I.U.T. de Nice Côte d’Azur

Département Informatique DUT S2

# Module M2106

TD n° 5

Introduction au langage PLSQL

**Objectif** : faire connaissance avec le langage PLSQL de Oracle : variables, types, fonctionnalités générale, programmation du côté client

**Note** : les exercices portent sur le schéma VIDEO et nous travaillons depuis SQLDEVELOPER

## 1. Utilisation des fonctions natives dans SQL

Note : à consulter la partie 3 « rappel de cours » pour faire ces exercices.

Vous devez vérifier si la base de données de teste a été bien définie sinon la créer avec le script

«1516\_videobuilding.sql »

* 1. Donner le nombre des casettes empruntés dans la période de février de toutes les années.

SELECT f.titre, f.rdate

FROM film f

WHERE To\_char(f.rdate, ‘MMDD’) Between ‘1217’ and ‘1231’

* 1. Donner la liste des titres, date de réalisation des films réalisés dans la période de Noël (1 semaine avant et une semaine après la date 24/12).

SELECT f.titre, f.rdate

FROM film f

WHERE To\_Char (f.rdate, ‘MMDD’) Between ‘1712’ and ’3112’

* 1. Donner la liste des noms de client, n° de cassette et date empruntée et le nombre des jours depuis cette date.

SELECT c.cnom, e.id\_cassette, e.date\_empt, Round (CURRENT\_date – e.date\_empt, 0) AS Temp

FROM client c, emprunt e

WHERE c.id\_client = e.id\_client

1.4. Donner le nombre de cassettes empruntées dans chaque mois. Le résultat est trié par mois.

SELECT count (e.id\_cassette) AS Nombre\_emprunts, To\_Char(e.date\_empt, ‘MM’) AS Mois

FROM emprunt e

GROUP BY To\_Char(e.date\_empt, ‘MM’)

ORDER BY To\_Char(e.date\_empt, ‘MM’)

1.5. Donner le nombre de cassettes n’ayant pas rendu dans le délai (21 jours).

Note : comparaison avec la valeur NULL : colonne IS NULL (exemple : date\_rendu IS NULL)

## 2. Blocs PLSQL anonymes

### Lettre de rappel aux clients non rendu cassette dans le délai

2.1. Réaliser un programme PLSQL qui pour chaque client ayant une cassette en cours d’emprunt dont le délai autorisé (21 jours) est dépassé, affiche une lettre selon le modèle suivante :

Lettre d’appel

Client : .Nom\_du\_client N° : id\_client

Cher(e) Client(e),

Vous avez emprunté la cassette numéro : id\_cassette depuis la date : date\_emprunt A ce jour, le délai autorisé (21 jours) est dépassé de : nombre\_j\_depassé jours.

Nous vous demandons de rendre cette cassette numéro : id\_cassette avant la date : date\_du\_jour +2. Dans le cas contraire, après cette date votre abonnement sera suspendu.

Cordialement.

----------------------

Note : les lettres sont séparée par une ligne « ---------------------------»

### Quelques statistiques sur la base « vidéo »

2.2. Réaliser un programme PLSQL qui affiche les statistiques selon la valeur de la variable « critere » définie et saisie dans le SQLPLUS pour sélectionner un des trois critères suivants :

* Critere = ‘1’ : afficher pour chaque film, son titre, nom et prénom du réalisateur et le nombre d’emprunts effectués de ce film. Le résultat est trié par l’ordre descendant de fréquences d’emprunts. Les films sont séparés par une ligne « -------------------------» Liste des films avec fréquences d’emprunts

Titre: .titre

Réalisateur : rprenom rnom -- séparés pas un blanc

Nombre d’emprunts : nombre-emprunts -- à calculer

----------------------------

* Critere = ‘2’ : afficher pour chaque client, son nom, la date de fin d’abonnement et le nombre d’emprunts effectués par ce client. Le résultat est trié par l’ordre descendant de fréquences d’emprunts. Les clients sont séparés par une ligne « -------------------------» Liste des clients avec fréquences d’emprunts

Nom: .nom

Date de la fin d’abonnement : date\_fin\_ab

Nombre d’emprunts : nombre-emprunts -- à calculer

----------------------------

* Critere = ‘3’ : afficher pour chaque cassette, son numéro, titre de film sur la cassette et le nombre d’emprunts de cette cassette. Le résultat est trié par l’ordre descendant de fréquences d’emprunts. Les cassettes sont séparées par une ligne « -------------------------» Liste des cassette avec fréquences d’emprunts n° de cassette: id\_cassette Titre du film sur la cassette : titre

Nombre d’emprunts : nombre-emprunts -- à calculer

----------------------------

Note : La solution n’affiche pas des films ou des clients ou des cassettes n’ayant participé à aucun emprunt.

### Recommandation des films de mêmes réalisateurs que ceux des films les plus empruntés (question supplémentaire)

2.3. Réaliser un programme PLSQL qui pour chaque film parmi les films les plus empruntés, affiche le titre du film, le nom et prénom du réalisateur, le nombre d’emprunts réalisés ainsi que la liste des autres titres de films réalisés par le même auteur selon le modèle suivante :

Un des films les plus téléchargés Titre: titre-du-film

Réalisateur : rprenom rnom -- séparés par un blanc

Nombre d’imprunts : nombre-emprunt -- à calculer

Liste d’autres films du même réalisateur : -- à calculer o Titre-films-1

* …
* Titre-film-n

----------------------------

**Note** : répéter le même modèle pour chaque filme parmi les plus empruntés (s’il y a plusieurs) séparer entre eux par une ligne « -----------------------»

**Indication** : utiliser le curseur de la question 2.1. pour trouver les films les plus empruntés et un deuxième curseur paramétré pour parcourir les films de même réalisateur

## 3. Rappel de cours

### 3.1. Structure d’un programme PLSQL : Bloc PLSQL

|  |  |  |
| --- | --- | --- |
| **Type** | **Propriété** | **Structure** |
| Bloc anonyme | Programme client non persistant | |  | | --- | | [DECLARE]  -- section déclaration facultative : Types, variables, constantes | | BEGIN  -- Section de traitement obligatoire | | [EXCEPTION]  -- Section de traitement d’erreurs facultative  END ; | |
| Bloc nommé | Programme  serveur, objet persistant de la base de donnée | |  | | --- | | PROCEDURE/FUNCTION/PACKAGE un\_nom [RETURN type] AS/IS  -- section déclaration facultative : Types, variables, constantes | | BEGIN  -- Section de traitement obligatoire | | [EXCEPTION]  -- Section de traitement d’erreurs facultative  END ; | |

3.2. **Affichage dans PLSQL :**

L’affichage dans un programme PLSQL est

* activé par la commande SQLPLUS SET SERVEROUTPUT ON
* assuré par 2 procédures spécifiques : DBMS\_OUTPUT.PUT() et DBMS\_OUTPUT.PUTLINE().

|  |  |
| --- | --- |
| **Procédure / COMMANDE** | **Propriété** |
| DBMS\_OUTPUT.PUT(argument) | Afficher la concaténation des chaines de caractères et des variables passées dans l’argument  L’opérateur de concaténation : « **||** » |
| DBMS\_OUTPUT.PUTLINE(argumen  t) | Afficher sur une nouvelle ligne, la concaténation des chaines de caractères et des variables passées dans l’argument |
| SET SERVEROUTPUT ON | Commande SQLPLUS à exécuter au début de la session de travail pour activer l’affichage de PLSQL |

### 3.3. Déclaration de types de données et de variables dans la section [DECLARE]

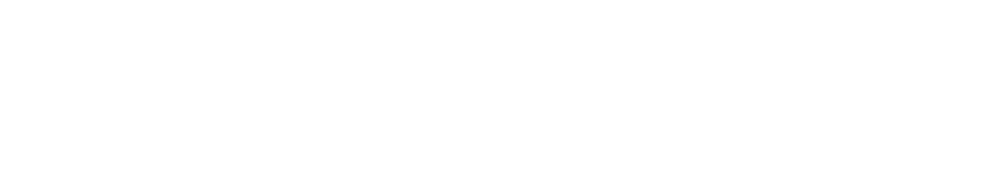
|  |  |
| --- | --- |
| **Déclaration** | **Explication** |
| Sous type | TYPE nom\_sous-type IS TYPE ;  SUBTYPE nom\_sous-type IS TYPE ;  Exemple : sous-type TypeNom  TYPE TypNom IS VARCHAR2(25) ; |
| Variable | nom\_variable TYPE [ := valeur\_par\_défaut] ;  Exemple : sous-type TypeNom  TYPE TypNom IS VARCHAR2(25) ; |
| Affectation | Opérateur d’affectation dans PLSQL est :   * Soit : Nom\_variable := expression ; * Soit dans une clause SQL : SELECT … INTO liste-variables |

**3.4.**

**Traitements conditionnels**



**Exemple d’un programme simple**



SET

SERVEROUTPUT

ON;

--

commentaire

:

initialisation

de

l’affichage

de

PLSQL

DECLARE

c varchar2(15) :='H e l l o World!';

BEGIN

DBMS\_OUTPUT.PUT\_LINE(c) ;

END;

/

--

commentaire

‘/’

marque

la

fin

d’un

programme

pour

lancer

la

compilation.

Deux instructions de traitement conditionnel :

-

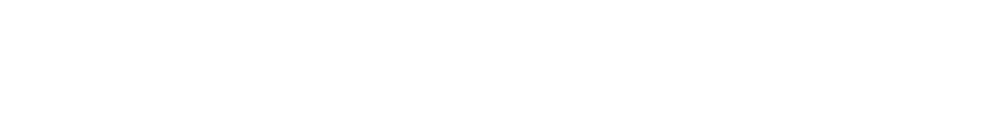
IF …. THEN …. ELSIF …. ELSE …. END IF;

-

CASE WHEN … THEN …. END CASE ;



**Modèles**



**IF**

/\* Condition \*/

**THEN**

/\* Instructions\*/

[

**ELSIF**

/\*Condition\*/

**THEN**

/\* Instructions\*/]

[

**ELSE**

/\* Instructions\*/]

**END IF;**

|  |
| --- |
| **CASE** /\* nom-variable \*/  **WHEN** /\*valeur\*/ **THEN** /\* Instructions\*/  [**WHEN** /\*valeur\*/ **THEN** /\* Instructions\*/] …  [**WHEN** /\*valeur\*/ **THEN** /\* Instructions\*/]  [**ELSE** /\* Instructions par défaut\*/] **END CASE;** |

### 3.5. Traitements répétitifs

Les traitements répétitifs sont assurés par :

- Une boucle de base LOOP …. END LOOP; - Des étiquettes WHILE et FOR.

|  |
| --- |
| **Modèles** |
| **Boucle “REPETER” généralisée**  **LOOP**  /\*instructions \*/  **EXIT WHEN** /\*conditions\*/  /\* Instructions\*/ **END LOOP;** |
| **Boucle “POUR” traditionnel**  **FOR** /\*nom\_variable\*/ **IN** /\*borne inférieure\*/ .. /\*borne supérieure\*/ **LOOP**  /\*instructions \*/ **END LOOP;** |
| **Boucle “POUR” avec requête SQL**  **FOR** /\*nom\_variable\*/ **IN** /\*Nom\_curseur ou clause SQL SELECT complète \*/ **LOOP**  /\*instructions \*/ **END LOOP;** |
| **Boucle “TANQUE”**  **WHILE** /\*condition\*/  **LOOP**  /\*instructions \*/ **END LOOP;** |

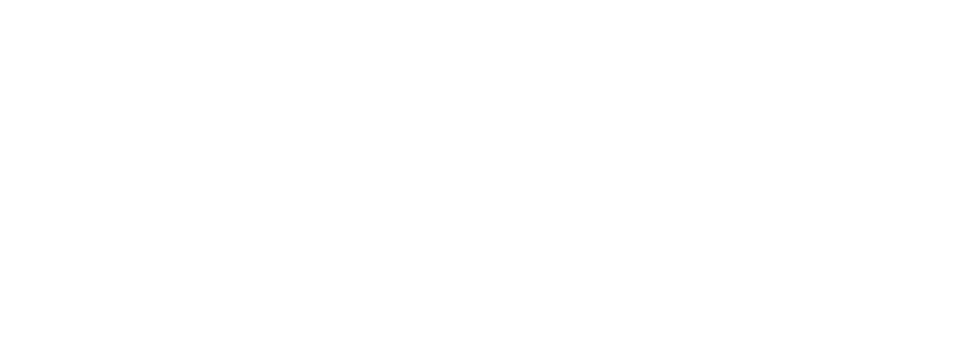
### 3.6. Types composites

Les deux types de données composites de PL/SQL sont TABLE / VARRAY et RECORD.

|  |  |
| --- | --- |
| **Type composite** | **Explication** |
| TABLE | Définir un tableau de taille variable contenant des objets de même type simple ou complexe. L’index du tableau est de type numérique ou alphanumérique. TYPE /\*nom-Typr\*/ IS TABLE OF /\*type d’éléments\*/ **[NOT NULL]** INDEX BY  /\*index\_by\_type\*/;  index\_by\_type prend la valeur dans BINARY\_INTEGER, PLS\_INTEGER, VARCHAR2(taille),  LONG  Exemple : Définition d’un sous-type tableaux de nombre de taille 22 caractères. TYPE TabTyp1 IS TABLE OF NUMBER(22) INDEX BY BINARY\_INTEGER; t TabTyp1; |
| VARRAY | Définir un tableau de taille maximum n éléments contenant des objets de même type simple ou complexe.  TYPE /\*nom-Typr\*/ IS VARRY(/\*taille\_entier\*/) OF /\*type d’éléments\*/;    Exemple : Définition d’un sous-type tableaux de 200 chaines de taille 22 caractères. TYPE TabTyp IS VARRAY(200) OF VARCHAR2 (22); i TabTyp; |
| Utilisation d’un tableau | Declare  Type TYPE\_TAB\_JOURS IS TABLE OF PLS\_INTEGER INDEX BY VARCHAR2(20) ; jour\_tab TYPE\_TAB\_JOURS ;  Begin jour\_tab( 'LUNDI' ) := 10 ;  jour\_tab( 'MARDI' ) := 20 ; jour\_tab( 'MERCREDI' ) := 30 ; End ;   * EXISTS(i), COUNT, LIMIT : élément i existe ?; nombre d’éléments y compris les valeurs NULL suite à des suppressions ; taille si type TABLE * FISRT, LAST, PRIOR(i), NEXT(i) : accès au premier, dernier élément saisis du tableau ou le précédent ou le suivant de la position i sans compté les éléments supprimés ou non initialisés. PRIOR(FIRST) et NEXT(LAST) donne NULL. * EXTEND : ajouter 1 élément nouveau vide d à la fin du tableau.   EXTEND (n[, m]) : étendre n éléments nouveaux vide ou avec la valeur de l’élément m.  Exemple : t := TabTyp() ; t.EXTEND (10) ;  Dans cet exemple, t.EXTEND(10); permet d’étendre la taille réelle d’un tableau de 10 éléments vides de t(1)au t(10). Ces éléments sont directement accessibles par son index. Si l’élément n’est pas saisi, l’exception NO\_DATA\_FOUND est levée.   * TRIM : supprimer n derniers éléments du tableau * DELETE(n[,m]), DELETE: supprimer les éléments entre n et m ; ou tous les éléments |
| RECORD | Le type de données RECORD permet de spécifier une structure des champs définis par l'utilisateur et des types de données pour ces champs.  TYPE /\*nom-Typr\*/ RECORD (  /\*nom\_champ1 Type1 [:=valeur]\*/  [/\*, nom\_champ2 Type2 [:=valeur]\*/] … [/\*, nom\_champn Typen [:=valeur]\*/]) ; Exemple :  TYPE etudiantType RECORD (nom Varchar2(25), prenom Varchr2(25)) ; un\_Etudiant etudiantTyp ;  un\_etudiant.nom := ‘Dupont’ ; un\_etudiant.prenom := ‘Jean’ ; |



**Exemple : Manipulation de tableau**



SET SERVEROUTPUT ON

Declare

TYPE TYP\_TAB is table of varchar2(100) ;

tab TYP\_TAB ;

Begin

tab := TYP\_TAB( 'lundi','mardi','mercredi' ) ;

tab.EXTEND(4) ;

tab(4) := 'jeudi' ;

tab(5) := 'vendredi' ;

tab(6) := 'samedi' ;

tab(7) := 'dimanche' ;

Ent;

### 3.7. Commandes SQL

Les commandes SQL d’accès aux données (SELECT), de mises à jours des tables (INSERT, DELETE, UPDATE), de gestion de la transaction en cours (COMMIT, ROLLBACK) peuvent être utilisées directement dans un bloc PLSQL.

|  |
| --- |
| **Modèles** |
| **SELECT** /\*liste de colonnes \*/  [**INTO** /\*liste des variables locales\*/ ] **FROM** /\*liste des tables\*/  [**WHERE** /\*condition\*/]  [**GROUP BY** /\*liste des attributs\*/]  [**HAVING** /\*Condition sur groupes\*/]  [**ORDER BY** /\*expression | position\*/ [ASC | DESC] [/\*expression | position \*/[ASC | DESC]]] ; |
| **INSERT INTO** /\*nom table\*/ ] **INSERT INTO (** /\*liste de valeurs\*/ **);**  **DELETE FROM** /\*nom table\*/ [ **WHERE** /\*condition\*/]**;**  **UPDATE** /\*nom table\*/ [**SET**/\*liste attribute = expression \*/ [**WHERE** /\*condition\*/]**;** |
| **COMMIT;**  **ROLLBACK;** |

### 3.8. Types scalaires de caractères

|  |  |
| --- | --- |
| **Type** | **Propriété** |
| Scalaires de caractères  CHAR (n)  VARCHAR (n)  LONG  RAW  LONG RAW  NCHAR(n)  NVARCHAR2(n)  ROWID  UROWID | n : longueur maximale  RAW et LONG RAW peuvent contenir des valeurs binaires (non interprétées par PLSQL)    NCHAR, NVARCHAR sont des chaines codes (un caractère par 1 ou plusieurs octets)  ROWID, UROWID représentent l’adresse absolue d’une ligne (tuple) |

|  |
| --- |
| **Fonctions natives de traitement des chaines de caractères** |
| Concaténation de deux chaines : CONCAT( chaîne1, chaîne2 ) |
| Remplissage à gauche ou à droite d’une chaine : LPAD( chaîne, longueur [, 'caractères'] )  RPAD( chaîne, longueur [, 'caractères'] )  formater une chaîne sur une longueur donnée par ajout de caractères avant (LPAD) ou après (RPAD) la chaîne passée en argument  Exemple  RPAD( LPAD( '2320 euros', 20, '\*' ), 30, '\*' )  \*\*\*\*\*\*\*\*\*\*2320 euros\*\*\*\*\*\*\*\*\*\* |
| Suppression des caractères à gauche ou à droite d’une chaine : LTRIM( chaîne [, 'caractères'] )  RTRIM( chaîne [, 'caractères'] ) - si caractères n'est pas spécifié, le caractère par défaut est l'espace  Exemple :  LTRIM( RTRIM( '"Libellé"', '"' ), '"' ) Libellé |
| Modification de la casse d’une chaine LOWER( chaîne )  NLS\_LOWER( chaîne [, nls\_paramètre ] ) UPPER( chaîne )  NLS\_UPPER( chaîne [, nls\_paramètre ] ) INITCAP( chaîne )  NLS\_INITCAP( chaîne [, nls\_paramètre ] )  Exemple : Conversion d'une chaîne avec la première lettre de chaque mot en majuscule INITCAP( 'le sgbd oracle' ) Le Sgbd Oracle |
| Longueur de la chaine caractères : LENGTH( chaîne ) Longueur de la chaine en octets : LENGTHB( chaîne )  - Si chaque caractère est codé sur un octet on obtient la même valeur. |
| Caractère correspondant au code dans le jeu de caractère utilisé. CHR(code [USING NCHAR CS]) |

|  |
| --- |
| **Fonctions natives de traitement des chaines de caractères (suite)** |
| Extraction d'une sous-chaîne à partir d'une chaîne SUBSTR( chaîne, début [, 'nombre'] ) **chaîne** représente le nom d'une colonne, d'une variable ou un littéral **début** représente la position de départ de recherche dans la chaîne **nombre** représente le nombre de caractères à extraire. Si **nombre** est omis, la fonction ramène tous les caractères à partir de la position début. si **nombre** est négatif l'extraction débute à partir de la fin de la chaîne Exemple : extraction de 4 caractères à partir du 4ème caractère  SUBSTR( 'le sgbd oracle', 4, 4 )  Sgbd |
| Recherche de la position d'une sous-chaîne dans une chaîne  INSTR( chaîne, sous-chaîne [, début [,nombre occurrences] ] ) INSTRB( chaîne, sous-chaîne [, début [,nombre occurrences] ] )  **chaîne** représente le nom d'une colonne, d'une variable ou un littéral passé en argument **sous-chaîne** représente le nom d'une variable ou un littéral dont on cherche la position **début** (optionnel) représente la position de départ de la recherche dans chaîne **nombre occurrences** (si début renseigné) représente le nombre d'occurrences trouvées à ignorer  Lorsque **sous-chaîne** représente plusieurs caractères, la fonction retourne la position du premier caractère de la sous-chaîne  Exemple : recherche de la deuxième position de la sous-chaîne oracle  INSTR( 'le sgbd oracle d''oracle corporation', 'oracle', 1, 2 ) 18 |
| Remplacement des caractères d'une chaîne par une sous-chaîne: REPLACE( chaîne, chaîne source, chaîne cible ) **chaîne** représente le nom d'une colonne, d'une variable ou un littéral passé en argument **chaîne source** représente le nom d'une variable ou un littéral de recherche **chaîne cible** représente le nom d'une variable ou un littéral de remplacement  Exemple : élimination de caractères REPLACE( '"champs1","champs2"', '"', '' )  champs1,champs2 |
| Remplacement caractère par caractère dans une chaîne : TRANSLATE( chaîne, chaîne source, chaîne cible ) **chaîne** représente le nom d'une colonne, d'une variable ou un littéral passé en argument **chaîne source** représente le nom d'une variable ou un littéral de recherche **chaîne cible** représente le nom d'une variable ou un littéral de remplacement  Chaque caractère présent dans **chaîne** qui est également présent dans **chaîne source** est remplacé par le caractère qui occupe la même position dans **chaîne cible** Exemple : élimination de caractères indésirables  TRANSLATE( 'Nom+de|fichier!unix', 'A+|!', 'A' )  Nomdefichierunix |

|  |
| --- |
| **Fonctions natives de traitement des chaines de caractères (suite)** |
| Conversion en chaine de caractères : TO\_CHAR(nombre[format[nls param]]) nls param est une chaîne de caractères formée à partir des expressions suivantes :   * NLS NUMERIC CHARACTERS = ”dg” o LS CURRENCY = ”symbole monétaire local” o NLS ISO CURRENCY = territoire   format est une chaîne de caractères formée des éléments suivants : o un chiffre quelconque   * un chiffre ou un z´ero si absence de chiffres * symbole mon´etaire américain o remplace le nombre 0 par des blancs o signe du nombre post-fixé s’il est négatif, un blanc post-fixé sinon. * signe du nombre o nombre mis entre ”<” et ”>” s’il est négatif, nombre mis entre 2 blancs sinon. o caractère qui sépare la partie entière de la partie fractionnaire o symbole de séparation de groupes de chiffres o symbole monétaire international o symbole monétaire local  * . , caractères affichés tels quels o multiplie le nombre par 10n où n est le nombre de 9 après V. * écriture scientifique du nombre * RM rm écriture en chiffres (entre 0 et 3999)   écriture scientifique du nombre |

o

### 3.9. Types scalaires numériques

|  |  |
| --- | --- |
| **Type** | **Propriété** |
| Scalaires numériques  NUMBER(e,d)  BINARY\_INTEGER  BINARY\_FLOAT  BINARY\_DOUBLE  PLS\_INTEGER  Etc. | e : largueur totale en nombre de caractères stockés et d : nombre de chiffres décimaux après le ‘,’            PLS\_INTEGER est plus rapide que BINARY\_INTEGER car il utilise les registres du processeur |

|  |
| --- |
| Fonctions **natives de traitement des nombres** |
| Valeur absolue d'un nombre : ABS( valeur ) |
| Entier supérieur ou égal à **valeur** : CEIL( valeur ) |
| Entier inférieur ou égal à **valeur** : FLOOR( valeur ) |
| Reste d’une division à **valeur** : MOD( valeur, diviseur )   * Lorsque **diviseur** est supérieur à **valeur**, la fonction retourne la valeur * Si **valeur** est un entier alors MOD( valeur, 1 ) = 0 (idéal pour tester que valeur est entier) - Si **valeur** est un entier pair alors MOD( valeur, 2 ) = 0   Exemple :  MOD( 10, 3 ) = 1 MOD( -10, 2 ) = 0  MOD( 10, .6 ) = 0.4 MOD( -10, .6 ) = -0.4 |
| Elévation d'un nombre à une puissance : POWER( valeur ) |
| e (2,71828182845905) élevé à une puissance : EXP( valeur, exposant ) |
| Racine carrée d’un nombre : SQRT(valeur) |

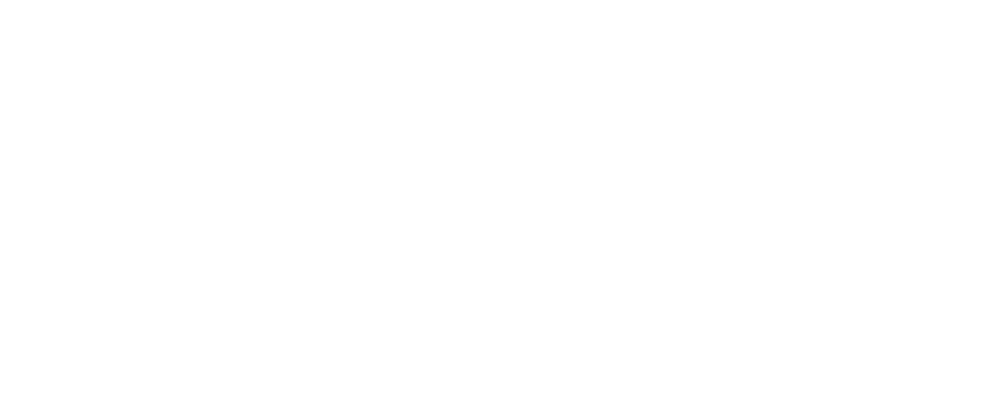
|  |
| --- |
| **Fonctions natives de traitement des nombres (suite)** |
| Logarithme naturel, ou base e, d'une valeur : LN(valeur) |
| Logarithme base ‘base’ d'une valeur : LOG (base, valeur) |
| Arrondi d'une valeur à un certain nombre de chiffres de précision : ROUND(valeur, précision )  Exemple  ROUND( 9.259, 2 ) ROUND( 9.258, 0 )  9,26 9 |
| Suppression d'une partie des chiffres de précision décimale : TRUNC( valeur, précision )  - une précision négative est intéressante pour arrondir des nombres sur des tranches de milliers, millions, milliards, etc.  Exemple :  TRUNC( 9.9, 0 ) TRUNC( 99.259, -1 )  9 90 |
| Signe d'une valeur : SIGN( valeur ) |
| Calculer la valeur trigonométrique d'un angle exprimé en radians :SIN( valeur ), COS( valeur ), TAN( valeur ) |
| Calculer la valeur trigonométrique hyperbolique d'un angle exprimée en radians:  SINH( valeur ), COSH( valeur ), TANH( valeur ) |
| Calculer respectivement l'arc sinus, cosinus et tangente en radians: ASIN( valeur ), ACOS( valeur ), ATAN( valeur ) |
| **Fonctions d'agrégation**  Ces fonctions portent sur des groupes de lignes et sont utilisées dans les ordres select  (Bien sûr rien n'empêche de coder : AVG(40), mais cela n'a pas beaucoup de sens)   * Valeur moyenne des valeurs de **colonne :** AVG( colonne )   (les valeurs NULL ne sont pas prises en compte)   * Nombre de valeurs de **colonne :** COUNT( colonne )   (les valeurs NULL ne sont pas prises en compte)   * Valeur minimum des valeurs de **colonne :** MIN( colonne )   Valeur maximum des valeurs de **colonne :** MAX( colonne ) - Somme des valeurs de **colonne :** SUM( colonne )    Exemple: Select SUM( SAL ) From EMP ; |
| **Fonctions de listes**  Ces fonctions portent sur un ensemble de colonnes d'une seule ligne. Elles peuvent être utilisées avec de nombreuses valeurs qui peuvent apparaître sous forme de colonnes, de littéraux, de calculs ou de combinaison d'autres colonnes.  Valeur la plus grande de la liste : GREATEST( valeur1, valeur2, valeur3,… ) Valeur la plus petite de la liste : LEAST( valeur1, valeur2, valeur3,… )  - Exemple :  GREATEST( 'Elmer', 'Achille','Richard', 'Josianne' )  Richard  LEAST( 3, 5+5, 8, 12-6 )  3 |
| **Fonctions de conversion de types**   * Chaine à nombre TO\_NUMBER(chaîne) * Chaîne hexa en chaine d’octets HEXTORAW(chaîne hexa) * Chaine à identification d’objet ROWID CHARTOROWID(chaîne) |

**3.10. Types date, temps, intervalle de temps**

|  |  |
| --- | --- |
| **Type** | **Propriété** |
| DATE | DATE : Date (du siècle à la seconde)  . Min : 01/01/-4712 (avant J.C) . Max : 31/12/9999 |
| INTERVAL YEAR | INTERVAL YEAR (year\_precision) TO MONTH    Valeur d’une période de temps en années et des mois, où  - year\_precision est le nombre de chiffres dans le champ datetime.YEAR. Les valeurs acceptées vont de 0 à 9. La valeur par défaut est 2 (99 années) |
| INTERVAL DAY | INTERVAL DAY (day\_precision) TO SECOND (fractional\_seconds\_precision)    Valeur d’une période de temps en jours, heures, minutes et secondes, où   * day\_precision est le nombre maximal de chiffres dans le domaine JOUR datetime. Valeurs acceptées vont de 0 à 9. La valeur par défaut est 2 (99 jours). * fractional\_seconds\_precision est le nombre de chiffres dans la partie fractionnaire de la deuxième champ. Valeurs acceptées vont de 0 à 9. La valeur par défaut est 6 (999999 secondes). |
| TIMESTAMP. | TIMESTAMP (fractional\_seconds\_precision)  Année, mois, jour et valeurs de la date, ainsi que l'heure, minute et seconde valeurs de temps, où fractional\_seconds\_precision est le nombre de chiffres dans la partie fractionnaire du champ datetime. seconde. Valeurs de fractional\_seconds\_precision acceptés sont 0 à 9. La valeur par défaut est 6.  TIMESTAMP (fractional\_seconds\_precision) WITH LOCAL TIME ZONE  La valeur de de temps dans le fuseau horaire locale, avec les exceptions suivantes:   * Les données sont normalisées pour le fuseau horaire du serveur quand il est stocké dans la base de données. * Lorsque les données sont récupérées, les utilisateurs voient les données dans le fuseau horaire de la session. |



**Fonctions natives de traitement des dates et temps**



Extraction d’une composante de date :

EXTRACT

**datetemps FROM valeur**

**(**

**)**

datetemps

prend valeur dans :

YEAR

MONTH

DAY

HOUR

MINUTE

SECOND

TIMEZONE\_HOUR

TIMEZONE\_MINUTE

TIMEZONE\_REGION

TIMEZONE\_ABBR

valeur

est l'un des arguments suivants

un littéral de type DATE

un littéral de type INTERVAL

|  |
| --- |
| **Fonctions natives de traitement des dates et temps (suite)** |
| Date courante : SYSDATE |
| La dernière date du mois contenant la date passée en argument. : LAST\_DAY (une\_date)  Dernière date du mois encours ?  SELECT LAST\_DAY(SYSDATE) DernierDateDuMois FROM Dual ; |
| La première date postérieure à un\_date et correspondant au jour de la semaine, passé en argument : NEXT\_DATE (une\_date, ‘UN-JOUR-DE-SEMAINE’)  Date du prochain mercredi ?  SELECT NEXT\_DAY (SYSDATE,'MERCREDI') ProchainMercredi FROM Dual ; |
| La date passée en argument augmentée d’un nombre de mois : ADD\_MONTHS (date, nbre mois) |
| Le nombre de mois qui séparent les 2 dates: MONTHS\_BETWEEN (date1, date2)  - Avec éventuellement une partie fractionnaire correspondant à une partie de 31 jours.  SELECT months\_between('18/03/2014','01/02/2014') Nombre\_mois FROM DUAL; |
| Troncature et arrondi d’une date :  - retourne une date tronquée selon le format spécifié TRUNC (date[format]) - retourne une date arrondie selon le format spécifié ROUND (date[format]) - Format :   * SCC, CC siècle (arrondi au milieu du siècle) o SYEAR, YEAR année (arrondi au 1er juillet) * SYYYY, YYYY année sur 4 chiffres avec et sans signe o YYY année sur les 3 derniers chiffres o YY année sur les 2 derniers chiffres o Y année sur les 1 derniers chiffres * IYYY, IYY, IY, I année ISO (arrondi au 1er juillet) o Q trimestre (arrondi au 16ème jour du 2ème mois du trimestre) o MONTH, MON, MM, RM mois (arrondi au 16ème jour) o WW semaine (7j) définie à partir du 1er janvier (arrondi au 5ème j) o IW semaine ISO (arrondi au vendredi) * DAY, DY, D premier jour de la semaine, dépend du pays : lundi/France, dimanche/US, (arrondi au 5ème jour) * DDD, DD, J jour, par défaut en absent de format (arrondi à la demi-journée) o HH, HH12, HH24 heure (arrondi à la demi-heure) o MI minute (arrondi à 30 secondes)     Exemple : Le premier jour de la première semaine de l’année ?  SELECT  To\_char(trunc(to\_date(’20/02/2014’),’WW’),’FM Day DD’),  To\_char(trunc(to\_date(‘20/02/2014’),’IW’),’FM Day DD’)  FROM Dual ; |

|  |
| --- |
| **Fonctions natives de traitement des dates et temps (suite)** |
| Conversion d’une chaine de caractères en date : TO DATE (chaîne[format[nls \_langue]])   * format est une chaîne de caractères constituée de mots clés o SCC, CC siècle avec et sans signe. o SYEAR, YEAR année en lettres avec et sans signe.   + SYYYY, YYYY année sur 4 chiffres avec et sans signe. o Y,YYY année sur 4 chiffres avec virgule.   + RRRR année sur 4 chiffres ou 2 chiffres avec correction année 2000. o YYY année sur les 3 derniers chiffres. o YY année sur 2 derniers chiffres. o RR année sur 2 chiffres avec correction année 2000.   + Y dernier chiffre de l’année. o BC, AD indication BC (Before Christ) ou AD (Ano Domini).   + Q n° du trimestre de l’année (1-4). o MONTH mois en toutes lettres sur 9 caractères. o MON abréviation du mois sur 3 lettres.   + MM n° du mois dans l’année. o WW n° de la semaine dans l’année (1-53). o IW n° de la semaine ISO dans l’ann´ee (1-52 ou 1-53). o W n° de la semaine dans le mois. o DAY jour en toutes lettres sur 9 caractères. o DY abréviation du jour (2 ou 3 lettres). o DDD n° du jour dans l’ann´ee. o DD n° du jour dans le mois. o D n° du jour dans la semaine. o J jour du calendrier Julien. o AM, PM indication AM ou PM. o HH, HH12 heure sur 12 heures. o HH24 heure sur 24 heures. o MI minutes. o SS secondes par minute (0-59). o SSSSS secondes par jour (0-86399). o ”chaîne” la chaîne est reproduite telle quelle * nls\_langue est une chaîne de la forme : ’NLS\_DATE\_LANGUAGE = langue’ où langue : french, american, arabic, german ... (46 langues supportées) - Les caractères − / , . ; : sont reproduits tels quels. * L’utilisation des majuscules et des minuscules précise le format de sortie.   exemple : ’MONTH’ donnera ’JUIN’ alors que ’Month’ donnera ’Juin’.   * suffixes : o TH ajout du suffixe ordinal o SP nombre en toutes lettres   + SPTH, THSP nombre en toutes lettres avec ajout du suffixe ordinal * modificateurs : o FM suppression de blancs et de zéros   + FX obligation de respecter exactement le format, les blancs sont significatifs   **Exemples d’utilisation :** - Nombre de jours entre 2 dates.  SELECT to\_date('18/03/2014') - to\_date('01/02/2014') Nombre\_jours FROM DUAL; - Date courante sous différentes formes.  SELECT  To\_char(trunc(to\_date('20/02/2014'),'WW'),'FM Day DD'),  To\_char(trunc(to\_date('20/02/2014'),'IW'),'FM Day DD') FROM Dual ; |