère, % n'importe quelle chaîne de caractères même vide.
ESCAPE 'c' : le caractère c sert de séquence d'échappement. Le caractère qui suit est pris au sens littéral.

Opérateurs logiques

AND OR	et, ou logique
NOT	négation
IS [NOT] NULL	absence, présence de valeur

REMARQUES

Sans parenthèses entre des conditions, **AND** est prioritaire par rapport à **OR**

WHERE, exemples

```
WHERE CLI_ID = 387

WHERE FAC_DATE > '1987-06-25'
  AND CLI_TYPE = 'SARL'

WHERE FAC_DATE BETWEEN '2014-01-10
                   AND '2015-01-09'

WHERE CLI_TYPE IN ('SARL', 'SA', 'EURL')

WHERE CLI_NOM LIKE '_UPON%'

WHERE TABLE_NAME LIKE 'T^_%' ESCAPE '^'

WHERE CLI_PRENOM IS NULL

WHERE (   CLI_NOM = 'DUPONT'
       OR CLI_NOM = 'DUPOND' )
  AND CLI_PRENOM = 'Marcel'

WHERE FAC_REMISE IS NOT NULL
  AND NOT FAC_QUANTITE = 1
```

Fonctions d'agrégation

MAX(e)	maximum
MIN(e)	minimum
COUNT([**DISTINCT**] e)	nombre
AVG(([**DISTINCT**] n)	moyenne
SUM(([**DISTINCT**] n)	somme
COUNT(*)	nombre de lignes
STDEV[**P**](**DISTINCT** n)	écart type
VAR[**P**](**DISTINCT** n)	variance
GROUPING(<liste_colonne>)	sous-total (**CUBE**/**ROLLUP**)**LISTAGG**, **GROUP_CONCAT**, **STRING_AGG**	concaténation

e : expression quelconque, n : expression numérique
DISTINCT ne compte pas les doublons.
Les NULL sont ignorés des calculs d'agrégats.

Clause GROUP BY, syntaxe

GROUP BY <liste_expression>, peut aussi contenir des groupages **ROLLUP** et **CUBE** (sous-totaux)
En principe, toutes les expressions non agrégées de la clause **SELECT** doivent figurer dans la clause **GROUP BY**.

REMARQUES

MySQL ne connaît pas **CUBE** ni **ROLLUP** et ne renvoie pas d'erreur en cas de **GROUP BY** incorrect, mais le résultat est hasardeux.

GROUP BY sans agrégat dans **SELECT** ou **HAVING** n'a aucun intérêt.

Clause **HAVING**, syntaxe

HAVING <prédicat_HAVING>
Le prédicat permet de filtrer les résultats des calculs agrégés.

Agrégats et GROUP BY / HAVING, exemples

```
SELECT MAX(FAC_DATE)
FROM   T_FACTURE

SELECT CLI_ID, MIN(FAC_DATE)
FROM   T_FACTURE
GROUP  BY CLI_ID

SELECT CLI_PRENOM, COUNT(*)
FROM   T_FACTURE
GROUP  BY CLI_ID
HAVING COUNT(*) > 1

SELECT YEAR(FAC_DATE), SUM(FAC_QUANTITE)
       COUNT(DISTINCT CLI_ID)
FROM   T_FACTURE
GROUP  BY YEAR(FAC_DATE)

SELECT TAILLE, COULEUR, TYPE, COUNT(*), SUM(PRIX)
FROM   T_VETEMENT
GROUP  BY ROLLUP(TAILLE, COULEUR), TYPE
```

Opérations ensemblistes

<req_SELECT_1> { **UNION** [**ALL**] | **INTERSECT** | **EXCEPT** } <req_SELECT_2>

REMARQUES

Les requêtes **SELECT** doivent avoir des colonnes « compatibles » en nombre et en types.
UNION dédoublonne. **UNION ALL** permet les doublons.
INTERSECT et **EXCEPT** sont prioritaires devant **UNION**.
Oracle utilise **MINUS** à la place de **EXCEPT**

Opérations ensemblistes, exemples

```
SELECT nom, prenom, 'physique' AS NATURE
FROM   T_PERSONNE
UNION  ALL
SELECT raison_sociale, enseigne, 'morale'
FROM   T_PERSONNE

SELECT nom
FROM   T_CLIENT
INTERSECT
SELECT nom
FROM   T_PROSPECT

SELECT date_ouverture
FROM   T_PLANING
EXCEPT
SELECT date_fact
FROM   T_FACTURES
```

Clause **ORDER BY**, syntaxe

ORDER BY <liste_expression_ORDER>

<liste_expression_ORDER> ::=
{ <posSelect1> | <exp1>} [{ [**ASC**] | **DESC** }]
[COLLATE nom_collation]
{ <posSelect2> | <ex2> } [{ [**ASC**] | **DESC** }]
[COLLATE nom_collation]
OFFSET n **ROW**[**S**] [**FETCH** { **FIRST** | **NEXT** } m **ROW**[**S**] **ONLY**]

REMARQUES

posSelect est la position ordinale (numérique) de l'expression dans la clause **SELECT**. À éviter.
Le tri *externe* (colonne ne figurant pas dans le **SELECT**) est supporté avec certaines conditions (pas de **DISTINCT**...)
COLLATE uniquement pour expressions littérales.
ASC est l'ordre par défaut (ascendant).
MySQL ne supporte pas **OFFSET**/**FETCH**

ORDER BY, exemples

```
ORDER BY 1, 2, nom ASC, date_naiss DESC

ORDER BY nom COLLATE French_CI_AI
         OFFSET 10 ROWS FETCH NEXT 1 ROW ONLY

ORDER BY nom COLLATE French_CI_AI,
         prenom ASC,
         date_naissance DESC
```

Collations

e1 **COLLATE** nom_collation_paramètres
nom_collation : nom de langue ou encodage
paramètres :
- **CS** ou **CI** : (Case Sensitive / Insensitive) sensible ou insensible à la casse (maj/min)
- **AS** ou **AI** : (Accent Sensitive / Insensitive) sensible ou insensible aux diacritiques (accents, cédille, ligature...)
- **KS** : sensible aux kana types du japonais (Kanatype Sensitive)
- **WS** : sensible à la largeur (Wide Sensitive)

Chaque colonne de table a sa propre collation et l'opérateur **COLLATE** s'utilise dans toute expression littérale.

REMARQUES

Seul **SQL Server** supporte **KS**, **WS** et les 4 combinaisons **C{I|S}** / **A{I|S}**.
Oracle supporte **COLLATE** depuis la version 12c Release 2.
Sinon utilisez le concept de NLS (NLS_COMPARE, NLS_SORT).
PostGreSQL et **MySQL** supportent mal les collations.

Opérateurs mathématiques

+ -	addition, soustraction
* /	multiplication, / division
%	modulo (reste de la division entière)

REMARQUES

La fonction **MOD** remplace parfois %.
* et / sont prioritaires devant + et -.

Opérateur de chaînes de caractères

||	concaténation

REMARQUES

SQL Server utilise + pour la concaténation.
CONCAT(...) est une fonction équivalente.

Fonctions de fenêtrage

Utilise une clause **OVER** de fenêtrage comportant :
- **PARTITION** BY <liste_part>
- **ORDER BY** <liste_ord>
- { **RANGE** | **ROWS** } <fenêtre>

Prend en compte :
les fonctions d'agrégation (**MAX**, **MIN**, **SUM**, **COUNT**...),
les fonctions d'ordonnancement (**ORDER BY** obligatoire) :
- **RANK**() rang
- **DENSE_RANK**() rang dense
- **ROW_NUMBER**() n° de ligne
- **NTILE**(n) paquets
- **PERCENT_RANK**() rang en pourcentage]0...1[

les fonctions analytiques :
- **LEAD, LAG** valeur précédente, suivante
- **FIRST_VALUES** première valeur
- **LAST_VALUE** dernière valeur
- **NTH_VALUE** nième valeur

les fonctions de distribution :
- **CUM_DIST** distribution cumulative]0...1[
- **PERCENTILE_DISC** percentile discret
- **PERCENTILE_CONT** percentile continu

qui nécessite une clause **WITHIN**.

REMARQUES

Pour filtrer le résultat d'une fonction fenêtrée, utilisez une sous-requête.
<fenetre> supporte différents mots clefs (**BETWEEN**, **PRECEDING**, **FOLLOWING**, **CURRENT ROW**, **UNBOUNDED**...).

Fonctions de fenêtrage, exemple

```
SELECT *, COUNT(*) OVER()
FROM   T_PERSONNE

SELECT *,
       COUNT(DISTINCT date_naiss) OVER()
FROM   T_PERSONNE

SELECT *,
       MAX(date_naiss) OVER() AS JEUNE,
       MIN(date_naiss) OVER() AS VIEUX
FROM   T_PERSONNE

SELECT *,
       COUNT(*) OVER(PARTITION BY prenom)
FROM   T_PERSONNE

SELECT *,
       SUM(montant)
          OVER(PARTITION BY YEAR(fac_date)
               ORDER BY fac_date) AS cumul
FROM   T_FACTURE

SELECT * FROM (
SELECT *, RANK() OVER(PARTITION BY MATIERE
                  ORDER BY NOTE DESC) AS RANG
FROM   NOTATION) AS N
WHERE  RANG <= 3


SELECT *,
       RANK() OVER(PARTITION BY service
                   ORDER BY salaire DESC)
FROM   T_EMPLOYE

SELECT *, ROW_NUMBER()
          OVER(ORDER BY date_naissance,
                        nom,
                        prenom) AS NUM
FROM   T_PERSONNE

SELECT *,
       SUM(FAC_TOTAL_HT)
          OVER(PARTITION BY YEAR(FAC_DATE)
               ORDER BY FAC_DATE
               ROWS BETWEEN UNBOUNDED
                            PRECEDING
               AND CURRENT ROW) AS CUMUL
FROM   T_FACTURE

SELECT *,
       LAG(FAC_TOTAL)
          OVER(PARTITION BY YEAR(FAC_DATE)
               ORDER BY FAC_DATE)
          AS TOTAL_PRECEDENT
FROM   T_FACTURE
```

Opérateur de choix CASE

CASE valué :

```
CASE <expression_test>
  WHEN <valeur1> THEN <val_retour_1>
  WHEN <valeur2> THEN <val_retour_2>
...
  WHEN <valeurN> THEN <val_retour_N>
  [ ELSE <valeur_retour_defaut>
END
```

CASE généralisé :

```
CASE
  WHEN <prédicat1> THEN <val_retour_1>
  WHEN <prédicat2> THEN <val_retour_2>
...
  WHEN < prédicatN> THEN <val_retour_N>
  [ ELSE <valeur_retour_defaut>
END
```

REMARQUES

Le premier **WHEN** vrai renvoie la valeur retour, défaut **NULL**.
Oracle utilise aussi la fonction **DECODE** pour le **CASE** valué et **SQL Server IIF** pour un choix binaire.

Opérateur CASE, exemple

```
SELECT CASE CIVILITE WHEN 'Mme.' THEN 'Femme'
                     WHEN 'M.' THEN 'Homme' END

SELECT CASE WHEN LEFT (TEL_NUMERO, 2) IN ('06', '07')
            THEN 'Internet' ELSE 'Filaire' END
```

Alias et identifiants SQL (noms des objets)

Caractères autorisés :
- Lettres de A à Z sans accent, casse indifférente ;
- Chiffre de 0 à 9 ;
- Blanc souligné (*underscore*).

Ne pas commencer par un chiffre.
Ne pas être un mot clef de SQL (**DATE**, **TYPE**, **CASE**...) sauf à être entouré de guillemets ("). À éviter.

REMARQUES

La plupart des SGBDR acceptent des caractères illicites dans les identifiants (accents, blanc, ligature...). À éviter absolument.
En sus des guillemets, **SQL Server** utilise les crochets [ ] pour délimiter les identifiants hors standards et **MySQL** des accents (` ´). À éviter pour la portabilité.

Type de données SQL et expressions

Chaînes de caractères (ex : **'l''exemple'**) :
- **CHAR**(n), **VARCHAR**(n) : encodage **ASCII**
- **NCHAR**(n), **NVARCHAR**(n) : encodage **UNICODE**

Nombres (ex : **-123.456789E-53**) :
- **INT**, **SMALLINT**, **BIGINT** : entiers
- **REAL**, **FLOAT**, **DOUBLE PRECISION** : réels
- **DECIMAL**(m, p), **NUMERIC**(m, p) : décimaux

Temps (ex : **'2015-01-31 23:19:33.93'**) :
- **DATE** : date de 1/1/1 à 9999/12/31,
- **TIME**(s) [**WITH TIME ZONE**] : heure [0h...24[
- **TIMESTAMP**(s) [**WITH TIME ZONE**] : date + heure
- **INTERVAL** <période_large> **TO** <période_fine> : durée

Binaires : **BIT**(n), **BITVARYING**(n)
Spéciaux : **XML**, **GEOMETRY**, **GEOGRAPHY**.

n : long., m : mantisse, p : précision, s : nb de sec.
<période>::= { **YEAR** | **MONTH** | **DAY** | **HOUR** | **MINUTE** }
WITH TIME ZONE : préciser le fuseau horaire

REMARQUES

SQL Server utilise **DATETIME2** à la place de **TIMESTAMP**.
SQL Server et **MySQL** ne supportent pas **INTERVAL**
Oracle ne supporte pas **TIME** ni **DATE**
Pour **Oracle** utilisez [**N**]**VARCHAR2** à la place de [**N**]**VARCHAR**
Certains SGBDR utilisent **BOOLEAN** pour le **BIT**

Sous-requêtes (requêtes imbriquées)

Requête **SELECT** (interne) placée à l'intérieur d'une requête dite externe. Parenthèses obligatoires.

La possibilité de placement de la sous-requête dépend de la forme du résultat :
- Requête « point » (renvoie une seule valeur) => *constante* dans **SELECT, WHERE, ON, HAVING**...
- Requête « liste » (renvoie une seule colonne avec plusieurs valeurs) => opérateur **IN, ALL** ou **ANY**
- Requête « table » (renvoie plusieurs colonnes avec plusieurs lignes) => clause **FROM**
- Requête « vide » (ne renvoie aucun résultat) => opérateur [**NOT**] **EXISTS**. Nécessite une corrélation.

WITH (CTE) : permet de déclarer des expressions de table (vue à usage local), utilisables en requête finale et réaliser des requêtes récursives (**UNION ALL** obligatoire).

La corrélation des sous-requêtes impose qu'une valeur de la requête externe soit exploitée par la requête interne (sous-requête).

REMARQUES

SQL Server impose un alias de table pour une sous-requête dans la clause **FROM**.

Sous-requêtes, exemples

```
--> nom, prénom de la plus jeune personne
SELECT nom, prenom
FROM   T_PERSONNE
WHERE  date_naissance =
           (SELECT MAX(date_naissance)
            FROM   T_PERSONNE);

--> liste des homonymes
SELECT *
FROM    T_PERSONNE
WHERE   nom IN (SELECT prenom
                FROM   T_PERSONNE
                GROUP  BY prenom
                HAVING COUNT(*) > 1);

--> homonymes avec date de naissance égale
SELECT *
FROM    T_PERSONNE AS P
        JOIN (SELECT nom, date_naissance
              FROM   T_PERSONNE
              GROUP  BY nom, date_naissance
              HAVING COUNT(*) > 1) AS T
           ON P.nom = T.nom AND
              P.date_naissance = T.date_naissance;

--> homonymes a date de naissance égale (version 2)
WITH T AS
(SELECT nom, date_naissance AS dn
 FROM   T_PERSONNE
 GROUP  BY nom, date_naissance
 HAVING COUNT(*) > 1
)
SELECT *
FROM   T_PERSONNE AS P
       JOIN T ON P.nom = T.nom
                 AND P.dn = T.dn;

--> clients n'ayant pas été facturé en 2015
SELECT *
FROM   T_CLIENT AS C
WHERE  NOT EXISTS (SELECT 1
                   FROM   T_FACTURE AS F
                   WHERE  C.CLI_ID = F.CLI_ID
                     AND  YEAR(FAC_DATE) = 2014);

--> affiche les nombres de 1 à 100 de manière récursive
WITH T AS (SELECT 1 AS N
           UNION  ALL
           SELECT N + 1 FROM T
           WHERE  N < 100 )
SELECT * FROM T
```

Fonctions scalaires

Sur les chaînes de caractères :
UPPER, **LOWER** : mise en majuscule, minus
[L|R]TRIM : suppression des blancs
LEFT, **RIGHT** : partie de droite, gauche
SUBSTR[ING] : sous-chaînes
POSITION, CHARINDEX : place d'une chaîne dans une chaîne
CHARACTER_LENGTH, **OCTET_LENGTH, LEN** : longueur
REPLACE, **REVERSE** : remplace, renverse
NCHR/NCHAR : caractères **UNICODE** d'un code
CHR/CHAR : caractère **ASCII** d'un code
ASCII : code d'un caractère **ASCII**
UNICODE : code d'un caractère **UNICODE**
CONCAT : concatène des chaînes de car.
SOUNDEX, DIFFERENCE : consonnance, différence de consonnances
LPAD, RPAD, REPLICATE, SPACE : remplit, réplique
TRANSLATE : remplacement de caractères

Sur des temporels :
CURRENT_DATE, **CURRENT_TIME** : date, heure, courante (aussi **GETDATE**, **SYSDATE**, **NOW** en fonction du SGBDR)
CURRENT_TIMESTAMP : date+heure courante
EXTRACT (**SQL Server DATEPART**) : partie de date
DAY, **MONTH**, **YEAR** : jour, mois, année
HOUR, **MINUTE**, **SECOND** : heure, min., sec.
DATE[_]ADD : ajout de durée à un temporel
DATE[_]DIFF : durée entre deux temporels
DATENAME : nom d'une partie de date
EOMONTH : date de fin de mois

Sur des nombres :
FLOOR : valeur entière plancher
CEIL[ING] : valeur entière plafond
ROUND : arrondi à n décimales
SIGN : signe (-1, 0 ,1)
SQUARE : élévation au carré
SQRT : racine carrée
POWER : élévation à la puissance
RAND, **RANDOM** : valeur aléatoire]0..1[
MOD (%) : modulo
PI : valeur de Pi

Génériques :
CAST : transtype (**Oracle** : **TO_CHAR/DATE/NUMBER**)
COALESCE : dénullifie (**Oracle** : **NVL**)
NULLIF : nullifie
USER : utilisateur SQL
SYSTEM_USER : utilisateur système
CURRENT_USER : utilisateur courant

Mémento SQL

Offert par sql spot

et ORSYS formation

Frédéric Brouard

Synoptique ordre SELECT
Langage SQL
Norme : ISO/IEC 9075 (2015)

Une création : www.mofegraphic.com