



## Langage d'Interrogation des données (LID)

# Plan

1. **SQL Basique**
2. **Fonctions monolignes**
3. **Fonctions de groupes**

# SELECT – Accès aux données

Un ordre **SELECT** permet d'extraire des informations d'une BD

- L'utilisation d'un ordre **SELECT** offre les possibilités suivantes:
- **Sélection** : SQL permet de choisir dans une table, les lignes que l'on souhaite ramener au moyen d'une requête. Divers critères de sélection sont disponibles à cet effet
- **Projection** : SQL permet de choisir dans une table, les colonnes que l'on souhaite ramener au moyen d'une requête. Vous pouvez déterminer autant de colonnes que vous le souhaitez
- **Jointure** : SQL permet de joindre des données stockées dans différentes tables, en créant un lien par le biais d'une colonne commune à chacune des tables

# Select – Les Possibilités de l'Ordre SQL SELECT

**Sélection**


**Table 1**

**Projection**


**Table 1**

**Jointure**




**Table 1**


**Table 2**

## Select – Syntaxe

```
SELECT    *|{[DISTINCT] column|expression [alias],...}  
FROM      table;
```

- Pour restreindre les lignes, on utilise la clause Where

```
SELECT          [DISTINCT] {*, column [alias],  
FROM            ...}  
[WHERE          table  
                condition(s)]  
;
```

# Sélectionner les Lignes

EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

“...rechercher  
tous les employés  
du département  
10”



EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7782	CLARK	MANAGER		10
7934	MILLER	CLERK		10

## Chaînes de Caractères et Dates

- **Les constantes chaînes de caractères et dates doivent être placées entre simples quotes.**
- **La recherche tient compte des majuscules et minuscules (pour les chaînes de caractère) et du format (pour les dates.)**
- **Le format de date par défaut est 'DD-MON-YY'.**

```
SQL> SELECT  ename, job, deptno  
2 FROM      emp  
3 WHERE     ename = 'JAMES';
```

## Select – Opérateurs de comparaison

**Les opérateurs de comparaison sont ceux du Pascal : <, >, =, <=, >=, != et aussi BETWEEN, NOT BETWEEN, IN, NOT IN, LIKE et IS NULL, IS NOT NULL**

- Il faut être attentif aux différences entre chaînes de longueur fixe et chaînes de longueur variable. Les premières sont complétées par des blancs ( ' ') et pas les secondes**
- Si SQL ne distingue pas majuscules et minuscules pour les mots-clés, il n'en va pas de même pour les valeurs. Donc 'SANTALBA' est différent de 'Santalba'**



## Utilisation des Opérateurs de Comparaison

```
SQL> SELECT ename, sal, comm  
2 FROM emp  
3 WHERE sal<=comm;
```

ENAME	SAL	COMM
MARTIN	1250	1400

# Utilisation de l'Opérateur

## BETWEEN

- **BETWEEN** permet de tester l'appartenance à une fourchette de valeurs.

```
SQL> SELECT  ename, sal
2  FROM      emp
3  WHERE      sal BETWEEN 1000 AND 1500;
```

ENAME	SAL	Limite	Limite
-----	-----	inférieure	supérieure
MARTIN	1250		
TURNER	1500		
WARD	1250		
ADAMS	1100		
MILLER	1300		

# Utilisation de l'Opérateur

## IN

- **IN permet de comparer une expression avec une liste de valeurs.**

```
SQL> SELECT empno, ename, sal, mgr
2 FROM emp
3 WHERE mgr IN (7902, 7566, 7788);
```

EMPNO	ENAME	SAL	MGR
7902	FORD	3000	7566
7369	SMITH	800	7902
7788	SCOTT	3000	7566
7876	ADAMS	1100	7788

# Select – Utilisation de l'Opérateur

## LIKE

- **LIKE** permet de rechercher des chaînes de caractères à l'aide de caractères génériques
- Les conditions de recherche peuvent contenir des caractères ou des nombres littéraux.
  - (%) représente zéro ou plusieurs caractères
  - ( \_ ) représente un caractère

```
SQL> SELECT  ename  
2 FROM      emp  
3 WHERE     ename LIKE 'S%';
```

# Select – Utilisation de l'Opérateur

## LIKE

- Vous pouvez combiner plusieurs caractères génériques de recherche.

```
SQL> SELECT  ename  
      2 FROM    emp  
      3 WHERE   ename LIKE '_A%';
```

ENAME

-----

JAMES

WARD

- Vous pouvez utiliser l'identifiant ESCAPE pour rechercher "%" ou "\_".

# Select – Utilisation de l'Opérateur IS

## NULL

- Recherche de valeurs NULL avec l'opérateur IS NULL

```
SQL> SELECT  ename, mgr  
2 FROM      emp  
3 WHERE     mgr IS NULL;
```

ENAME	MGR
-----	-----
KING	

## Opérateurs Logiques

Opérateur	Signification
AND	Retourne TRUE si <i>les deux</i> conditions sont VRAIES
OR	Retourne TRUE si <i>l'une au moins</i> des conditions est VRAIE <u>Ramène la valeur TRUE si la condition qui NOT</u> suit l'opérateur est FAUSSE

# Select – Utilisation de l'Opérateur AND

- Avec AND, les deux conditions doivent être VRAIES.

```
SQL> SELECT empno, ename, job, sal
2 FROM emp
3 WHERE sal >= 1100
4 AND job = 'CLERK';
```

EMPNO	ENAME	JOB	SAL	
7876	ADAMS	CLERK	1100	
7934	MILLER	CLERK	1300	



## Select – Utilisation de l'Opérateur OR

- Avec OR, l'une ou l'autre des deux conditions doit être VRAIE.

```
SQL> SELECT empno, ename, job, sal
2 FROM emp
3 WHERE sal >= 1100
4 OR job = 'CLERK';
```

EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7566	JONES	MANAGER	2975
7654	MARTIN	SALESMAN	1250
...			

14 rows selected.

## Utilisation de l'Opérateur NOT

```
SQL> SELECT ename, job
2 FROM emp
3 WHERE job NOT IN ('CLERK', 'MANAGER', 'ANALYST');
```

ENAME	JOB
KING	PRESIDENT
MARTIN	SALESMAN
ALLEN	SALESMAN
TURNER	SALESMAN
WARD	SALESMAN

```
... WHERE sal NOT BETWEEN 1000
... WHERE AND 1500 ename NOT LIKE '%A%'
... WHERE comm IS NOT NULL
```

## Select – Règles de priorité

Ordre de priorité	Opérateur
1	Tous les opérateurs de comparaison
2	NOT
3	AND
4	OR

**Remarque:** Les parenthèses permettent de modifier les règles de priorité


## Select – Règles de priorité

```
SQL> SELECT ename, job, sal
2 FROM emp
3 WHERE
4   job='SALESMAN'
5   OR job='PRESIDENT'
6   AND sal>1500;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
MARTIN	SALESMAN	1250
ALLEN	SALESMAN	1600
TURNER	SALESMAN	1500
WARD	SALESMAN	1250

## Select - Règles de priorité

```
SQL> SELECT  ename, job, sal
  2  FROM      emp
  3  WHERE      (job='SALESMAN'
  4  OR          job='PRESIDENT')
  5  AND          sal>1500;
```



ENAME	JOB	SAL
KING	PRESIDENT	5000
ALLEN	SALESMAN	1600

## Select – Expressions arithmétiques

- **Possibilité de créer des expressions avec des données de type NUMBER et DATE au moyen d'opérateurs arithmétiques.**

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division

## Select – Expressions arithmétiques (Exemple)

```
SQL> SELECT ename, sal, sal+300  
2      FROM emp;
```

ENAME	SAL	SAL+300
KING	5000	5300
BLAKE	2850	3150
CLARK	2450	2750
JONES	2975	3275
MARTIN	1250	1550
ALLEN	1600	1900
...		

14 rows selected.

## Priorité des Opérateurs



- La multiplication et la division ont priorité sur l'addition et la soustraction.
- A niveau de priorité identique, les opérateurs sont évalués de gauche à droite.
- Les parenthèses forcent la priorité d'évaluation et permettent de clarifier les ordres.



## Priorité des Opérateurs

```
SQL> SELECT ename, sal, 12*sal+100  
2      FROM emp;
```

ENAME	SAL	12*SAL+100
KING	5000	60100
BLAKE	2850	34300
CLARK	2450	29500
JONES	2975	35800
MARTIN	1250	15100
ALLEN	1600	19300
...		

14 rows selected.

## Utilisation des Parenthèses

```
SQL> SELECT ename, sal, 12*(sal+100)
       2 FROM emp;
```

ENAME	SAL	12*(SAL+100)
KING	5000	61200
BLAKE	2850	35400
CLARK	2450	30600
JONES	2975	36900
MARTIN	1250	16200
...		

14 rows selected.

# La Valeur NULL

- **NULL représente une valeur non disponible, non affectée, inconnue ou inapplicable.**
- **La valeur NULL est différente du zéro ou de l'espace.**

```
SQL> SELECT  ename, job, comm  
2    FROM    emp;
```

ENAME	JOB	COMM
KING	PRESIDENT	
BLAKE	MANAGER	
...		
TURNER	SALESMAN	0
...		

14 rows selected.

# Valeurs NULL dans les Expressions

## Arithmétiques

- Les expressions arithmétiques comportant une valeur NULL sont évaluées à NULL

```
SQL> select  ename , 12*sal+comm  
2   from      emp  
3  WHERE      ename='KING';
```

ENAME	12*SAL+COMM
-----	-----
KING	

## Select – Alias

- Permettent de renommer des colonnes à l'affichage ou des tables dans la requête.
- Les alias de colonnes sont utiles pour les calculs.
- Suit immédiatement le nom de la colonne ; le mot-clé AS placé entre le nom et l'alias est optionnel.
- Doit obligatoirement être inclus entre guillemets s'il contient des espaces, des caractères spéciaux ou si les majuscules/minuscules doivent être différenciées.
- ORACLE traduit les noms des alias en majuscules

```
SELECT nome AS "Nom Employé",  
sal AS "Salaire Mensuel",  
Sal*12 AS "Salaire Annuel" FROM emp  
WHERE idService NOT IN (10,40,60)
```

## Select – Utilisation des alias de colonnes

```
SQL> SELECT ename AS name, sal salary  
2      FROM emp;
```

NAME	SALARY
-----	
...	

```
SQL> SELECT ename "Name",  
2           sal*12 "Annual Salary"  
3      FROM emp;
```

Name	Annual Salary
-----	
...	

## Select – Opérateur de concaténation

- L'opérateur de concaténation permet de concaténer des expressions (colonnes, calculs, fonctions ou constantes).  
La colonne résultante est considérée comme une chaîne de caractères.

```
SQL> SELECT 2      ename||job AS "Employees"  
          FROM emp;
```

Employees

```
-----  
KINGPRESIDENT  
BLAKEMANAGER  
CLARKMANAGER  
JONESMANAGER  
MARTINSALESMAN  
ALLENSALESMAN
```

```
...  
14 rows selected.
```

# Littéral

- **Un littéral est un caractère, une expression, ou un nombre inclus dans la liste SELECT.**
- **Les valeurs littérales de type date et caractère doivent être placées entre simples quotes.**
- **Chaque littéral apparaît sur chaque ligne ramenée.**



## Utilisation des Chaînes de Caractères Littérales

```
SQL> SELECT ename || ' ' || 'is a' || ' ' || job  
2          AS "Employee Details"  
3 FROM      emp;
```

```
Employee Details  
-----  
KING is a PRESIDENT  
BLAKE is a MANAGER  
CLARK is a MANAGER  
JONES is a MANAGER  
MARTIN is a SALESMAN  
...  
14 rows selected.
```

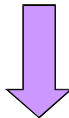
# Select - Élimination des enregistrements redondants

- Les directives **DISTINCT** ou **UNIQUE** éliminent les éventuels duplicatas.

```
SELECT DISTINCT idSer FROM emp;
```

idEmp nomEmp idSer salaire

100	Michel	20	2000
200	Sylvie	10	3000
300	Bernard	20	1000
400	Claude	10	2000
500	Thomas	10	1000



idSer
----
- 10
20

(Projection)

## Select – WHERE

- Les éléments de la clause **WHERE** d'une requête permettent de programmer l'opérateur de restriction. Cette clause limite la recherche aux enregistrements qui respectent une condition **simple** ou **complexe**.
- **condition:** est composée de colonnes, d'expressions, de constantes liées deux à deux entre opérateurs:
  - De comparaison (>, =, <, >=, <=, <>);
  - Logique (NOT, AND, OR);
  - Intégrés (BETWEEN, IN, LIKE, IS NULL).

## Select – ORDER BY

- **Tri des lignes avec la clause ORDER BY**
  - ASC : ordre croissant (par défaut)
  - DESC : ordre décroissant
- **La clause ORDER BY se place à la fin de l'ordre SELECT**

```
SQL> SELECT  ename, job, deptno, hiredate
2  FROM      emp
3  ORDER BY  hiredate;
```

ENAME	JOB	DEPTNO	HIREDATE
SMITH	CLERK	20	17-DEC-80
ALLEN	SALESMAN	30	20-FEB-81
...			

14 rows selected.

## Select – ORDER BY (Tri décroissant)

```
SQL> SELECT  ename, job, deptno, hiredate
2  FROM      emp
3  ORDER BY  hiredate DESC;
```

ENAME	JOB	DEPTNO	HIREDATE
ADAMS	CLERK	20	12-JAN-83
SCOTT	ANALYST	20	09-DEC-82
MILLER	CLERK	10	23-JAN-82
JAMES	CLERK	30	03-DEC-81
FORD	ANALYST	20	03-DEC-81
KING	PRESIDENT	10	17-NOV-81
MARTIN	SALESMAN	30	28-SEP-81
...			

14 rows selected.

## Select - ORDER BY (tri sur alias)

```
SQL> SELECT empno, ename, sal*12 annsal  
2 FROM emp  
3 ORDER BY annsal;
```

EMPNO	ENAME	ANNSAL
7369	SMITH	9600
7900	JAMES	11400
7876	ADAMS	13200
7654	MARTIN	15000
7521	WARD	15000
7934	MILLER	15600
7844	TURNER	18000
...		

14 rows selected.

## Select – ORDER BY (tri sur plusieurs lignes)

- L'ordre des éléments de la liste ORDER BY donne l'ordre du tri.

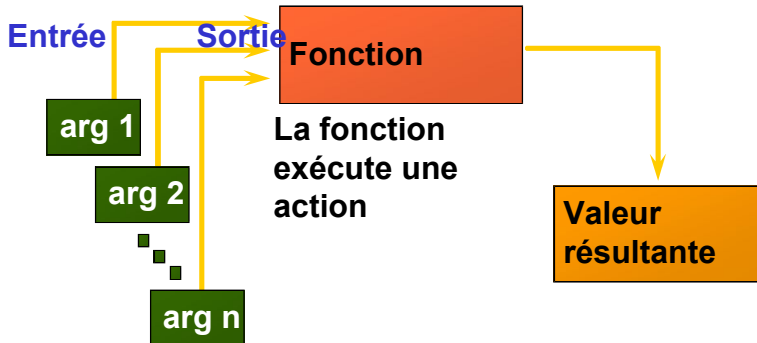
```
SQL> SELECT  ename, deptno, sal  
2   FROM      emp  
3   ORDER BY deptno, sal DESC;
```

ENAME	DEPTNO	SAL
KING	10	5000
CLARK	10	2450
MILLER	10	1300
FORD	20	3000
...		

14 rows selected.

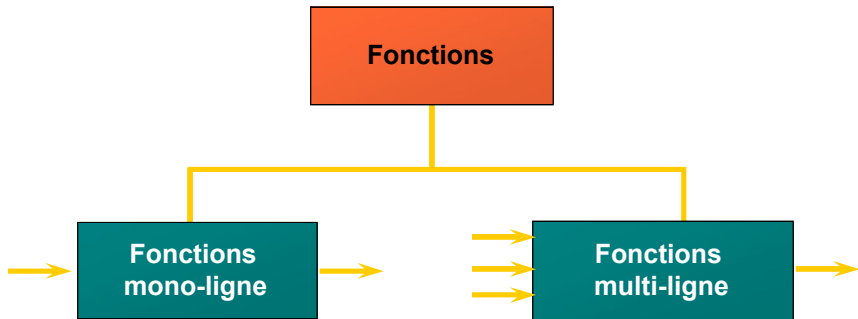
## Fonctions SQL – Définition

- Une fonction est une expression d'un type de données spécifique qui fait partie d'une instruction utilisée pour calculer une valeur .





## Fonctions SQL – Deux types

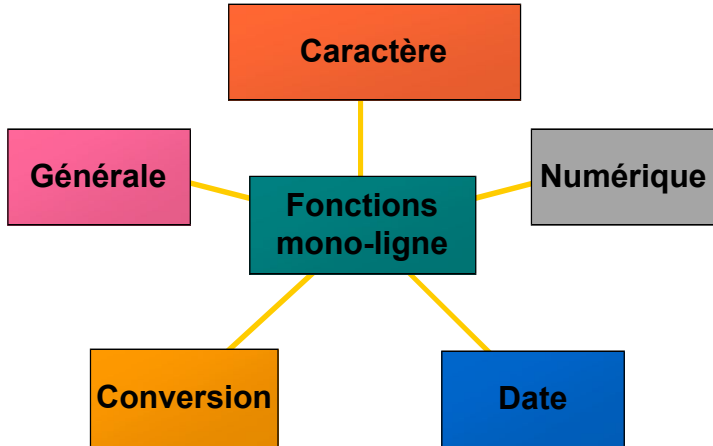


# Fonctions Mono-Ligne

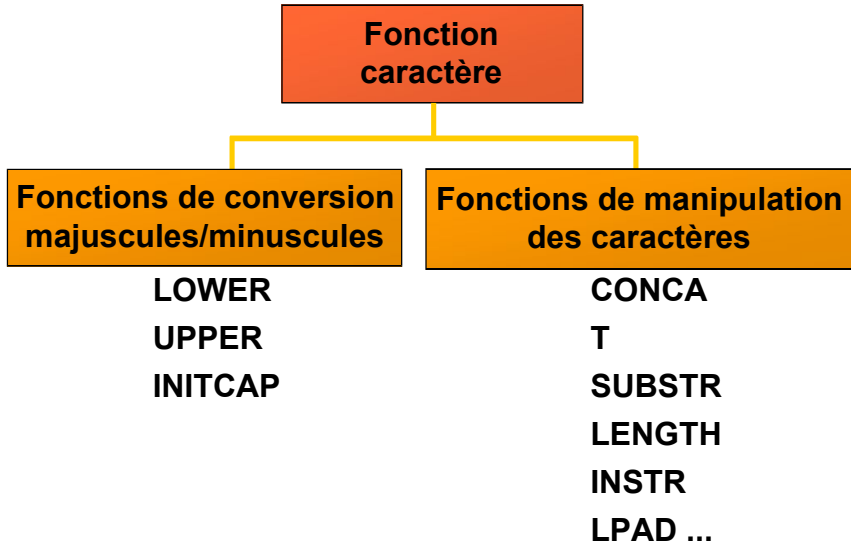
- Manipulent des éléments de données
- Acceptent des arguments et ramènent une valeur
- Agissent sur chacune des lignes rapportées
- Ramènent un seul résultat par ligne
- Peuvent modifier les types de données
- Peuvent être imbriquées

```
function_name (column|expression, [arg1,  
arg2, ...])
```

# Fonctions Mono-Ligne



# Fonctions Caractère



# Fonctions de Conversion Majuscules/Minuscules

Fonction	Résultat
LOWER('Cours SQL')	cours sql
UPPER('Cours SQL')	COURS SQL
INITCAP('Cours SQL')	Cours Sql

# Utilisation des Fonctions de Conversion Majuscules/Minuscules

- Afficher le matricule, le nom et le numéro de département de l'employé Blake.

```
SQL> SELECT empno, ename, deptno
  2 FROM emp
  3 WHERE ename = 'blake';
no rows selected
```

```
SQL> SELECT empno, ename, deptno
  2 FROM emp
  3 WHERE LOWER(ename) = 'blake';
```

EMPNO	ENAME	DEPTNO
7698	BLAKE	30

# Fonctions de Manipulation des Caractères

- **Manipulation de chaînes de caractères**

Fonction	Résultat
CONCAT('Une', 'Chaîne')	UneChaîne
SUBSTR('Chaîne',1,3)	Cha
LENGTH('Chaîne')	6
INSTR('Chaîne', 'a')	3
LPAD(sal,10,'*')	*****5000
RPAD(sal,10,'*')	5000*****

## Utilisation des Caractères

## Fonctions de des

## Manipulation

```
SQL> SELECT ename, CONCAT (ename, job),  
LENGTH(ename) INSTR(ename, 'A')  
3 FROM emp  
4 WHERE SUBSTR(job,1,5) = 'SALES';
```

ENAME	CONCAT (ENAME, JOB)	LENGTH (ENAME)	INSTR (ENAME, 'A')
MARTIN	MARTINSALESMAN	6	2
ALLEN	ALLENSALESMAN	5	1
TURNER	TURNERSALESMAN	6	0
WARD	WARDSALESMAN	4	2



# Fonctions Numériques

- **ROUND** : Arrondit la valeur à la précision spécifiée

**ROUND(45.926, 2)**      **45.93** →

- **TRUNC** : Tronque la valeur à la précision spécifiée

**TRUNC(45.926, 2)**      **45.92** →

- **MOD** : ramène le reste d'une division

**MODE(1600,300)**      →      **100**

# Utilisation de la Fonction ROUND

- **Affichage de la valeur 45.923 arrondie au centième, à 0 décimale et à la dizaine supérieure.**

```
SQL> SELECT ROUND(45.923,2), ROUND(45.923,0),  
2      ROUND(45.923,-1)  
3      FROM DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
45.92	46	50

# Utilisation de la Fonction TRUNC

- **Affichage de la valeur 45.923 tronquée au centième, à 0 décimale et à la dizaine.**

```
SQL> SELECT TRUNC(45.923,2), TRUNC(45.923),  
2      TRUNC(45.923,-1)  
3      FROM DUAL;
```

```
TRUNC(45.923,2) TRUNC(45.923)  
TRUNC(45.923,-1)
```

45.92      45      40

# Utilisation de la Fonction MOD

- Calculer le reste de la division salaire par commission pour l'ensemble des employés ayant un poste de vendeur.

```
SQL> SELECT ename, sal, comm, MOD(sal, comm)
2  FROM emp
3  WHERE job = 'SALESMAN';
```

ENAME	SAL	COMM	MOD (SAL, COMM)
MARTIN	1250	1400	1250
ALLEN	1600	300	100
TURNER	1500	0	1500
WARD	1250	500	250

## Autres Fonctions Numériques

- **ABS(x)** : Valeur absolue de x
- **CEIL(n)** : Plus petit entier supérieur ou égal à n.
- **SIGN(n)** : Si  $n < 0$ , -1; si  $n = 0$ , 0; si  $n > 0$ , 1.
- **FLOOR(n)** : Plus grand entier supérieur ou égal à n.

## Utilisation des Dates

- **Oracle stocke les dates dans un format numérique interne : siècle, année, mois, jour, heures, minutes, secondes.**
- **Le format de date par défaut est DD-MON-YY.**
- **La fonction SYSDATE ramène la date et l'heure courante.**
- **DUAL est une table factice qu'on peut utiliser pour visualiser SYSDATE.**

## Opérations Arithmétiques sur les Dates

- Ajout ou soustraction d'un nombre à une date pour obtenir un résultat de type **date**.
- Soustraction de deux dates afin de déterminer le **nombre** de jours entre ces deux dates.
- Ajout **d'un nombre d'heures** à une date en divisant le nombre d'heures par 24.

## Utilisation d'Opérateurs Arithmétiques avec les Dates

```
SQL> SELECT ename, (SYSDATE-hiredate)/7 WEEKS  
2 FROM emp  
3 WHERE deptno = 10;
```

ENAME	WEEKS
KING	830.93709
CLARK	853.93709
MILLER	821.36566



## Fonctions Date

FONCTION	DESCRIPTION
MONTHS_BETWEEN(d1,d2)	Nombre de mois situés entre deux dates
ADD_MONTHS(date, n)	Ajoute des mois calendaires à une date
NEXT_DAY(date,'char')	Jour qui suit la date spécifiée
LAST_DAY(date)	Dernier jour du mois
ROUND(date [, 'fmt' ] )	Arrondit une date
TRUNC (date [, 'fmt' ] )	Tronque une date

## Fonctions monolignes - Date

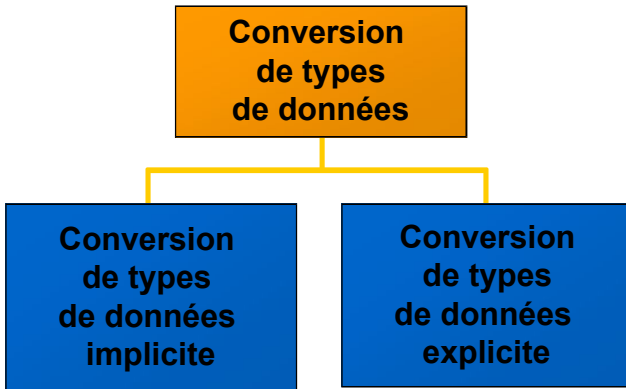
Année en Cours	Date Spécifiée	Format RR	Format YY
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		Si l'année spécifiée est située entre	
			0-49      50-99
Si les 2 chiffres de l'année en cours sont	0-49	La nouvelle date appartient au siècle courant.	La nouvelle date appartient au siècle précédent.
	50-99	La nouvelle date appartient au siècle suivant.	La nouvelle date appartient au siècle courant.

## Utilisation des Fonctions Date

- **MONTHS\_BETWEEN ('01-SEP-95','11-JAN-94')**  
→ 19.6774194
- **ADD\_MONTHS ('11-JAN-94',6)** → '11-JUL-94'
- **NEXT\_DAY ('01-SEP-95','FRIDAY')** → '08-SEP-95'
- **LAST\_DAY('01-SEP-95')** → '30-SEP-95'

# Fonctions de Conversion



## Conversion de Types de Données Implicite

- Pour les affectations, Oracle effectue automatiquement les conversions suivantes

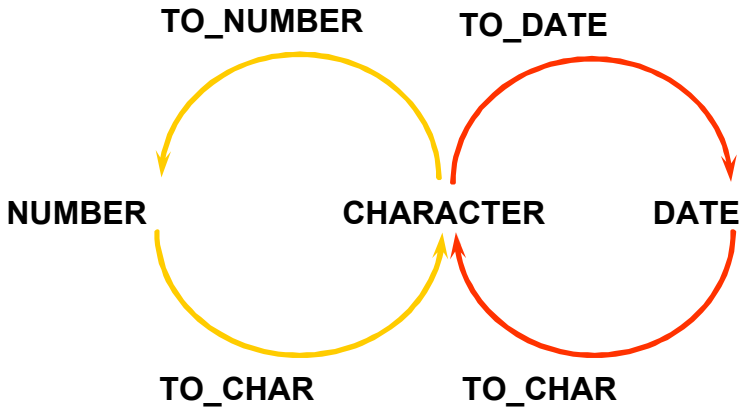
De	Vers
VARCHAR2 ou CHAR	NUMBER
VARCHAR2 ou CHAR	DATE
NUMBER	VARCHAR 2
DATE	VARCHAR 2

# Conversion de Types de Données Implicite

- Pour l'évaluation d'expressions, Oracle effectue automatiquement les conversions suivantes

De	Vers
VARCHAR2 ou CHAR	NUMBER
VARCHAR2 ou CHAR	DATE

# Conversion de Types de Données Explicite



# Utilisation de la Fonction TO\_CHAR avec les Dates

```
TO_CHAR(date, 'fmt') 
```

## Le modèle de format :

- Doit être placé entre simples quotes et différencie les majuscules et minuscules.
- Peut inclure tout élément valide de format date
- Comporte un élément *fm* qui supprime les espaces de remplissage ou les zéros de tête
- Est séparé de la valeur date par une virgule



## Modèles de Format Date

YYYY	Année exprimée avec 4 chiffres
YEAR	Année exprimée en toutes lettres
MM	Mois exprimé avec 2 chiffres
MONTH	Mois exprimé en toutes lettres
DY	3 premières lettres du nom du jour
DA	Jour exprimé en toutes lettres

Y

# Modèles de Format pour les Dates

- Les éléments horaires formatent la partie horaire de la date.

HH24:MI:SS AM	15:45:32 PM
---------------	-------------

- Pour ajouter des chaînes de caractères, les placer entre guillemets.

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- Différents suffixes existent pour les nombres.

ddspth	fourteenth
--------	------------

## Utilisation de la Fonction TO\_CHAR avec les Dates

```
SQL> SELECT  ename,  
2           TO_CHAR(hiredate, 'fmDD Month YYYY') HIREDATE  
3 FROM      emp;
```

ENAME	HIREDATE
-----	-----
KING	17 November 1981
BLAKE	1 May 1981
CLARK	9 June 1981
JONES	2 April 1981
MARTIN	28 September 1981
ALLEN	20 February 1981
...	

14 rows selected.

## Utilisation de la Fonction TO\_CHAR avec les Nombres

- Utilisez les formats suivants avec TO\_CHAR pour afficher un nombre sous la forme d'une chaîne de caractère.

```
TO_CHAR(number, 'fmt') 
```

9	Représente un chiffre
0	Force l'affichage du zéro
\$	Place un signe dollar flottant
L	Utilise le symbole monétaire local flottant
.	Imprime un point décimal
,	Imprime un séparateur de milliers

## Utilisation de la Fonction TO\_CHAR avec les Nombres

```
SQL> SELECT TO_CHAR(sal, '$99,999') SALARY  
2 FROM emp  
3 WHERE ename = 'SCOTT';
```

```
SALARY  
-----  
$3,000
```

## Fonctions TO\_NUMBER et TO\_DATE

- Conversion d'une chaîne de caractères en format numérique avec la fonction **TO\_NUMBER**

```
TO_NUMBER ( char )
```

- Conversion d'une chaîne de caractères en format date avec la fonction **TO\_DATE**

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

## Fonction NVL

- **Convertit une valeur NULL en une valeur réelle**
- **Fonctionne avec les données de type date, caractère et numérique.**
- **Les types de données doivent correspondre**
  - NVL(comm,0)
  - NVL(hiredate,'01-JAN-97')
  - NVL(job,'No Job Yet')

## Utilisation de la Fonction NVL

```
SQL> SELECT ename, sal, comm,  
           (sal*12)+NVL(comm,0)  
           2 FROM emp;
```

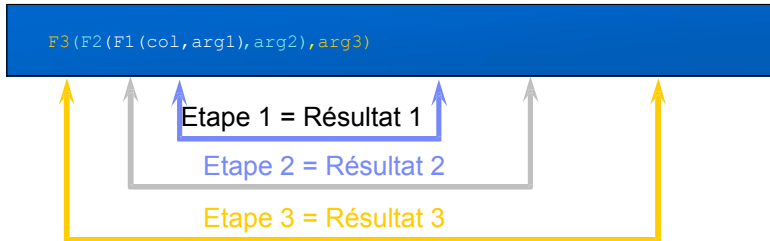
ENAME	SAL	COMM	(SAL*12)+NVL(COMM,0)
KING	5000		60000
BLAKE	2850		34200
CLARK	2450		29400
JONES	2975		35700
MARTIN	1250	1400	16400
ALLEN	1600	300	19500
...			

14 rows selected.



# Imbrication des Fonctions

- Le niveau d'imbrication des fonctions mono-ligne est illimité
- Les fonctions imbriquées sont évaluées de l'intérieur vers l'extérieur

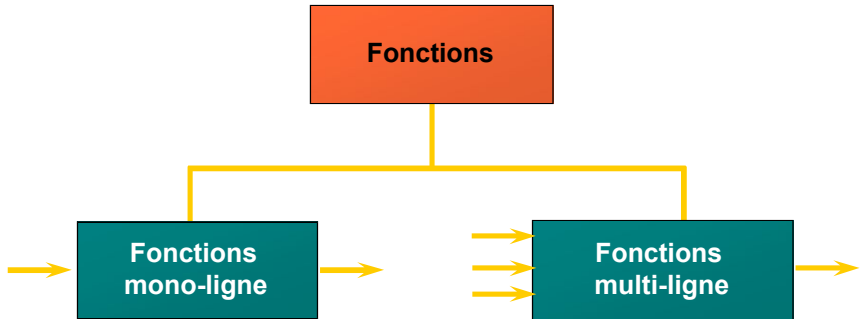


## Imbrication des Fonctions

```
SQL> SELECT  ename,  
2           NVL(TO_CHAR(mgr), 'No Manager')  
3 FROM      emp  
4 WHERE     mgr IS NULL;
```

ENAME	NVL(TO_CHAR(MGR), 'NOMANAGER')
-----	-----
KING	No Manager

# Fonctions SQL – Fonction groupe



**Les fonctions de groupe agissent sur plusieurs lignes pour produire une seule valeur par groupe**

## Les fonctions de groupe ou d'agrégation

- **Ces fonctions s'appliquent à une colonne, en général de type numérique**
  - COUNT qui compte le nombre de valeurs non nulles
  - MAX et MIN
  - AVG qui calcule la moyenne des valeurs de la colonne
  - SUM qui effectue le cumul
  - VARIANCE calcule la variance
  - STDDEV calcule l'écart type

### Remarques importantes:

- **Pour le type DATE et chaîne de caractère, on peut utiliser seulement les fonctions COUNT, MIN et MAX**
- **Toutes les fonctions, excepté COUNT, ignorent les valeurs NULL**

## Syntaxe de fonctions de groupe...

```
SELECT      [column,] group_function(column), ...  
FROM        table  
[WHERE      condition]  
[GROUP BY   column]  
[ORDER BY   column];
```

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%';
```

# Fonctions AVG et SUM

- **AVG et SUM s'utilisent avec des données numériques.**

```
SQL> SELECT AVG(sal), MAX(sal),  
2          MIN(sal), SUM(sal)  
3 FROM emp  
4 WHERE job LIKE 'SALES%';
```

AVG (SAL)	MAX (SAL)	MIN (SAL)	SUM (SAL)	
-----	-----	-----	-----	
1400	1600	1250	5600	

# Fonctions MIN et MAX

- MIN et MAX s'utilisent avec tous types de données.

```
SQL> SELECT MIN(hiredate), MAX(hiredate)  
2 FROM emp;
```

```
MIN(HIRED) MAX(HIRED)  
17-DEC-80 12-JAN-83
```

# Utilisation de la Fonction COUNT

- **COUNT(\*)** ramène le nombre de lignes d'une table.

```
SQL> SELECT COUNT(*)  
2 FROM emp  
3 WHERE deptno = 30;
```

```
COUNT (*)  
-----  
6
```



# Utilisation de la Fonction COUNT

- **COUNT(*expr*)** ramène le nombre de lignes non NULL.

```
SQL> SELECT COUNT(comm)
2 FROM emp
3 WHERE deptno = 30;
```

```
COUNT (COMM)
-----
4
```

# Fonctions de Groupe et Valeurs NULL

- Les fonctions de groupe ignorent les valeurs NULL des colonnes.

```
SQL> SELECT AVG(comm)  
2 FROM emp;
```

AVG (COMM)
550

# Création de Groupes de Données

EMP

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

2916.6667

**"salaire  
moyen pour  
chaque département  
de la table  
EMP"**

1566.6667

DEPTNO	AVG (SAL)
10	2916.6667
20	2175
30	1566.6667

# Création de Groupes de Données: la Clause GROUP BY

- Divisez une table en groupes de lignes avec la clause GROUP BY.

```
SELECT      column,  
FROM        group_function table  
[WHERE      condition]  
[GROUP BY   group_by_expression  
[ORDER BY   column];
```

# Utilisation de la Clause GROUP BY

- La clause **GROUP BY** doit inclure toutes les colonnes de la liste **SELECT** qui ne figurent pas dans des fonctions de groupe.

```
SQL> SELECT deptno, AVG(sal)
2 FROM emp
3 GROUP BY deptno;
```

DEPTNO	AVG(SAL)
10	2916.6667
20	2175
30	1566.6667

# Utilisation de la Clause GROUP BY

- La colonne citée en **GROUP BY** ne doit pas nécessairement figurer dans la liste **SELECT**.

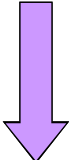
```
SQL> SELECT  AVG(sal)
      2  FROM    emp
      3  GROUP BY deptno;
```

```
AVG (SAL)
-----
2916.6667
2175
1566.6667
```

## Clause de groupage : GROUP BY

```
SELECT idSer, AVG(salaire), COUNT(*)  
FROM emp  
GROUP BY idSer;
```

100	Michel	20	
2000			
200		10	
Sylvie		3000	
300		20	
Bernard		1000	
400		10	
Claude		2000	
500		10	
Thomas		1000	



idSer	AVG(salaire)	COUNT(*)
10	2000	3
20	1500	2

# Regroupement sur Plusieurs Colonnes

EMP

DEPTNO	JOB	SAL
10	MANAGER	2450
10	PRESIDENT	5000
10	CLERK	1300
20	CLERK	800
20	CLERK	1100
20	ANALYST	3000
20	ANALYST	3000
20	MANAGER	2975
30	SALESMAN	1600
30	MANAGER	2850
30	SALESMAN	1250
30	CLERK	950
30	SALESMAN	1500
30	SALESMAN	1250

**"somme des salaires de la table EMP pour chaque poste, regroupés par département"**

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600



## Utilisation de la Clause GROUP BY sur Plusieurs Colonnes

```
SQL> SELECT deptno, job, sum(sal)
2 FROM emp
3 GROUP BY deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
...		

9 rows selected.

# Erreurs d'Utilisation des Fonctions de Groupe dans une Requête

- **Toute colonne ou expression de la liste SELECT autre qu'une fonction de groupe, doit être incluse dans la clause GROUP BY.**

```
SQL> SELECT deptno, COUNT(ename)
      2 FROM emp;
```

```
SELECT deptno, COUNT(ename)
      *
ERROR at line 1:
ORA-00937: not a single-group group
function
```

Colonne manquante dans la clause GROUP BY

# Erreurs d'utilisation des Fonctions de Groupe dans une Requête

- Vous ne pouvez utiliser la clause WHERE pour limiter les groupes.
- Utilisez la clause HAVING.

```
SQL> SELECT      deptno, AVG(sal)
2  FROM          emp
3  WHERE         AVG(sal) > 2000
4  GROUP BY     deptno;
```

```
WHERE AVG(sal) > 2000
```

```
*
```

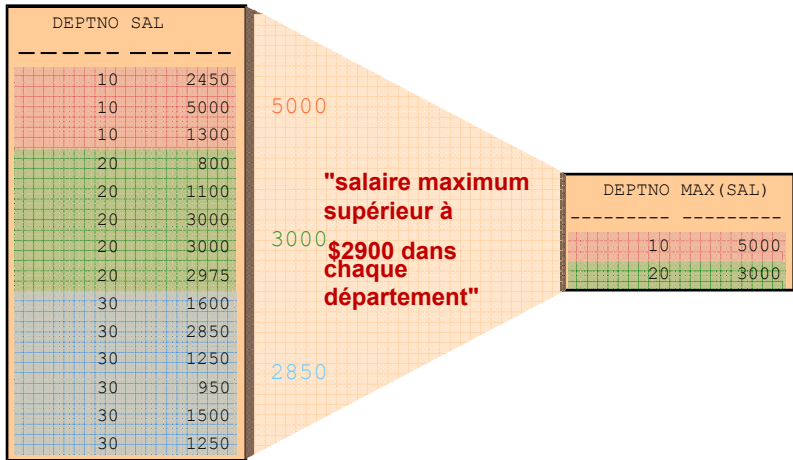
```
ERROR at line 3:
```

```
ORA-00934: group function is not allowed
```

```
here
```

# Exclusion de Groupes

EMP



## Exclusion de Groupes : la Clause HAVING

- **Utilisez la clause HAVING pour restreindre les groupes**
  - Les lignes sont regroupées.
  - La fonction de groupe est appliquée.
  - Les groupes qui correspondent à la clause HAVING sont affichés.

```
SELECT      column,  
FROM        group_function table  
[WHERE      condition]  
[GROUP BY   group_by_expression  
[HAVING     group_condition]  
[ORDER BY   ] column];
```

## Utilisation de la clause HAVING

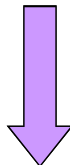
```
SQL> SELECT    deptno, max(sal)
  2  FROM      emp
  3  GROUP BY  deptno
  4  HAVING
      max(sal)>2900;
```

DEPTNO	MAX (SAL)
10	5000
20	3000

# Restriction sur les groupages : HAVING

```
SELECT idSer, AVG(salaire), COUNT(*)
FROM emp
GROUP BY idSer
HAVING COUNT(*) > 2;
```

100	Michel	20	
200		10	
300	Sylvie	3000	
400	Bernard	1000	
500	Claude	2000	
	Thomas	1000	



idSer	AVG(salaire)	COUNT(*)
----	)	)
- 10	-----	-----
-		- 3

2000

## Utilisation de la Clause HAVING

```
SQL> SELECT      job, SUM(sal) PAYROLL
  2  FROM          emp
  3  WHERE         job NOT LIKE 'SALES%'
  4  GROUP BY     job
  5  HAVING        SUM(sal)>5000
  6  ORDER BY     SUM(sal);
```

JOB	PAYROLL
ANALYST	6000
MANAGER	8275



# Récapitulatif

• La syntaxe SQL fournit donc :

**selon**

- Le moyen de partitionner une relation en groupes certains critères.
- Le moyen d'exprimer des conditions sur ces groupes
- Des fonctions de groupe

```
SELECT      column,  
FROM        group_function table  
[WHERE      condition]  
[GROUP BY   group_by_expression  
[HAVING     group_condition]  
[ORDER BY   ] group_condition  
            column];
```

**Questions??**

