# SQL

## LDD

### TABLE

**Création de table :**

- create table SGBD ( , , , ) ;
- create table ldd as select * from SGBD where (condition);

#### LES CONTRAINTES

- Il existe 5 types de contraintes :
  - Not null
  - Unique
  - Check
  - Primary key
  - Foreign key
- Il existe 2 niveaux de définitions :
  - Contrainte au niveau colonne
  - Contrainte au niveau table
- Primary key: constraint pk_const primary key
- Foreign key: constraint fk_const foreign key (colonne) references table (colonne)

**Modification de table :**

- alter table SGBD
  - Add : ajouter une colonne
  - Modify : modifier une colonne
  - Drop : supprimer une colonne
  - Add constraint
  - Drop constraint
  - Enable : activer une contrainte
  - Disable : désactiver une contrainte

**Suppression de table :**

- drop table SGBD ;
- truncate table SGBD ;

**Renommer table :**

- rename ancien to nouveau ;

### VUE

- Est une table virtuelle qui ne contient pas des donnés

**Création et modification de vue :**

- create or replace view v_sgbd (alias) as select col1,col2 from table where (condition) with
  - with check option
  - with read only
  - constraint

**Suppression de vue :**

- drop view v_sgbd;

### SÉQUENCE

- Générer une valeur de clé primaire

**Création de séquence :**

- create sequence seq_sgbd
  start with val
  increment by pas
  maxvalue val_max
  minvalue val_min
  cycle
  cache val ;

**Modification de séquence :**

- alter sequence seq_sgbd
  start with val
  increment by pas
  maxvalue val_max
  minvalue val_min
  cycle
  cache val ;

**Utilisation de séquence :**

- Se fait par des pseudo-colonnes :
  - Nextval : incrémente la séquence et retourne la nouvelle valeur
  - Currval : retourne la valeur courante de la séquence
- insert into table values (seq_sgbd.nextval, 'y') ;

**Suppression de séquence :**

- drop sequence seq_sgbd ;

### INDEX

- Est un objet qui permet d'accélérer la recherche des lignes
- Il contient 2 champs :
  - Clé d'index
  - @ du bloc contenant la clé
- Créez index si :
- ✓ Colonne contient un grand nbr de valeurs NULL
- ✓ Condition de jointure
- ✓ Utilisée souvent dans la Clause WHERE
- ✓ Table de grande taille

**Création d'index :**

- create unique index idx_sgbd on SGBD (col1,col2) ;
- alter index ancient rename to nouveau;

**Suppression d'index :**

- drop index idx_sgbd ;

## SYNONYME

- Est un alias, objet qui peut être une table, une vue, une séquence, une procédure, une fonction, un package, etc
- Masquer le vrai nom des objets
- Simplifier les noms des objets

**Création et suppression de synonyme :**

- create public synonym sy_sgbd for nom_objet;
- drop synonym sy_sgbd;

# LMD

**Insertion des lignes :**

- ✓ **Insertion implicite:** insert into table(col1,col2) values(val1,val2);
- ✓ **Insertion explicite:** insert into table values(val1,val2);
- ✓ **Insertion à partir d'autres tables:** insert into table(col1,col2) select att1,att2 from table where condition;

**Mise à jour de lignes :**

- ✓ update table set att1=val where condition;
- ✓ update table set (col) = (select att from table where condition) where condition;
- ✓ update table t set alias = (select att from table1 t1 where t.att=t1.att);

**Suppression des lignes :**

- ✓ delete from table ;
- ✓ delete from table where condition;
- ✓ delete from table where att= (select att from tab where condition);

# LID

## INSTRUCTION SELECT

- ✓ select * from table;
- ✓ select col1, col2 from table;
- ✓ select distinct col from table;

## ALIAS DE COLONNE

- ✓ select last_name nom, first_name as prénom, salary*12 "revenue annuel" from employes;

## CONCATÉNATION

- ✓ representee par le symbole ||
- ✓ select dep_id||'**'||dep_name as "dep" from departments;

## OPÉRATEURS DE COMPARAISON

***BETWEEN***

- ✓ SELECT COL FROM TABLE WHERE COL BETWEEN .. AND ..;

***IN***

- ✓ SELECT COL FROM TABLE WHERE COL IN(,);

***LIKE***

- ✓ SELECT COL FROM TABLE WHERE COL LIKE 'Y_R%';

***IS NULL***

- ✓ SELECT COL FROM TABLE WHERE COL IS NULL;

## OPERATEURS LOGIQUES

***AND***

- ✓ SELECT COL1,COL2 FROM TABLE WHERE COL2>=VAL AND COL1 LIKE '%SGBD%';

***OR***

- ✓ SELECT COL1,COL2 FROM TABLE WHERE COL2>=VAL OR COL1 LIKE '%SGBD%';

***NOT***

- ✓ SELECT COL FROM TABLE WHERE COL NOT IN(' ',' ',' ');

➔**LES PARENTHESES > OPÉRATEURS DE COMPARAISON > NOT > AND > OR**

## CLAUSE ORDER BY

- ✓ Tri des lignes ASC ou DESC
- ✓ La dernière clause dans SELECT
- ✓ select col from table order by col DESC;
- ✓ select col1,col2 from table order by col1, col2;

## FONCTIONS DE CARACTÈRES

| | |
|---|---|
| **Lower(ch)** | Maj→min |
| **Upper(ch)** | Min→maj |
| **Initcap(ch)** | 1er caractere en maj |
| **Concat(ch1,ch2)** | Concaténation |

| | |
|---|---|
| **Substr(ch,pos,long)** | Extraction |
| **Length(ch)** | Taille |
| **Instr(ch1,ch2)** | Pos de ch2 ds ch1 |
| **Lpad(ch,long,car)** | Complète ch par car à gauche |
| **Rpad(ch,long,car)** | Complète ch par car à droite |
| **Ltrim(ch,car)** | Supprime car à gauche |
| **Rtrim(ch,car)** | Supprime car à droite |
| **Replace(ch,ch1,ch2)** | Remplace ch1 par ch2 dans ch |
| **Ascii(ch)** | Retourne le code ascii |
| **Chr(n)** | Retourne le car ayant le code ascii n |

## FONCTIONS NUMERIQUES

| **ABS(n)** | **Valeur absolue** |
|---|---|
| **MOD(n1,n2)** | Reste |
| **POWER(n,e)** | Puissance |
| **SQRT(n)** | Racine carrée |
| **ROUND(n,p)** | Arrondissement |
| **TRUNC(n,p)** | Tronque n à la pos p |
| **FLOOR(n)** | Partie entière inferieure |
| **CEIL(n)** | Partie entière supérieure |
| **GREATEST(n,n1,n2)** | Max |
| **LEAST(n,n1,n2)** | Min |

## FONCTIONS DE CONVERSION

### *TO_CHAR*

- ✓ SELECT TO_CHAR(COL,'FM DD MONTH YYYY') AS DATE FROM TABLE ;

### *TO_DATE*

- ✓ SELECT HIRE_DATE FROM TAB WHERE HIRE_DATE>TO_DATE('01/01/1982','DD-MM-YYYY');

### *TO_NUMBER*

- ✓ SELECT COL FROM TABLE WHERE COL>= TO_NUMBER('7777');

## FONCTIONS ANALYTIQUES

### *ROW_NUMBER*

- ✓ SELECT COL1,COL2, ROW_NUMBER() OVER (ORDER BY EXPRESSION [ASC|DESC]) "NUMERO" FROM TABLE;

### *RANK*

- ✓ SELECT NOM, SCORE, RANK() OVER (PARTITION BY EXPRESSION ORDER BY EXPRESSION [ASC|DESC]) AS CLASSEMENT FROM SCORES;

### *DENSE_RANK*

- ✓ SELECT NOM, SCORE, DENSE_RANK() OVER (PARTITION BY EXPRESSION ORDER BY EXPRESSION [ASC|DESC]) AS CLASSEMENT FROM SCORES;

### *FIRST_VALUE*

- ✓ SELECT PRODUIT, DATE, MONTANT, FIRST_VALUE(MONTANT) OVER (PARTITION BY PRODUIT ORDER BY DATE ASC) AS PREMIERMONTANT FROM VENTES;

### *LAST_VALUE*

- ✓ SELECT PRODUIT, DATE, MONTANT, LAST_VALUE(MONTANT) OVER (PARTITION BY PRODUIT ORDER BY DATE ASC) AS DERNIERMONTANT FROM VENTES;

## FONCTIONS MULTI-LIGNES

### *AVG*

- ✓ SELECT AVG(MONTANT) AS MOYENNEVENTES FROM VENTES;

### *COUNT*

- ✓ SELECT COUNT(*) FROM TABLE WHERE CONDITION;
- ✓ SELECT COUNT(NOM) AS NOMBRECLIENTSAVECNOM FROM CLIENTS;

### *MAX*

- ✓ SELECT MAX(PRIX) AS PRIXMAXIMUM FROM PRODUITS;

### *MIN*

- ✓ SELECT MIN(PRIX) AS PRIXMINIMUM FROM PRODUITS;

### *STDDEV*

- ✓ SELECT STDDEV(MONTANT) AS ECARTTYPEVENTES FROM VENTES;

### *SUM*

- ✓ SELECT SUM(MONTANT) AS TOTALVENTES FROM VENTES;

### *VARIANCE*

- ✓ SELECT VARIANCE(MONTANT) AS VARIANCEVENTES FROM VENTES;

## LES JOINTURES

### *JOINTURE INTERNE*

- ✓ SELECT TAB1.A, TAB1.B, TAB2.A,TAB2.B FROM TAB1 INNER JOIN TAB2 ON TAB1.A=TAB2.A AND TAB1.B=TAB2.B;

### *JOINTURE EXTERNE*

- ✓ SELECT TAB1.A, TAB1.B, TAB2.A,TAB2.B FROM TAB1 LEFT OUTER JOIN TAB2 ON TAB1.A=TAB2.A AND TAB1.B=TAB2.B;
- ✓ SELECT TAB1.A, TAB1.B, TAB2.A,TAB2.B FROM TAB1 RIGHT OUTER JOIN TAB2 ON TAB1.A=TAB2.A AND TAB1.B=TAB2.B;
- ✓ SELECT TAB1.A, TAB1.B, TAB2.A,TAB2.B FROM TAB1 FULL OUTER JOIN TAB2 ON TAB1.A=TAB2.A AND TAB1.B=TAB2.B;

### *EQUIJOINTURE*

- ✓ SELECT TAB1.A, TAB1.B, TAB2.A,TAB2.B FROM TAB1 LEFT OUTER JOIN TAB2 ON TAB1.A=TAB2.A;

### *NON-EQUIJOINTURE*

- ✓ SELECT TAB1.A, TAB2.A, TAB2.C FROM TAB1 INNER JOIN TAB2 ON TAB1.A>=TAB2.A AND TAB1.A<=TAB2.C;

### *AUTO-JOINTURE*

- ✓ SELECT T1.A, T2.B, T2.C FROM TAB1 T1 INNER JOIN TAB1 T2 ON T2.C=T1.A

### *JOINTURE NATURELLE*

- ✓ SELECT DEPARTMENT ID, DEPARTMENT NAME, LOCATION ID, CITY FROM DEPARTMENTS NATURAL JOIN LOCATIONS WHERE LOCATION ID <>1700;

# PLSQL

## BLOCS ANONYMES

```
Declare – optionnelle
/* déclaration var, cte, types, curseurs
Integer/number := val ;
Varchar2() := ‘ ’ ;
Date;
Dec%type ;
Dec%rowtype ; */
Begin
/*traitements*/
Exception – optionnelle
End ;
/
```

### Déclaration et initialisation des variables :

- ✓ v_date date default ’01-01-2009’ ;
- ✓ v_cte constant := 7 ;
- ✓ v_char char(5) notnull := ‘SGBD’ ;
- ❖ dbms_output.put_line(‘ ‘ || val);

### Structures de contrôle :

```
If condition then
Elsif condition then
Else
End if ;
```

```
Case v1
When cond then
When cond then
When cond then
End case;
```

```
while condition loop
End loop;
```

```
Loop
Exit when condition
End loop;
```

```
For I in inf..sup Loop
End loop;
```

```
Select * intro v_sgbd from SGBD where condition;
```

## LES CURSEURS

- ❖ Variable qui pointe vers le résultat d’une requête

### Manipulation des curseurs :

#### *DECLARATION*

- ✓ CURSOR CUR_SGBD IS SELECT * FROM SGBD WHERE CONDITION ORDER BY COL;

#### *OUVERTURE*

- ✓ OPEN CUR_SGBD :

#### *EXECUTION*

- ✓ FETCH CUR_SGBD INTO VARS;
  WHILE/EXIT WHEN CUR_SGBD%ATTS;

***FERMETURE***

- ✓ CLOSE CUR_SGBD ;

**Attributs des curseurs :**

- %ISOPEN : curseur ouvert
- %FOUND : fetch réussi
- %NOTFOUND : fetch échoué
- %ROWCOUNT : renvoie nbr de lignes contenues

**Utilisation des curseurs :**

- ✓

```
Declare
Cursor cur_sgbd is select … ;
Rec_sgbd cur_sgbd%rowtype;
Begin
Open cur_sgbd;
Loop
Fetch cur_sgbd into rec_sgbd;
…
End;
```

- ✓

```
Declare
Cursor cur_sgbd is select … ;
Begin
For rec_sgbd in cur_sgbd
Loop
…
End;
```

**Curseurs paramétrés :**

```
Declare
Cursor cur_sgbd(param type,param1 type) is select …;
```

## PROCEDURES ET FONCTIONS

### STOCKÉES

**Procédures :**

- ❖ create or replace procedure nom (arg1 IN type, arg2 OUT type) IS
  begin … end;

**Fonctions :**

- ❖ create or replace function fn_nom(arg1 type,arg2 type)
  return type IS
  declaration;
  begin
  return ;
  end;

### NON STOCKÉES

**Procédures :**

- ❖ DECLARE …
  PROCEDURE myproc(a in type, b in type, prod out type) IS BEGIN prod:=a*b;
  END myproc;
  BEGIN myproc(a,b,p);
  dbms output.put line(Le produit: '||s); END;

**Fonctions :**

- ❖ DECLARE
  a type := val;
  b type := val;
  c type;
  FUNCTION myfunc(a type, b type) RETURN type IS
  prod type;
  BEGIN
  prod:=a*b;
  return prod;
  END;
  BEGIN c:=myfunc(a,b);
  dbms output.put line('Le produit: '||c); END;

## LES TRIGGERS

- ✓ Se déclenche automatiquement lorsqu'un événement se produit (insert,update,delete)

### DECLENCHEUR DE TABLE

```
Create trigger trig_sgbd
After / before insert or update or delete on table
Begin
If inserting then
…
Elsif updating then
…
Else
…
End if;
End;
```

```
Create or replace trigger trig_sgbd
After / before insert or update of att1,att2
or delete on table
Begin
Case
when inserting then
…
when updating('att1') then
…
when deleting then
…
End case;
End;
```

## DECLENCHEUR DE LIGNE

```
CREATE OR REPLACE TRIGGER trig dep
BEFORE INSERT OR UPDATE OR DELETE ON table
FOR EACH ROW
BEGIN
IF INSERTING THEN
dbms output.put line('Insertion');
END IF;
IF UPDATING THEN
dbms output.put line('Modification');
END IF;
IF DELETING THEN
dbms output.put line('Suppression');
END IF;
END;
```

```
CREATE OR REPLACE TRIGGER trig_insert
BEFORE INSERT ON table
FOR EACH ROW
WHEN (new.col < val)
BEGIN
:new.col := :new.col+10;
END;
```

# LES EXCEPTIONS

## LES EXCEPTIONS PREDEFINIES

- CREATE OR REPLACE PROCEDURE nom (arg type)
  IS
  var type ;
  BEGIN
  SELECT arg INTO var FROM table
  WHERE condition;
  …
  EXCEPTION WHEN TOO MANY ROWS –THEN
  DBMS OUTPUT.PUT LINE ('La requˆete revoie plusieurs lignes, utilisez un curseur');
  WHEN OTHERS THEN
  ROLLBACK;
  DBMS OUTPUT.PUT LINE('code de l"erreur : '||SQLCODE);
  DBMS OUTPUT.PUT LINE('message de l"erreur : '||SQLERRM); END;
- NO_DATA_FOUND
- ZERO_DIVIDE
- DUP_VAL_ON_INDEX
- INVALID_CURSOR
- CURSOR_ALREADY_OPEN
- INVALID_NUMBER
- ROWTYPE_MISMATCH

**Redéclaration des exceptions prédéfinies :**

- ✓ DECLARE a type;
  b type;
  nom EXCEPTION;
  PRAGMA EXCEPTION_INIT(nom,code); BEGIN
  …
  EXCEPTION WHEN nom THEN
  …
  END;

## LES EXCEPTIONS DEFINIES

**Avec le mot clé RAISE :**

```
CREATE OR REPLACE PROCEDURE pr_check_salary (current_salary number) IS
   salary_too_high EXCEPTION;   – déclarer l'exception
   PRAGMA EXCEPTION_INIT(salary_too_high,-20100); – associer optionellement le code -20100 à l'exception
   max_salary NUMBER := 10000;
BEGIN
   IF current_salary > max_salary THEN
   RAISE salary_too_high;   – déclencher l'exception
   END IF;
   EXCEPTION
      WHEN salary_too_high THEN   – gérer l'exception
      DBMS_OUTPUT.PUT_LINE('ERREUR : '||SQLCODE);
      DBMS_OUTPUT.PUT_LINE('MESSAGE : '||SQLERRM);
      DBMS_OUTPUT.PUT_LINE('Salaire '||current_salary||' est très élevé');
      DBMS_OUTPUT.PUT_LINE ('Le salaire maximum est ' || max_salary);
END;
```

**Avec la procédure RAISE_APPLICATION_ERROR :**

```
DECLARE
   mgr_id NUMBER;
BEGIN
   SELECT manager_id INTO mgr_id FROM employees WHERE employee_id=300;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20999, 'Numéro d"employé non valide');
END;
```