INTRODUCTION A SQL*PLUS

Quelques commandes élémentaires

help	Aide de SQL*PLUS
exit, quit	Fin de SQL*PLUS
sqlplus	Exécution de SQL*PLUS

Editeur d'Oracle

l[ist]	Liste des lignes de commande (ligne courante marquée par ***)**
l*n* **or** *n*	Affichage de la ligne *n*
l*n m*	Affichage des lignes *n* à *m*
a *text*	Ajout d'une chaîne *text* à la ligne courante
c/*oldstring*/*newstring*	Change la chaîne *oldstring* par la chaîne *newstring* dans la ligne courante
i	Insère une ligne après la ligne courante
del	Efface la ligne courante
r[un]	Exécute et affiche le buffer de commandes
/	Exécute le buffer de commande
;	Affiche le contenu du buffer de commandes

Utilisation des fichiers SQL

Vous pouvez utiliser les fichiers de commandes pour sauver des commandes complexes. Le format par défaut utilise l'extension **.sql**.

sav[e] *filename*	Création du fichier *filename* et sauvegarde du buffer de commandes dans le fichier
sav[e] *filename* **[option]**	Sauvegarde du buffer de commandes dans le fichier *filename* Options possible: **cre[ate], app[end], rep[lace].**
get *filename*	Charge le contenu du fichier *filename* dans le buffer de commandes
sta[rt] *filename*	Exécute le fichier *filename* sans affichage des commandes
@ *filename*	Exécute le fichier *filename* sans affichage des commandes
r[un] *filename*	Exécute le fichier *filename* et affiche les différentes commandes

SQL : Manipulation des tables

Types syntaxiques (presque les domaines)

VARCHAR2(*n***)**	Chaîne de caractères de longueur variable (maximum *n*)
CHAR(*n***)**	Chaîne de caractères de longueur fixe (*n* caractères)
NUMBER	Nombre entier (40 chiffres au maximum)
NUMBER(*n***,***m***)**	Nombre entier de longueur totale *n* avec *m* décimales
DATE	Date (DD-MON-YY est le format par défaut)
LONG	Flot de caractères

Création de table

CREATE TABLE *table* **(**<*spécificationDeColonne*> **[,**<*spécificationDeColonne*>**]…[,**<*contrainte*>**]…)**
<*spécificationDeColonne*> = <*nom de colonne*> <*domaine*> **[NOT NULL]**
<*contrainte*> = **CONSTRAINT** <*nom de contrainte*> <*sorte de contrainte*>
<*sorte de contrainte*> = **PRIMARY KEY (***att1* **[,***att2***])**
FOREIGN KEY (*att1* **[,***att2***]) REFERENCES** *tableAssociée* **(***att1* **[,***att2***])**
CHECK (*att condition***)**

Exemple :

```
CREATE TABLE Commande (
   noCommande NUMBER,
   dateComm DATE,
   noClient NUMBER,
   CONSTRAINT key1 PRIMARY KEY (noArticle),
   CONSTRAINT key2 PRIMARY FOREIGN (noClient) REFERENCES Client (idClient)
);
```

Modification de la structure d'une table

ALTER TABLE *table* **ADD ([**<*spécificationDeColonne*>**]…[,**<*contrainte*>**]…)** Ajout de données
ALTER TABLE *table* **MODIFY ([**<*spécificationDeColonne*>**]…)** Modification de données

Exemples :

```
ALTER TABLE Commande ADD (
   noFabricant NUMBER,
   CONSTRAINT key3 PRIMARY FOREIGN (noFabricant) REFERENCES Fabricant (idFabricant)
);
ALTER TABLE Commande MODIFY (
   noCommande NUMBER(5,0)
);
```

Destruction d'une table

DROP TABLE *table* **[**<*option*>**]**
<*option*> = **CASCADE CONSTRAINTS**, **RESTRICT CONSTRAINTS**

Exemple :

```
DROP TABLE Commande CASCADE CONSTRAINTS;
```

Renommage d'une table

RENAME *table* **TO** *nouvelleTable*

Exemple :

```
RENAME Commande TO ListeCommande;
```

Consultation d'une table

DESC[RIBE] *table* Affiche la structure de la *table*

Exemple :

```
DESCRIBE Commande;
```

SQL : Exécution de requêtes

Sélection de colonnes dans une table (SELECT)

SELECT [DISTINCT] <*nom de colonne*>**[,**<*nom de colonne*>**]… FROM** *table*
DISTINCT : élimination des doublons

Exemples :

`SELECT * FROM Commande;`	Affiche toute les colonnes de *Commande*
`SELECT DISTINCT noCommande FROM Commande;`	Affiche sans doublon la colonne *noCommande*
`SELECT noCommande,noClient FROM Commande;`	Affiche les colonnes *noCommande* et *noClient*
`SELECT noCommande,noClient*3 FROM Commande;`	Affiche les colonnes *noCommande* et *noClient**3
`SELECT 2*3 FROM DUAL;`	Affiche 2*3 à l'écran

Sélection de lignes dans une table avec ou sans tri

SELECT [DISTINCT] <*nom de colonne*>**[,**<*nom de colonne*>**]… FROM** *table* **WHERE** <*condition*> **GROUP BY** <*nom de colonne*>**[,**<*nom de colonne*>**]… HAVING** <*condition*> **ORDER BY** <*nom de colonne*>**[,**<*nom de colonne*>**]… ASC**|**DESC**

WHERE <*condition*> : sélection de ligne vérifiant une condition donnée
GROUP BY <*nom de colonnes*> : regroupement des lignes dont le contenu des colonnes est identique

HAVING <*condition*> : permet de rajouter une condition de groupe (comptage,…)
ORDER BY <*nom de colonnes*> **ASC|DESC** : tri ascendant (ASC) ou descendant (DESC) par colonnes

<*condition*> = *attribut* **BETWEEN** <*valeur minimum*> **AND** <*valeur maximum*>
attribut **IN (**<*liste de valeurs*>**)**
attribut1 = <**valeur**> **AND|OR** *attribut2* >= <*valeur*>
attribut **LIKE '***chaine de caractères'* **(caractère joker '_' et chaine de caractères joker '%')**
attribut **IS NOT NULL**

Fonctions de groupes : **COUNT** : compte le nombre d'occurrence
SUM|MAX|MIN|AVG|STDDEV|VARIANCE : réalise la somme|maximum|minimum| moyenne|écart type|variance des données d'un groupe

Exemples sur la clause WHERE :
```
SELECT * FROM Commande WHERE dateComm>='2000-06-01' AND dateComm<='2000-06-30';
SELECT * FROM Commande WHERE dateComm BETWEEN '2000-06-01' AND '2000-06-30';
SELECT * FROM Commande WHERE noClient IN (10,40,80);
SELECT * FROM Commande WHERE noClient=10 OR noClient=40 OR noClient=80;
SELECT * FROM Client WHERE nomClient LIKE '%LE%';   (contient le mot 'LE')
SELECT * FROM Client WHERE nomClient LIKE '_o%k';   (la seconde lettre 'o' et dernière lettre 'k')
SELECT * FROM Article WHERE description IS NOT NULL;
```

Exemples sur la clause GROUP BY :
```
SELECT noClient,COUNT(*) AS nbCommandes FROM Commande GROUP BY noClient ASC;
SELECT noComm,noArticle,SUM(quantiteLivree) AS totalLivre,COUNT(*) AS nbLivraisons
   FROM DetailLivraison GROUP BY noComm,noArticle
```

Exemple sur la clause HAVING :
```
SELECT noClient,COUNT(*) AS nbCommandes FROM Commande GROUP BY noClient HAVING
   COUNT(*)>=2;
```

Exemple sur la clause ORDER BY :
```
SELECT * FROM Client ORDER BY nomClient DESC, noTelephone ASC;
```

Manipulation des données

INSERT INTO *table* **[(**<*nom de colonne*>**[**<*nom de colonne*>**]...] VALUES (**<*valeur*>**,[**<*valeur*>**]…)**
UPDATE *table* **SET** <*nom de colonne*>=<*valeur*> **WHERE** <*condition*>
DELETE FROM *table* **WHERE** <*condition*>

Exemples :
```
INSERT INTO Commande (noCommande) VALUES (10,20);
UPDATE Commande SET noCommande=5 WHERE noArticles=1025;
DELETE FROM Commande WHERE noCommande<10;
```

Opérateurs ensemblistes

UNION : retourne toutes les lignes de l'ensemble des deux tables, sans doublon
UNION ALL : retourne toutes les lignes de l'ensemble des deux tables, avec doublon(s)
INTERSECT : retourne toutes les lignes qui appartiennent aux deux tables à la fois
MINUS : retourne toutes les lignes de la première table n'appartenant par à la seconde table

Exemples :
```
SELECT * FROM table1 UNION SELECT * FROM table2;
SELECT * FROM table1 INTERSECT SELECT * FROM table2;
SELECT * FROM table1 MINUS SELECT * FROM table2;
```

Opérations spéciales (produit cartésien et jonction)

Produit cartésien

SELECT * FROM *table1*,*table2*

Exemple :
```
SELECT * FROM ClientRenault,ClientCitroen;
```

Jonction simple

SELECT * **FROM** *table1*, *table2***WHERE** *table1.col2***=** *table2.col2*
SELECT *col1*, *table1.col2*,*col3* **FROM** *table1*, *table2***WHERE** *table1.col2***=** *table2.col2*

Exemples :
```
SELECT * FROM Client,Commande WHERE Client.noClient=Commande.noClient;
SELECT Client.noClient,nomClient,neTelephone,noCommande FROM Client,Commande WHERE
   Client.noClient=Commande.noClient;
```

Jonction naturelle

SELECT * **FROM** *table1* **NATURAL JOIN** *table2*

Exemple :
```
SELECT * FROM Client NATURAL JOIN Commande;
```

Commandes de transaction

Une transaction (ou unité logique de travail) est une séquence de commandes SQL considérée comme unitaire, indivisible.

- Une transaction commence avec la première commande exécutable qui suit un **COMMIT**, un **ROLLBACK**, ou la connexion à la base de données.
- Une transaction se termine avec la commande **COMMIT**, **ROLLBACK**, ou lors d'une déconnexion.
- Utiliser la commande **COMMIT WORK** permet de valider la transaction courante : on ne peut plus revenir en arrière.
- Utiliser la commande **SAVEPOINT** <*name*> à l'intérieur d'une transaction permet de situer et nommer un point éventuel de retour vers l'état de la base de données au moment de la création de ce point de sauvegarde. Ce retour s'effectue avec la commande **ROLLBACK WORK TO SAVEPOINT** <*name*>**.**
- Utiliser la commande **ROLLBACK** sans autre option termine la transaction, annule tous les changements de cette transaction courante, et élimine tous les points de sauvegarde créés lors de cette transaction.

Fonctions arithmétiques

- **ABS(***nb***)** : Renvoie la valeur absolue de *nb*.
- **CEIL(***nb***)** : Renvoie le plus petit entier supérieur ou égal à *nb*.
- **COS(***n***)** : Renvoie le cosinus de *n*, *n* étant un angle exprimé en radians.
- **COSH(***n***)** : Renvoie le cosinus hyperbolique de *n*.
- **EXP(***n***)** : Renvoie e puissance *n*.
- **FLOOR(***nb***)** : Renvoie le plus grand entier inférieur ou égal à *nb*.
- **LN(***n***)** : Renvoie le logarithme népérien de *n* qui doit être un entier strictement positif.
- **LOG(***m***,***n***)** : Renvoie le logarithme en base *m* de *n*. *m* doit être un entier strictement supérieur à 1, et *n* un entier strictement positif.
- **MOD(***m***,***n***)** : Renvoie le reste de la division entière de *m* par *n*, si *n* vaut 0 alors renvoie *m*. Attention, utilisée avec au moins un de ses arguments négatifs, cette fonction donne des résultats qui peuvent être différents d'un modulo classique. Cette fonction ne donne pas toujours un résultat dont le signe du diviseur.
- **POWER(***m***,***n***)** : Renvoie m puissance *n*, *m* et *n* peuvent être des nombres quelconques entiers ou réels mais si *m* est négatif *n* doit être un entier.
- **ROUND(***n***[,***m***])** : Si *m* est positif, renvoie *n* arrondi (et non pas tronqué) à *m* chiffres après la virgule. Si *m* est négatif, renvoie *n* arrondi à *m* chiffres avant la virgule. *m* doit être un entier et il vaut 0 par défaut.
- **SIGN(***nb***)** : Renvoie -1 si *nb* est négatif, 0 si *nb* est nul, 1 si *nb* est positif.
- **SIN(***n***)** : Renvoie le sinus de *n*, *n* étant un angle exprimé en radians.
- **SINH(***n***)** : Renvoie le sinus hyperbolique de *n*.
- **SQRT(***nb***)** : Renvoie la racine carrée de *nb* qui doit être un entier positif ou nul.
- **TAN(***n***)** : Renvoie la tangente de *n*, *n* étant un angle exprimé en radians.
- **TANH(***n***)** : Renvoie la tangente hyperbolique de *n*.
- **TRUNC(***n***[,***m***])** : Si *m* est positif, renvoie *n* arrondi tronqué à *m* chiffres après la virgule. Si *m* est négatif, renvoie *n* tronqué à *m* chiffres avant la virgule. *m* doit être un entier et il vaut 0 par défaut.

Exemple : Donner pour chaque employé son salaire journalier.
```
SELECT nom, ROUND(salaire/22.2) FROM emp;
```

Expressions et fonctions sur les chaînes de caractères

- **CONCAT(***chaîne1***,***chaîne2***)** = *chaîne1* **||** *chaîne2* : Renvoie la chaîne obtenue en concaténant *chaîne1* à *chaîne2*. Cette fonction est équivalente à l'opérateur de concaténation | |.
- **INITCAP(***chaîne***)** : Renvoie *chaîne* en ayant mis la première lettre de chaque mot en majuscule et toutes les autres en minuscule. Les séparateurs de mots sont les espaces et les caractères non alphanumériques.
- **LOWER(***chaîne***)** : Renvoie *chaîne* en ayant mis toutes ses lettres en minuscules.
- **LPAD(***chaîne***,***long***,[***char***])** : Renvoie la chaîne obtenue en complétant, ou en tronquant, *chaîne* pour qu'elle ait comme longueur *long* en ajoutant éventuellement à gauche le caractère (ou la chaîne de caractères) *char*. La valeur par défaut de *char* est un espace.
- **LTRIM(***chaîne***[,***ens***])** : Renvoie la chaîne obtenue en parcourant à partir de la gauche *chaîne* et en supprimant tous les caractères qui sont dans *ens*. On s'arrête quand on trouve un caractère qui n'est pas dans *ens*. La valeur de défaut de *ens* est un espace.
- **REPLACE(***chaine***,***avant***,***après***)** : Renvoie *chaine* dans laquelle toutes les occurrences de la chaîne de caractères *avant* ont été remplacés par la chaîne de caractères *après*.
- **RPAD(***chaîne***,***n***,[***char***]) :** Renvoie la chaîne obtenue en complétant, ou en tronquant, *chaîne* pour qu'elle ait comme longueur *n* en ajoutant éventuellement à droite le caractère (ou la chaîne de caractères) *char*. La valeur par défaut de *char* est un espace.
- **RTRIM(***chaîne***[,***ens***]) :** Renvoie la chaîne obtenue en parcourant à partir de la droite *chaîne* et en supprimant tous les caractères qui sont dans *ens*. On s'arrête quand on trouve un caractère qui n'est pas dans *ens*. La valeur de défaut de *ens* est un espace.
- **SOUNDEX(***chaîne***) :** Renvoie la chaîne de caractères constituée de la représentation phonétique des mots de *chaîne*.
- **SUBSTR(***chaîne***,***m***[,***n***]) :** Renvoie la partie de *chaîne* commençant au caractère *m* et ayant une longueur de *n*.
- **TRANSLATE(***chaîne***,** *avant***,** *après***) :** Renvoie une chaîne de caractères en remplaçant chaque caractère de *chaîne* présent dans *avant* par le caractère situé à la même position dans *après*. Les caractères de *chaîne* non présents dans *avant* ne sont pas modifiés. *avant* peut contenir plus de caractères que *après*, dans ce cas les caractères de *avant* sans correspondant dans *après* seront supprimés de *chaîne*.
- **UPPER(***chaîne***) :** Renvoie *chaîne* en ayant mis toutes ses lettres en majuscules.
- *number* = **INSTR(***chaîne***,** *sous-chaîne***,** *debut***,** *occ***) :** Renvoie la position, noté *number*, du premier caractère de *chaîne* correspondant à l'occurrence *occ* de *sous-chaîne* en commençant la recherche à la position *début*.
- *number* = **LENGTH(***chaîne***) :** Renvoie la longueur de *chaîne*, noté *number*, exprimée en nombre de caractères.

Exemple : Afficher le nom des employés en MASJUCULE.
```
SELECT UPPER(nom) FROM emp;
```

Expressions et fonctions sur les dates

Opérateurs sur les dates

- *date* +/- *nombre* : le résultat est une date obtenue en ajoutant/soustrayant le *nombre* de jours nombre à la date *date*.
- *date2* - *date1* : le résultat est le nombre de jours entre les deux dates.

Fonctions sur les dates

- **ADD_MONTHS(***date***,***n***) :** Renvoie la date obtenue en ajoutant *n* mois à *date*. *n* peut être un entier quelconque. Si le mois obtenu a moins de jours que le jour de *date*, le jour obtenu est le dernier du mois.
- **LAST_DAY(***date***) :** Renvoie la date du dernier jour du mois de *date*.
- **MONTHS_BETWEEN(***date2***,** *date1***) :** Renvoie le nombre de mois entre *date2* et *date1*, si *date2* est après *date1* le résultat est positif, sinon le résultat est négatif. Si les jours *date2* et *date1* sont les mêmes, ou si ce sont les derniers jours du mois, le résultat est un entier. La partie fractionnaire est calculée en considérant chaque jour comme 1/31ème de mois
- **NEXT_DAY(***date***,***nom_du_jour***) :** Renvoie la date du prochain jour de la semaine dont le nom est *nom_de_jour*.
- **ROUND(***date***[,***précision***]) :** Renvoie *date* arrondie à l'unité spécifiée dans *précision*. L'unité de *précision* est indiquée en utilisant un des masques de mise en forme de la *date*. On peut ainsi arrondir une *date* à l'année, au mois, à la minute,... Par défaut la *précision* est le jour.

- **SYSDATE :** Renvoie la date et l'heure courantes du système d'exploitation hôte.
- **TRUNC(***date***[,***précision***]) :** Renvoie *date* tronquée à l'unité spécifiée dans *précision*. Les paramètres sont analogues à ceux de la fonction ROUND.

Exemple : Donner la date du lundi suivant l'embauche de chaque employé.
```
SELECT NEXT_DAY(embauche,'MONDAY') FROM emp;
```

Exemple : Donner la date d'embauche de chaque employé arrondie à l'année
```
SELECT ROUND(embauche,'Y') FROM emp;
```

Exemple : Donner pour chaque employé le nombre de jours depuis son embauche.
```
SELECT ROUND(SYSDATE-embauche) FROM emp;
```

Fonctions de conversion

- **ASCII(***chaîne***) :** Renvoie le nombre correspondant au code ascii du premier caractère de *chaine*.
- **CHR(***nombre***) :** Renvoie le caractère dont *nombre* est le code ascii.
- **TO_CHAR(***nombre***,** *format***) :** Renvoie la chaîne de caractères en obtenue en convertissant *nombre* en fonction de *format*. *Format* est une chaîne de caractères pouvant contenir les caractères suivants :
  - **9** : représente un chiffre (non représenté si non significatif)
  - **0** : représente un chiffre (représenté même si non significatif)
  - **.** : point décimal apparent
  - **V** : définit la position du point décimal non apparent
  - **,** : une virgule apparaîtra à cet endroit
  - **$** : un $ précédera le premier chiffre significatif
  - **B** : le nombre sera représenté par des blancs s'il vaut 0
  - **EEEE** : le nombre sera représenté avec un exposant (le spécifier avant MI ou PR)
  - **MI** : le signe négatif sera à droite
  - **PR** : un nombre négatif sera entre <>
- **TO_CHAR(***date***,** *format***) :** Renvoie conversion d'une *date* en chaîne de caractères. Le *format* indique quelle partie de la *date* doit apparaître, c'est une combinaison des codes suivants :
  - **SCC** : siècle avec signe
  - **CC** : siècle
  - **SY,YYY** : année (avec signe et virgule)
  - **Y,YYY** : année(avec virgule)
  - **YYYY** : année
  - **YYY** : 3 derniers chiffres de l'année
  - **YY** : 2 derniers chiffres de l'année
  - **Y** : dernier chiffre de l'année
  - **Q** : numéro du trimestre dans l'année
  - **WW** : numéro de la semaine dans l'année
  - **W** : numéro de la semaine dans le mois
  - **MM** : numéro du mois
  - **DDD** : numéro du jour dans l'année
  - **DD** : numéro du jour dans le mois
  - **D** : numéro du jour dans la semaine
  - **HH** ou **HH12** : heure (sur 12 heures)
  - **HH24** : heure sur 24 heures
  - **MI** : minutes
  - **SS** : secondes
  - **SSSSS** : secondes après minuit
  - **J** : jour du calendrier julien

  Les formats suivants permettent d'obtenir des dates en lettres (en anglais) :
  - **SYEAR** ou **YEAR** : année en toutes lettres
  - **MONTH** : nom du mois
  - **MON** : nom du mois abrégé sur 3 lettres
  - **DAY** : nom du jour
  - **DY** : nom du jour abrégé sur 3 lettres
  - **AM** ou **PM** : indication *am* ou *pm*
  - **BC** ou **AD** : indication avant ou après Jésus Christ

Les suffixes suivants modifient la présentation du nombre auquel ils sont accolés :
- **TH** : ajout du suffixe ordinat *st*, *nd*, *rd*, *th*
- **SP** : nombre en toutes lettres

Tout caractère spécial inséré dans le format sera reproduit tel quel dans la chaîne de caractères résultat.

- **TO_DATE(***chaîne***,** *format***) :** Permet de convertir une *chaîne* de caractères en donnée de type *date*. Le *format* est identique à celui de la fonction TO_CHAR.
- **TO_NUMBER(***chaîne***) :** Convertit *chaîne* en sa valeur numérique.

Remarque : On peut également insérer dans le format une chaîne de caractères quelconque, à condition de la placer entre guillemets "".

Exemple : Donner les dates d'embauche de tous les employés sous le format : jour/mois/année heure:minute:seconde
```
SELECT TO_CHAR(embauche,'DD/MM/YY HH24:MI:SS') FROM emp;
```

Exemple : Donner la liste de tous les employés dont le nom ressemble à 'DUPONT'.
```
SELECT nom FROM emp WHERE SOUNDEX(nom)=SOUNDEX('DUPONT');
```

Exemple : Donner la liste de tous les noms des employés en ayant supprimé tous les 'L' et les 'E' en tête des noms.
```
SELECT LTRIM(nom,'LE') FROM emp;
```

Exemple : Donner la liste de tous les noms des employés en ayant remplacé les A et les M par des * dans les noms.
```
SELECT TRANSLATE(nom,'AM','**') FROM emp;
```

Exemple : Afficher tous les salaires avec un $ en tête et au moins trois chiffres (dont deux décimales).
```
SELECT TO_CHAR(salaire,'<MATH>99900.00') FROM emp;
```

Autres fonctions

- **GREATEST(***expr1***,** *expr2***,...) :** Renvoie la plus grande des valeurs *expr1*, *expr2*,.... Toutes les expressions sont converties au format de *expr1* avant comparaison.
- **LEAST(***expr1***,** *expr2***,...) :** Renvoie la plus petite des valeurs *expr1*, *expr2*,.... Toutes les expressions sont converties au format de *expr1* avant comparaison.
- **NVL(***expr1***,** *expr2***) :** Prend la valeur *expr1*, sauf si *expr1* est NULL auquel cas NVL prend la valeur *expr_2*. Une valeur NULL en SQL est une valeur non définie. Lorsque l'un des termes d'une expression a la valeur NULL, l'expression entière prend la valeur NULL. D'autre part, un prédicat comportant une comparaison avec une expression ayant la valeur NULL prendra toujours la valeur faux. La fonction NVL permet de remplacer une valeur NULL par une valeur significative.
- **DECODE(***crit***,** *val_1***,** *res_1* **[,** *val_2***,** *res_2* ...**],** *def***)** : Cette fonction permet de choisir une valeur parmi une liste d'expressions, en fonction de la valeur prise par une expression servant de critère de sélection.
  Le résultat récupéré est :
  - *res_1* si l'expression *crit* a la valeur *val_1*
  - *res_2* si l'expression *crit* a la valeur *val_2*
  - *def* (la valeur par défaut) si l'expression *crit* n'est égale à aucune des expressions *val_1*, *val_2*,...,.

  Les expressions résultats *res_1*, *res_2*, ..., *def* peuvent être de types différents : caractère et numérique, ou caractère et date (le résultat est du type de la première expression rencontré dans le DECODE).
  La fonction DECODE permet également de mélanger dans une colonne résultat des informations venant de plusieurs colonnes d'une même table.

Exemple : Donner pour chaque employé ses revenus (salaire + commission).
```
SELECT nom, salaire, comm, salaire+NVL(comm,0) FROM emp;
```

Exemple : Donner la liste des employés avec pour chacun d'eux sa catégorie (président = 1, directeur = 2, autre = 3)
```
SELECT nom, DECODE(fonction,'president',1,'directeur',2,3) FROM emp;
```

Exemple : Donner la liste des employés en les identifiant par leur fonction dans le département 10 et par leur nom dans les autres départements.
```
SELECT DECODE(n_dept,10,fonction,nom) FROM emp;
```