Index

L'index permettra de faire des recherches par dichotomie et d'accélérer les tris sur le champ concerné.

Un index est une table de correspondance où les données seront classées .

Le langage SQL

Structured Query Language (SQL)

• Langage de requête structurée.

Comme tout langage qui sera traduit pour être exécuté par la machine, il a un **LEXIQUE**, une **SYNTAXE** et une **SÉMANTIQUE** (et possède un métalangage qu'on appelle aussi grammaire du langage).

• **Requête** : instruction demandant une action sur la base de données.

• SQL: Langage standardisé pour **interroger**, **manipuler** et **définir** des données, et pour fournir un contrôle des accès dans les BDDR

• Développé chez IBM fin des années 79 avec la forte participation d'Oracle standard SQL ANSI.

Les ensembles du langage SQL

Il y 16 commandes SQL classées 4 sous ensembles d'utilisation :

• Le Langage de Définition de Données (**DDL**) pour créer et supprimer des objets dans la base (schéma de la BDD) :
■ CREATE ■ ALTER ■ DROP ■ RENAME ■ TRUNCATE ■ COMMENT

• Le Langage de Contrôle de Données (**DCL**) pour gérer les droits sur les objets de la base:
■ GRANT ■ REVOKE

• Le Langage de Manipulation de Données (**DML**) pour la recherche, l'insertion, la mise à jour et la suppression de données ■ SELECT ■ INSERT ■ UPDATE ■ DELETE ■ MERGE

• Le Langage de Contrôle de Transaction (**TCL**) pour la gestion des transactions de la base
■ COMMIT ■ ROLLBACK ■ SAVEPOINT

DDL : L'instruction CREATE

Elle concerne :

■ Tables ■ Views ■ Synonyms ■ Indexes ■ Sequences

Function : SUBSTR (Col, Départ, nbChar)

Permet d'extraire une sous chaine d'une chaine..

A l'aide de cette fonction donnez
le nombre de recrutement par mois et par an

```
SELECT substr(hire_date,7,2) ,
       substr(hire_date,4,2) ,
       count(*)
       from employees
group by substr(hire_date,7,2),
substr(hire_date,4,2);
```

La clause COUNT

Permet de compter le nombre

```
SELECT COUNt(*) FROM employees;
```

```
select count(DEPARTMENT_ID) from employees;
```

```
select count (distinct DEPARTMENT_ID) from employees;
```

```
select distinct substr(hire_date,7,2)
from employees;
```

```
select count (distinct substr(hire_date,7,2))
from employees;
```

Exemple : GROUP BY

La fonction de groupage consiste à faire des calculs sur un critère.
Exemple somme des salaires par département.

```
SELECT DEPARTMENT_ID, SUM(salary)
FROM employees
GROUP BY  DEPARTMENT_ID
ORDER BY DEPARTMENT_ID;
```

Les ALIAS

On peut utiliser des alias pour renommer les colonnes

```
SELECT substr(hire_date,7,2) an,
       substr(hire_date,4,2) mois,
       count(*) total
       from employees
group by substr(hire_date,7,2),
substr(hire_date,4,2)
order by an,mois;
```

Exemple : HAVING

La having est l'équivalent du WHERE pour les champs caculés
Exemple somme des salaires par département.

```
select department_id, sum(salary) total
from employees
group by department_id
having sum(salary) > 50000;
```

Les opérateurs arithmétiques

SQL permet d'utiliser les opérateurs arithmétique classiques

Somme des salaires de tous les employés

```
select sum(SALARY) from employees;
```

Somme de 1% du salaire de chaque employé

```
select sum(SALARY)*1/100 from employees;
```

On souhaite simuler l'augmentation de 4% du salaire de chaque employé

	Nom	Salaire	Majoration	Nouveau Salaire
1	TJ	2100	105	2205
2	Steven	2200	110	2310

Les opérateurs arithmétiques

SQL permet d'utiliser les opérateurs arithmétique classiques

Somme des salaires de tous les employés

```
select sum(SALARY) from employees;
```

Somme de 1% du salaire de chaque employé

```
select sum(SALARY)*1/100 from employees;
```

On souhaite simuler l'augmentation de 4% du salaire de chaque emplové

```
select FIRST_NAME "Nom", salary "Salaire", SALARY*5/1(
salary+SALARY*5/100 "Nouveau Salaire" from employees
order by SALARY asc;
```

Les opérateurs arithmétiques

Sous ORACLE, la clause FROM est obligatoire

```
SELECT 30+20 ;
```

```
SELECT 30+20 FROM DUAL;
```

Autres fonctions sur les dates

ADD_MONTHS(date1, n) : Ajoute n mois a la date **exemple** SELECT ADD_MONTHS(hire_date,3) PROB_DATE,

LAST_DAY(date1) donne le dernier jour du mois de la date **28,29,30,31 exemple** LAST_DAY(hire_date)

NEXT_DAY(date1, char1) donne la date du 1er jour –char1- qui suit date1 **exemple**
NEXT_DAY(ADD_MONTHS(hire_date,2), **'LUNDI'**) PROJ_DATE, 1er lundi 2 mois après recrutement.

MONTHS_BETWEEN(date1, date2) : nombre de mois entre 2 dates
ROUND(date1, {fmt})
SYSDATE donne la date systeme.

Exercice, donnez : PROB_DATE est calculé en ajoutant trois mois à HIREDATE (fin période de test), PROJ_DATE est le premier lundi qui tombe immédiatement après deux mois à compter de la date d'embauche et LAST_DATE est le dernier jour du mois au cours duquel l'employé a été embauché.

Autres fonctions sur les dates

```
SELECT ADD_MONTHS(hire_date,3) PROB_DATE,
NEXT_DAY(ADD_MONTHS(hire_date,2), 'LUNDI')
LAST_DAY(hire_date) LAST_DATE, SYSDATE
FROM employees  WHERE salary >= 3000;
```

PROB_DATE	PROJ_DATE	LAST_DATE	SYSDATE
17/09/03	18/08/03	30/06/03	19/03/20
21/12/05	28/11/05	30/09/05	19/03/20

Autres fonctions de groupage

AVG(x)
MIN(x)
MAX(x)
COUNT({*|x})
SUM(x)
STDDEV(x) S
VARIANCE(x)

Syntaxe :
SELECT [column1],[col2], group_function(column), ……
FROM table_name
[WHERE condition]
[GROUP BY column1,[col2]
[ORDER BY column];

SELECT MAX(salary) maxsal, MIN(salary) minsal, AVG(salary) salavg, COUNT(*) TOTAL FROM; employees

Autres fonctions sur les strings

TO_CHAR(number, fmt) – convertit un nombre donné en type de données de caractères au format spécifié.

```
SELECT employee_id, TO_CHAR(salary, 'L99G999D00') amount  FROM employees;
```

	EMPLOYEE_ID	AMOUNT
1	100	€24 000,00
2	101	€17 000,00
3	102	€17 000,00
4	103	€9 000,00
5	104	€6 000,00

FORMAT:

L : currency
G : séparateur de groupe
D : décimal

Jointures : produit cartésien (CROSS JOIN)

Produit cartésien – pas de comparaison entres champs des tables
Exemple :

```
SELECT employee_id, first_name, department_name
FROM employees e, departments d
```

	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_NAME
1	100	Steven	Administration
2	101	Neena	Administration
3	102	Lex	Administration

Jointures : Équijointure

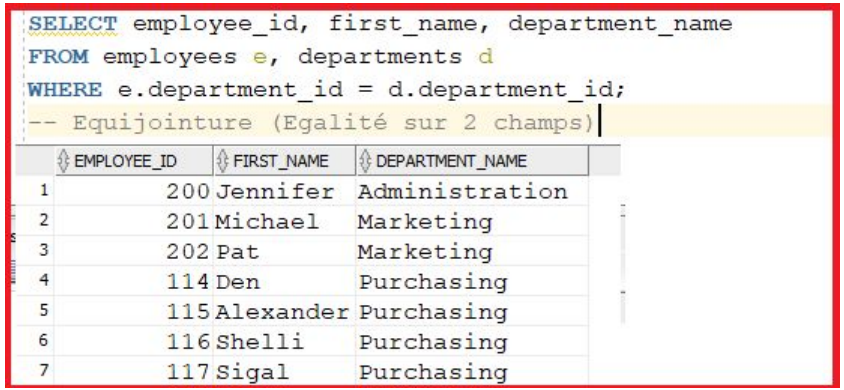
Utilisation des alias sur les noms de table

Equi Jointure – where porte sur l'égalité entre 2 champs
Exemple :
SELECT le nom du salarié et le nom de son département.

Jointures : Équijointure

Utilisation des alias sur les noms de table

```
SELECT employee_id, first_name, department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```



```
SELECT employee_id, first_name, department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
-- Equijointure (Egalité sur 2 champs)
```

	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_NAME
1	200	Jennifer	Administration
2	201	Michael	Marketing
3	202	Pat	Marketing
4	114	Den	Purchasing
5	115	Alexander	Purchasing
6	116	Shelli	Purchasing
7	117	Sigal	Purchasing

Jointures : Jointure naturelle

JOINTURE NATURELLE – la colonne doit avoir le même nom et le même type dans les 2 tables
Exemple :

```
SELECT employee_id, first_name, department_name
FROM employees
Natural join departments
# la jointure se fera sur le champs : department_id
```

```
SELECT employee_id, first_name, department_name
FROM employees
natural join departments
```

	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_NAME
1	101	Neena	Executive
2	102	Lex	Executive
3	104	Bruce	IT
4	105	David	IT
5	106	Valli	IT
6	107	Diana	IT
7	109	Daniel	Finance
8	110	John	Finance

Jointures : Jointure avec mot-clé **USING**

Vous pouvez avoir des situations où plusieurs noms de colonnes correspondent.
Dans de tels cas, NATURAL JOIN utilise toutes les colonnes avec des noms et des types de données correspondants pour établir la jointure.
Au lieu de la JOINTURE NATURELLE, Vous pouvez spécifier les colonnes à utiliser pour créer une condition de jointure à l'aide de la clause USING

Exemple : SELECT e.employee_id, e.first_name, d.department_name
FROM employees e
JOIN departments d
USING (department_id);

```
SELECT e.employee_id, e.first_name, d.department_name
FROM employees e
JOIN departments d
USING (department_id);
```

	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_NAME
1	200	Jennifer	Administration
2	201	Michael	Marketing
3	202	Pat	Marketing
4	114	Den	Purchasing
5	115	Alexander	Purchasing

Jointures : Nonequijoin – non équijointure

condition d'équi jointure implique que les valeurs sont égales dans les deux tableaux.

Vous pouvez avoir une situation où les tables doivent être jointes mais il n'y a pas de relation d'égalité entre les tables.

Dans de tels cas, vous pouvez établir un nonequijoin.

Un nonequijointure est une condition autre que l'égalité.

Le cas typique est une notation par interval de valeur : de 0 à 5 : pas bon, de 6 à 10 médiocre, de 11 à 12 moyen …

Jointures : Nonequijoin – non équijointure

```
create table grade (grade_id number(1), low_sal number(6), max_sal number(6));
select distinct salary from employees order by salary;
insert into grade values (9, 1000, 2000);
insert into grade values (8, 2001, 4000);
insert into grade values (7, 4001, 6000);
insert into grade values (6, 6001, 8000);
insert into grade values (5, 8001, 10000);
insert into grade values (4, 10001, 15000);
insert into grade values (3, 15001, 20000);
insert into grade values (2, 20001, 40000);
insert into grade values (1, 40001, 100000);
```

On souhaite connaître la catégorie de chaque salarié

```
select first_name, salary, grade_id grade
from employees, grade
where salary between low_sal and max_sal;
```

La table Employees et la table Grade fournissent un exemple de nonequijointure.

Les employés de la table Employé doivent avoir leur salaire entre la valeur dans les colonnes lo_sal et hi_sal de la table Grade.

Jointures : Nonequijoin – non équijointure

condition d'équijointure implique que les valeurs sont égales dans les deux tableaux.

Vous pouvez avoir une situation où les tables doivent être jointes mais il n'y a pas de relation d'égalité entre les tables.

Dans de tels cas, vous pouvez établir un nonequijoin.

Un nonequijointure est une condition autre que l'égalité.

Exemple : La table Employees et la table Grade fournissent un exemple de nonequijointure.

Les employés de la table Employé doivent avoir leur salaire entre la valeur dans les colonnes lo_sal et hi_sal de la table Grade.

les **jointures externes** :

Jointures : qu'en est –il des lignes qui ne satisfont pas la condition ?

 Si on veut aussi les récupérer on utilise les **jointures externes** : outer join

Les instructions equijoin récupèrent uniquement les enregistrements avec correspondants dans les 2 tables.

Vous pouvez récupérer les enregistrements sans correspondance à l'aide des jointures externes.

Une jointure entre deux tables qui renvoie les lignes correspondantes ainsi que les lignes sans correspondance est appelée une jointure externe

Il existe deux types de jointures externes, **RIGHT OUTER JOIN** et **LEFT OUTER JOIN**

EMPLOYES		
107	Diana	60
108	Nancy	100
109	Daniel	70
110	John	110
111	Ismael	170
112	Jose Manuel	170
113	Luis	60
114	Den	
115	Alexander	
116	Shelli	
117	Sigal	
118	Guy	

DEPARTMENTS	
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive
100	Finance
110	Accounting
120	Treasury
130	Corporate Tax
140	Control And Credit
150	Shareholder Services
160	Benefits
170	Manufacturing
180	Construction

EMPLOYES		DEPARTMENTS	
107	Diana	60	IT
108	Nancy	100	Finance
109	Daniel	70	Public Relations
110	John	110	Accounting
111	Ismael	170	Accounting
112	Jose Manuel	170	Accounting
113	Luis	60	IT
114	Den		
115	Alexander		
116	Shelli		
117	Sigal		
118	Guy		
		10	Administration
		20	Marketing
		30	Purchasing
		40	Human Resources
		50	Shipping
		120	Treasury
		130	Corporate Tax
		140	Control And Credit
		150	Shareholder Services
		160	Benefits

Jointures : LEFT OUTER JOIN

En plus de lignes correspondante, le LEFT OUTER JOIN récupère aussi les lignes sans correspondance pour la table à gauche de l'égalité.

Exemple

```
SELECT e.first_name, d.department_name
FROM employees e                                   employees est ici à gauche
LEFT OUTER JOIN departments d ON (e.department_id = d.department_id);
```

```
SELECT e.first_name, d.department_name
FROM employees e
LEFT OUTER JOIN departments d ON (e.department_id = d.department_id);
-- Kimberly qui n'a pas de correspondance apparait
```

	FIRST_NAME	DEPARTMENT_NAME
97	Neena	Executive
98	Lex	Executive
99	Nancy	Finance
100	Daniel	Finance
101	John	Finance
102	Ismael	Finance
103	Jose M...	Finance
104	Luis	Finance
105	Shelley	Accounting
106	William	Accounting
107	Kimberely	(null)

Jointures : RIGHT OUTER JOIN

En plus de lignes correspondante, le LEFT OUTER JOIN récupère aussi les lignes sans correspondance pour la table à droite de l'égalité.

Exemple

SELECT e.first_name, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d ON (e.department_id = d.department_id); □ department est ici à droite

```
SELECT e.first_name, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d ON (e.department_id = d.department_id);
-- Les départements sans affectation apparaissent
```

	FIRST_NAME	DEPARTMENT_NAME
105	William	Accounting
106	Shelley	Accounting
107	(null)	Treasury
108	(null)	Corporate Tax
109	(null)	Control And ...
110	(null)	Shareholder ...
111	(null)	Benefits
112	(null)	Manufacturing
113	(null)	Construction
114	(null)	Contracting
115	(null)	Operations
116	(null)	IT Support
117	(null)	NOC

Jointures : FULL OUTER JOIN

En plus de lignes correspondante, le FULL OUTER JOIN récupère aussi les lignes sans correspondance **pour les deux tables.**

Exemple

```
SELECT e.first_name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON (e.department_id = d.department_id);
```

118	Joshua	Shipping
119	Trenna	Shipping
120	Curtis	Shipping
121	Randall	Shipping
122	(null)	Treasury
123	Kimberely	(null)

PL/SQL

PL/SQL est un langage procédural (déclaration de variable, structure de contrôle -if then else- structure itératives …) **conçu pour accepter les instructions SQL au sein de sa syntaxe**.

Les unités du programme PL/SQL sont compilées par le serveur Oracle Database et peuvent stockées dans la base de données.

Au moment de l'exécution, les deux langages PL/SQL et SQL s'exécutent au sein du même processus de serveur, pour une efficacité optimale.

```
SET SERVEROUTPUT ON;   -- activer l'affichage

        /* Structure d'un block PL/SQL
un bloc PL/SQL peut être constitué de 3 sections : */
```

1- Section déclare (en commentaire car non obligatoire)

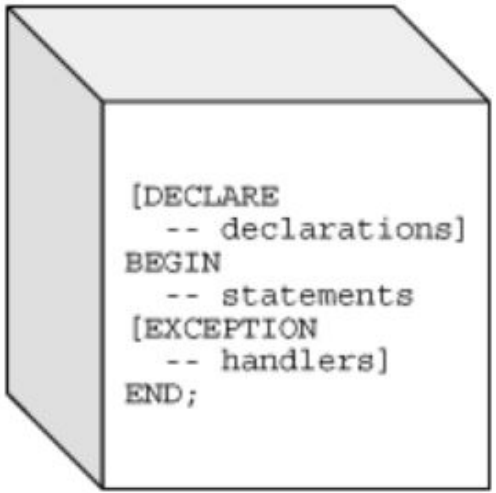
```
-- DECLARE
    -- déclarations et initialisation -- facultative
```

2- Section BEGIN qui contient la section **EXCEPTION**

```
BEGIN
    -- instructions exécutables  -- obligatoire
    DBMS_OUTPUT.PUT_LINE('hello');
-- EXCEPTION
--  interception des erreurs -- facultative
END;
```

ce block est appelé block anonyme car non stocké dans la base sous forme de procédure ou de fonction.

Figure 1-2 Block Structure




```
DBMS_OUTPUT.PUT_LINE(argument);
```
Pour afficher un résultat

`:=` pour valoriser une variable

Block imbriqué sous PL/SQL

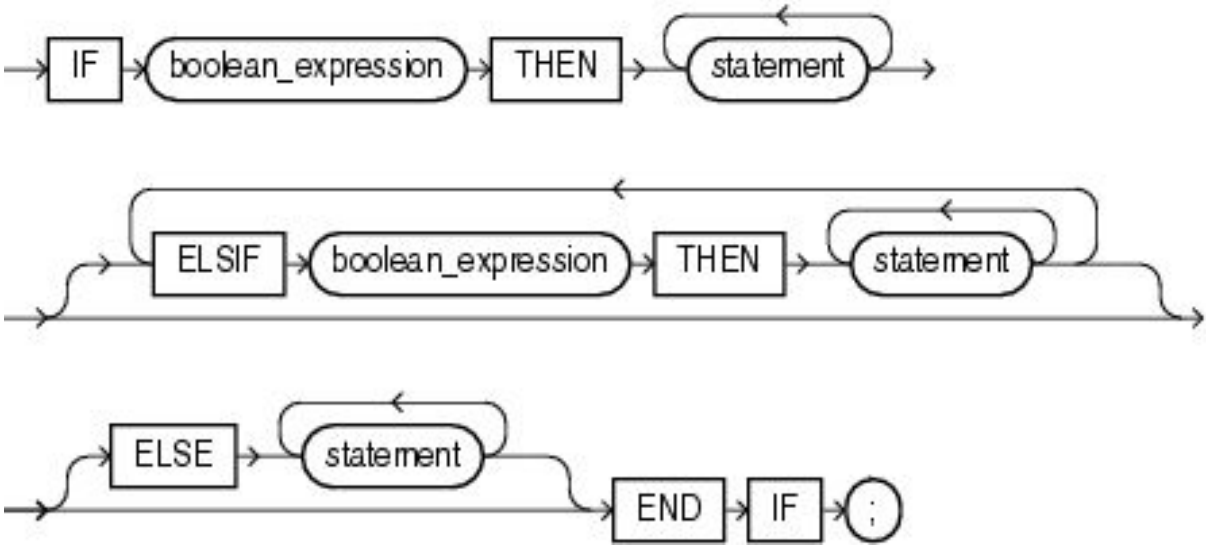
```
DECLARE
    val1  int ; val2    int ;
BEGIN
val1 := 15; val2 := 10;

 declare
  resultat int :=val1+val2;
  begin
    DBMS_OUTPUT.put_line ( 'la somme de '||val1 || ' et '||val2 ||' est : ' ||resultat);
  end;

EXCEPTION
   WHEN OTHERS
  THEN
    DBMS_OUTPUT.put_line (SQLERRM);
END;
```

Section exception du bloc PL/SQL

```
set serveroutput  on;
DECLARE
  l_message   varchar2(2) ;
BEGIN
 l_message := 'hllo';
   DBMS_OUTPUT.put_line (l_message);

EXCEPTION
  WHEN OTHERS
  THEN
    DBMS_OUTPUT.put_line  ('erreur '||DBMS_UTILITY.format_error_stack||' ---  '||sqlerrm);
END;
```

Référence : DOCUMENTATION OFFICIELLE ORACLE

https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/if_statement.htm



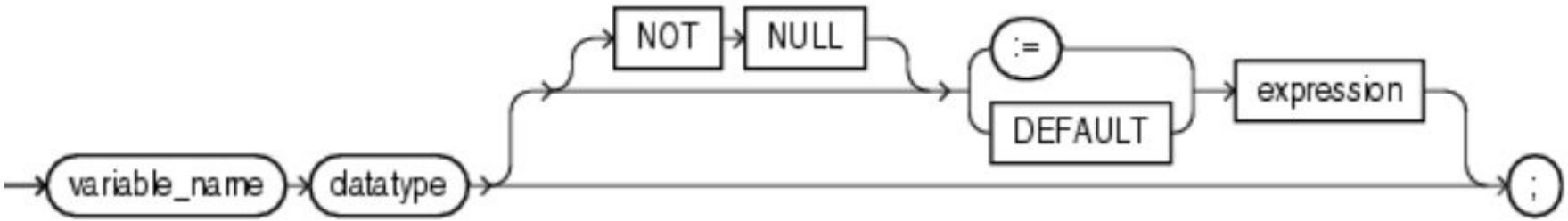
Déclaration de variable sous PL/SQL

https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/constantvar_declaration.htm

Déclaration de variable sous PL/SQL

```
Déclare pour déclarer des variables;
:= pour valoriser une variable

Begin
          -- liste des instructions
End;
```

variable declaration ::=



https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/constantvar_declaration.htm

Utilisation de variables sous PL/SQL

```
DECLARE
    val1  int ; val2    int ;
BEGIN
val1 := 15; val2 := 10;
  DBMS_OUTPUT.put_line ( 'la somme de '||val1 || ' et '||val2 ||' est : ' ||(val1+val2));
EXCEPTION
   WHEN OTHERS
  THEN
    DBMS_OUTPUT.put_line (SQLERRM);
```

https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/constantvar_declaration.htm

Utilisation de variables sous PL/SQL

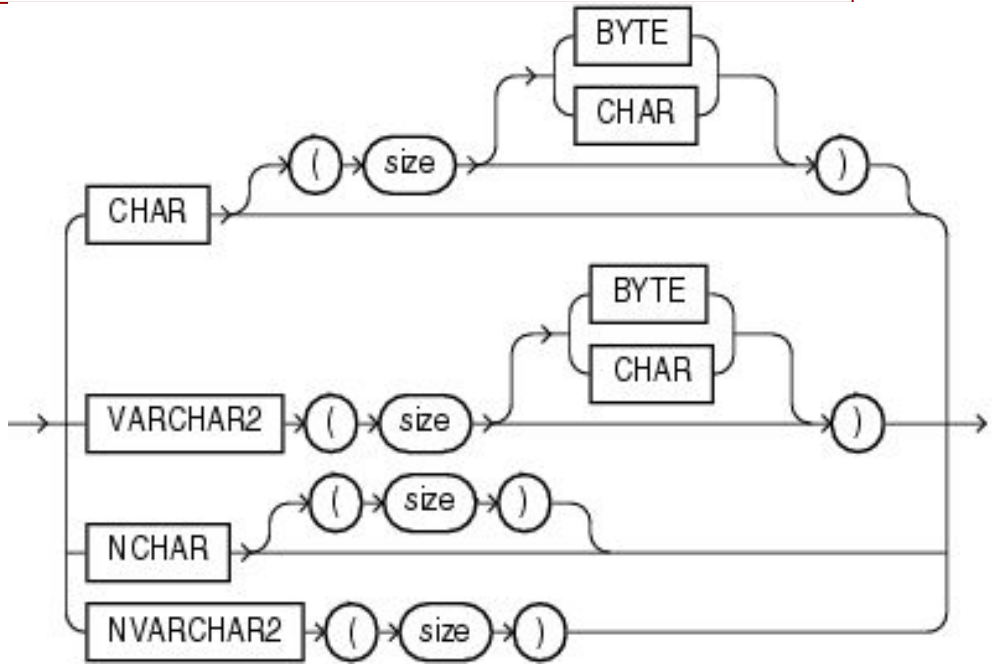
```
DECLARE
   wages           NUMBER;
   hours_worked    NUMBER := 40;
   hourly_salary   NUMBER := 22.50;
   bonus           NUMBER := 150;
   country         VARCHAR2(128);
   counter         NUMBER := 0;
   done            BOOLEAN;
   valid_id        BOOLEAN;
   emp_rec1        employees%ROWTYPE;
   emp_rec2        employees%ROWTYPE;
   TYPE commissions IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
   comm_tab        commissions;
BEGIN
   wages := (hours_worked * hourly_salary) + bonus;
   country := 'France';
   country := UPPER('Canada');
   done := (counter > 100);
   valid_id := TRUE;
   emp_rec1.first_name := 'Antonio';
   emp_rec1.last_name := 'Ortiz';
   emp_rec1 := emp_rec2;
   comm_tab(5) := 20000 * 0.15;
END;
```

Les types de données sous Oracle

- les types de données sous oracle
- **char** vs **varchar2**
- les nombres sous oracle
- a la découverte du type **number(p,s)**
- constantes
- type de données **booléen**
- les types **datetime**
- quelques fonctions de conversions : la fonction **to_char**() - la fonction **to_date**() la fonction extract() …

Les types de données sous Oracle CHAR, VARCHAR2

Types	Description	Size
VARCHAR2(n)	Variable-length character string.	From 1 byte to 4KB.
NVARCHAR2(size)	Variable-length Unicode character string having maximum length size characters.	Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2.
Char	Fixed length	



{ **CHAR** [(size [**BYTE** | **CHAR**])]
| **VARCHAR2** (size [**BYTE** | **CHAR**])
| **NCHAR** [(size)]
| **NVARCHAR2** (size)
}

https://docs.oracle.com/cd/B28359_01/server.111/b28286/sql_elements001.htm#SQLRF0021

Char VS Varchar2

Comme l'indique le nom des types :

- Utiliser char lorsque vous avez une longueur fixe (Exemple numéro de tél sur 10)
- Utiliser char , pour optimiser l'espace de stockage

Soit la table **Ora_Type_char** qui contient une colonne dont la **longueur est fixe**

```
CREATE TABLE Ora_Type_char (
    password_Fixe char(15)) ;
```

Et la table **Ora_Type_varchar** qui contient une colonne dont la l**ongueur est variable**

```
CREATE TABLE Ora_Type_char_v (
    col1 varchar2(15)) ;
```

EXERCICE : peuplez les 2 tables avec le même nombre d'enregistrements et comparez leur taille en se servant la requête suivante :

```
select segment_name,segment_type, sum(bytes) Octets
 from USER_SEGMENTS
group by segment_name,segment_type;
```

CHAR VS VARCHAR2

```
declare
i int ;

Begin
    for i in 1 .. 100000
Loop
   insert into Ora_Type_char (tel) select  dbms_random.string('A', 15)  from dual;
end loop;
end;
```

Créer une table char mais nop - avec des longueurs varaiables :

Même espace est réservé.

Créer comme suit une table dont la longueur est variable :

```
declare
i int ;
n_aleatoire int;
Begin
    for i in 1 .. 100000
Loop
  select  dbms_random.value(1, 15) into n_aleatoire  from dual;
   insert into Ora_Type_char_nop (tel) select  dbms_random.string('A', n_aleatoire)  from dual;
end loop;
end;
```

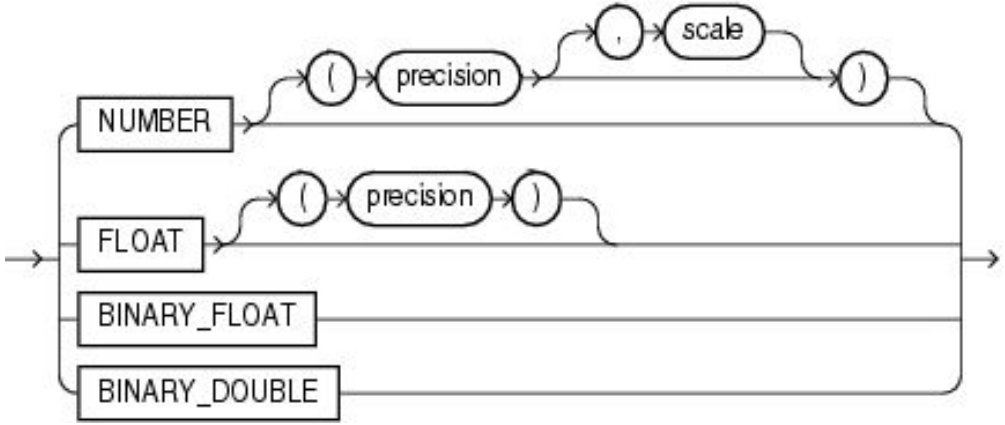
```
CREATE TABLE Ora_Type_char_v2 (  tel varchar2(15)) ;
  insert into Ora_Type_char_v2  select rtrim(tel) from  Ora_Type_char_nop;
```

LES NOMBRES SOUS ORACLE

Table B-2 Mapping ANSI Data Types to Oracle Data Types

ANSI	Oracle
SMALLINT	NUMBER(5)
INTEGER	NUMBER(10)
NUMERIC(p,s)	NUMBER(p,s)
FLOAT	FLOAT(23)
DOUBLE PRECISION	FLOAT(49)
VARCHAR	VARCHAR2
DATE	DATE
TIME	DATE
TIMESTAMP	DATE

LES NOMBRES SOUS ORACLE



x number (6,2) ⟷ p = 6, 1234.56, s = 2

{ NUMBER [(precision [, scale])]
| FLOAT [(precision)]
| BINARY_FLOAT
| BINARY_DOUBLE
}

Un nombre de longueur **p** dont **s** décimaux (au **s**-ième près).

x = 1234.56 ⟶ OK

x = 1234.56 7⟶ OK sera arrondi à 1234.57

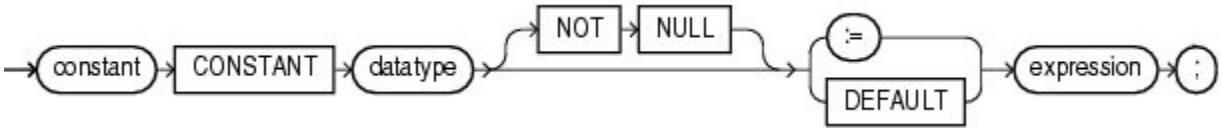
Exercices 1

```
  CREATE TABLE Ora_Type_demo ( number_value NUMERIC(13, 2));
--
insert into ora_type_demo (number_value)  values (12345678901.23); -- s < p
select * from ora_type_demo;

---------
insert into ora_type_demo (number_value)  values (12345678901.23); -- s < p
select * from ora_type_demo;

-------------------------
alter table ora_type_demo add number_ps_neg number(5,-2);
insert into ora_type_demo (number_ps_neg) values (12345);
```

Exercice 2 dirigé : A la découverte du type Number(p,s)

```
DECLARE
x NUMBER(4,2);
BEGIN
 -- x := 23.45;
 -- x := 23.456;
   x := 123.456;
  DBMS_OUTPUT.put_line(' x = '||x);
END;
```

CONSTANTES



Exemples :

```
credit_limit      CONSTANT REAL    := 5000.00;
max_days_in_year CONSTANT INTEGER := 366;
urban_legend      CONSTANT BOOLEAN := FALSE;
hours_worked     INTEGER := 40;
employee_count   INTEGER := 0;
pi      CONSTANT REAL := 3.14159;
```

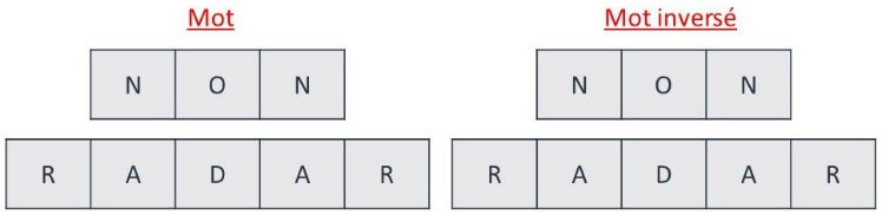
TYPE DE DONNÉES BOOLÉEN

Le type de données PL/SQL BOOLEAN stocke des valeurs logiques, qui sont **TRUE** , **FALSE** ou **NULL**.

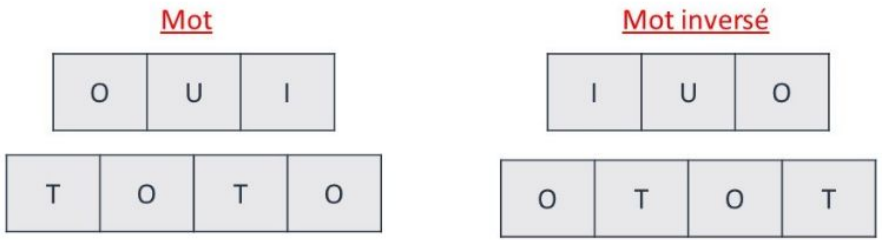
La syntaxe pour déclarer une variable BOOLEAN est :

nom_variable BOOLEEN

Exercice EstPalindrome



Exemple de mots qui ne sont pas des palindromes :



Exercice EstPalindrome, PSEUDO-CODE

```
FONCTION estPalindrome (mot)
VARIABLES
        motInverse : CHAINE DE CARACTERES
DEBUT
        motInverse ← «  »
        POUR i ALLANT DE taille(mot) - 1 A 0 PAR PAS DE -1 FAIRE
                motInverse ← motInverse + mot[i]
        FINPOUR

        POUR i ALLANT DE 0 A taille(mot) - 1 PAR PAS DE 1 FAIRE
                //vérification lettre à lettre
        FINPOUR

FIN
```

https://www.algorithmique-h2prog.com/index.php/2021/03/19/exercice-53-algorithme-palindrome-moyen/

TYPE DE DONNÉES BOOLÉEN

```
declare
c1 varchar2(250);
c1_inverse varchar2(250) := '';
reponse boolean := true;
compteur integer ;
reponse_afficher varchar2(50) := ' est un palindrome ';
begin
       dbms_output.put_line('saisir un mot :');
       c1 := '&c1';
       dbms_output.put_line(length(c1));
       for compteur IN REVERSE 1 .. length(c1)
       loop
           c1_inverse :=  c1_inverse||substr(c1,compteur,1);
             dbms_output.put_line(substr(c1,compteur,1));

       end loop;
       for compteur IN  1 .. length(c1)
       loop
          reponse := (substr(c1,compteur,1) = substr(c1_inverse,compteur,1));
          if (reponse = false) then
          reponse_afficher := ' n''est pas un palindrome';
          end if;
       end loop;

       dbms_output.put_line(c1_inverse ||' --- '|| c1 || reponse_afficher);

end;
```

https://www.algorith[illegible]ne-moyen/

LES TYPES DATETIME

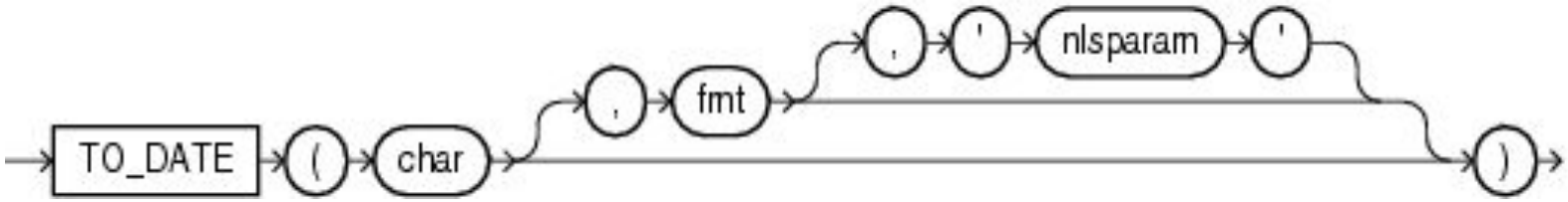
Les type datetimes sont :

- **DATE,**
- **TIMESTAMP,**
- **TIMESTAMP WITH TIME ZONE,**
- et **TIMESTAMP WITH LOCAL TIME ZONE**.

Exemple :
--https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements004.htm#i34924

LA FONCTION TO_DATE()

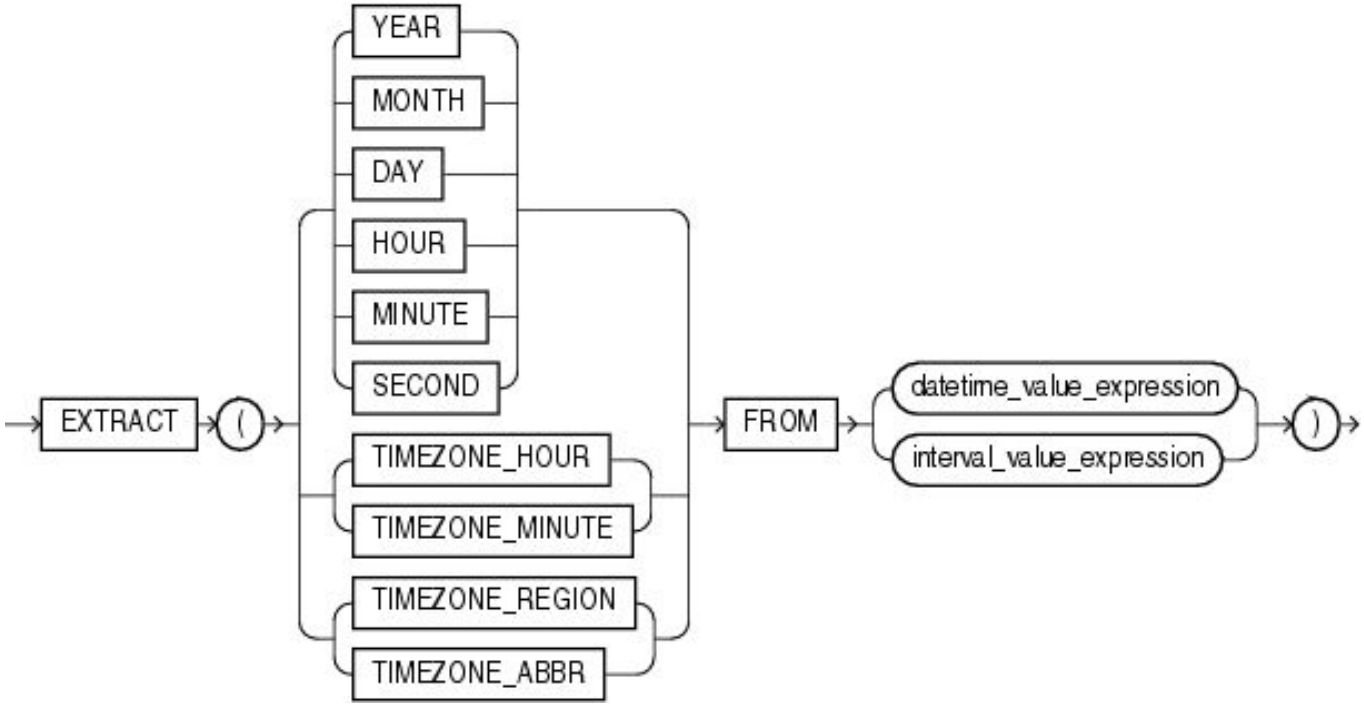
Elle convertie en date, un string fourni en entrée suivant un format spécifique.



TO_DATE(char [, fmt [, 'nlsparam']])

fmt : DD - DDTH - DDSP - DDSPTH - DAY (jours)
MM - MON - MONTH (mois)
YY - YYYY - RR - RRRR - YEAR - (année)
nlsparam : français,

LA FONCTION EXTRACT()



EXTRACT({ { **YEAR** | **MONTH** | **DAY** | **HOUR** | **MINUTE** | **SECOND** }
| { **TIMEZONE_HOUR** | **TIMEZONE_MINUTE** }
| { **TIMEZONE_REGION** | **TIMEZONE_ABBR** } }
FROM { datetime_value_expression | interval_value_expression })

LES TYPES DATETIME

```
DECLARE
d1 DATE := sysdate;
d2 TIMESTAMP with time zone := sysdate;
BEGIN
--dbms_output.put_line (d1);
dbms_output.put_line (to_char(d1, 'DAY, DD MM (MONTH) YYYY HH:Mi:SS "de lan" YEAR  '));
dbms_output.put_line (d2);
dbms_output.put_line (to_char(d2, 'DAY, DD MM (MONTH) YYYY HH:Mi:SS:ff YEAR TZR TZD TZH TZM '));

END;
--ALTER SESSION SET TIME_ZONE = '-07:00';
alter session set  time_zone='00:00';
```

LA FONCTION EXTRACT()

```
declare
d1 date  ;
begin


  -- PROMPT   "saisir la date de rendez-vous"

  d1 := to_date('&d1', 'DD/MM/YYYY');
    dbms_output.put_line('nous sommes :'||to_char(d1,'DAY'));
    dbms_output.put_line('dans 3jours, nous seront :'||to_char((d1+3),'DAY'));
 dbms_output.put_line('l''année :'|| extract(YEAR from d1));

end;
```

EXERCICE

/*

Afficher **SECOND** et MILLIÈME de seconde à partir de CURRENT_TIMESTAMP

*/

LES BLOCS PL SQL : LES VARIABLES et TYPE ATTRIBUTS

```
SET SERVEROUTPUT ON
DECLARE
v_name EMPLOYEES.first_name%TYPE;
v_depno departments.department_name%TYPE;
BEGIN
DBMS_OUTPUT.PUT_LINE(NVL(v_name, 'Sans nom ')||
' Appartient au departerment  '||NVL(v_depno, ': Aucun' ));
END;
```

Exercices

- Ecrire un bloc anonyme pour afficher la somme, le produit de 2 nombres
- Ecrire un bloc anonyme pour calculer le temps qu'il faut à votre pc pour réaliser une addition de 2 nombres
-

Exercices

```
-------------------------------------------------------------------
---------------------  LES VARIABLES et initialisation -----------
-------------------------------------------------------------------
SET SERVEROUTPUT ON
DECLARE
v_valeur1 int := 10;
v_valeur2 int := 20;

BEGIn
DBMS_OUTPUT.PUT_LINE ('La somme est  :'|| (v_valeur1 + v_valeur2) );
END;
```

Exercices : puissance

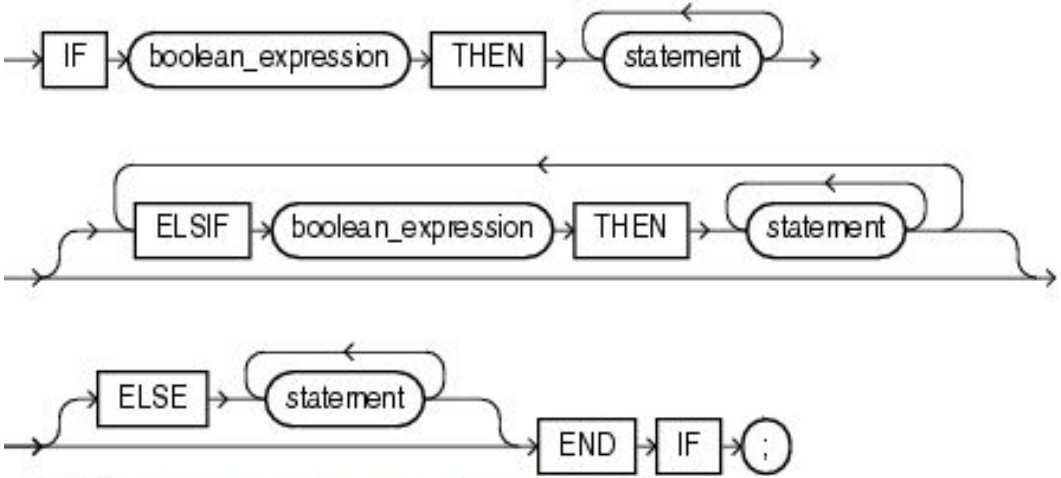
```
SET SERVEROUTPUT ON
DECLARE
v_valeur1 int := 10;
v_valeur2 int := 20;

BEGIn
DBMS_OUTPUT.PUT_LINE ('Le produit est  :'|| power(v_valeur1, v_valeur2) );
END;
```

IF THEN : forme simple

```
IF CONDITION THEN
        STATEMENT 1;
ELSE
        STATEMENT 2;
END IF;
```

```
IF CONDITION THEN
    BEGIN
        STATEMENT 1;
        STATEMENT 2;
     END;
ELSE
        STATEMENT 3;
END IF;
```

STRUCTURE DE CONTRÔLE : IF - THEN - ELSE




```
IF boolean_expression THEN
    statement [statement]...
 [ELSIF boolean_expression THEN
    statement [statement]...]
   [ELSIF boolean_expression THEN
   statement [statement]...]...
 [ELSE statement [statement]...]
END IF;
```

EXERCICE

/*
EXERCICE 1: Demander à l'utilisateur de saisir un nombre et lui dire si ce nombre est SUP ou INF à 100

EXERCICE 2:
Demandez à l'utilisateur de saisir l'id d'un employé
si :
- le job_id = **IT_PROG** alors **afficher** son salaire + une augmentation de 0.8%
- le job_id = **FI_ACCOUNT** alors **afficher** son salaire + une augmentation de 0.5%
- le job_id = **ST_CLERK** alors **afficher** son salaire + une augmentation de 0.3%
- si non 0.2%
*/

EXERCICE preparation

```
select * from employees;
desc employees;
select job_id from employees;
select distinct job_id from employees
where job_id in ('AC_ACCOUNT','FI_ACCOUNT','FI_MGR');
---
select  EMPLOYEE_ID from employees
where job_id in ('AC_ACCOUNT','FI_ACCOUNT','FI_MGR');
-- Augmentation de salaire 0.9 pour AC_ACCOUNT
-- 0.08 pour  FI_ACCOUNT
-- 0.07 pour FI_MGR
```

EXERCICE preparation

```
DECLARE
   EMPID      EMPLOYEES.EMPLOYEE_ID%TYPE ;
      JOBID      EMPLOYEES.JOB_ID%TYPE ;
      ENAME EMPLOYEES.FIRST_NAME%TYPE;
   SAL_RAISE  NUMBER(3,2);
BEGIN
 EMPID := '&empid';
  DBMS_OUTPUT.PUT_LINE ('pour l''employé :'||EMPID);
  SELECT JOB_ID,FIRST_NAME INTO JOBID,ENAME FROM EMPLOYEES WHERE EMPLOYEE_ID = EMPID;
  IF JOBID = 'AC_ACCOUNT' THEN SAL_RAISE := .09;
  ELSIF JOBID = 'FI_ACCOUNT' THEN SAL_RAISE := .08;
  ELSIF JOBID = 'FI_MGR' THEN SAL_RAISE := .07;
  ELSE SAL_RAISE := 0;
  END IF;
  DBMS_OUTPUT.PUT_LINE (ENAME||' sera augmenté de  '||SAL_RAISE||' job id'||JOBID);
END;
```

IF THEN ELSE

```
DECLARE
v_num NUMBER := &sv_user_num;
BEGIN
    IF MOD(v_num,2) = 0 THEN
        DBMS_OUTPUT.PUT_LINE (v_num||' est un nombre paire');
    ELSE
        DBMS_OUTPUT.PUT_LINE (v_num||' un nombre impaire');
    END IF;
DBMS_OUTPUT.PUT_LINE (' Fin');
END;
```