

# LANGUAGE SQL:LDD

# 1. QU'EST CE QU'UN SGBDR ?

Un (SGBDR) est un Système de Gestion de Base de Données Relationnel  
ce qui lui confère une très grande capacité à gérer les données tout  
en conservant leur intégrité et leur cohérence.

Un SGBD est chargé de :

- Stocker les données
- Vérifier les contraintes d'intégrité définies,
- Garantir la cohérence des données qu'il stocke, même en cas de panne  
arrêt brutal) du système,
- Assurer les relations entre les données définies par les utilisateurs.

# OBJECTIFS

- ⦿ SQL (structured Query Language) est le langage standard (et donc normalisé) d'interrogation et de mise à jour de tous les **SGBDRs**. Plus précisément il permet de créer des données contenues dans des tables, mais aussi de les consulter / interroger, modifier, supprimer.
- ⦿ Il utilise des opérateurs simples comme :
  - \* les opérateurs de projection (choix de colonnes),
  - \* les opérateurs de restrictions ou de selection (choix de lignes),
  - \* les opérateurs de jointures (mise en relations de 2 ou plusieurs tables)
  - \* les opérateurs ensemblistes (Union, intersection, différence).

# OBJECTIFS

Langage de description de donnée (LDD):

- ⦿ Création d'une structure de données (create)
- ⦿ Modification de la structure d'un objet de la base (Alter)
- ⦿ Suppression d'une structure de données(Drop )

# DÉFINITION

**DDL** = Data Definition Language (Language des définitions de données LDD)

Permet principalement d'agir sur la structure des données de la base de données

- tablespace
- tables
- index
- trigger
- séquences

**Attention :**

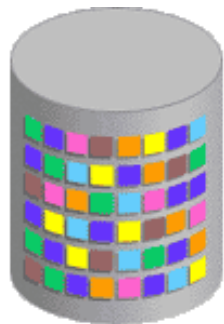
Ces commandes ne permettent pas un retour arrière (ROLLBACK)

# INSTRUCTION CREATE TABLE

```
CREATE TABLE [schema.] Table  
    (column type [DEFAULT expr], ...);
```

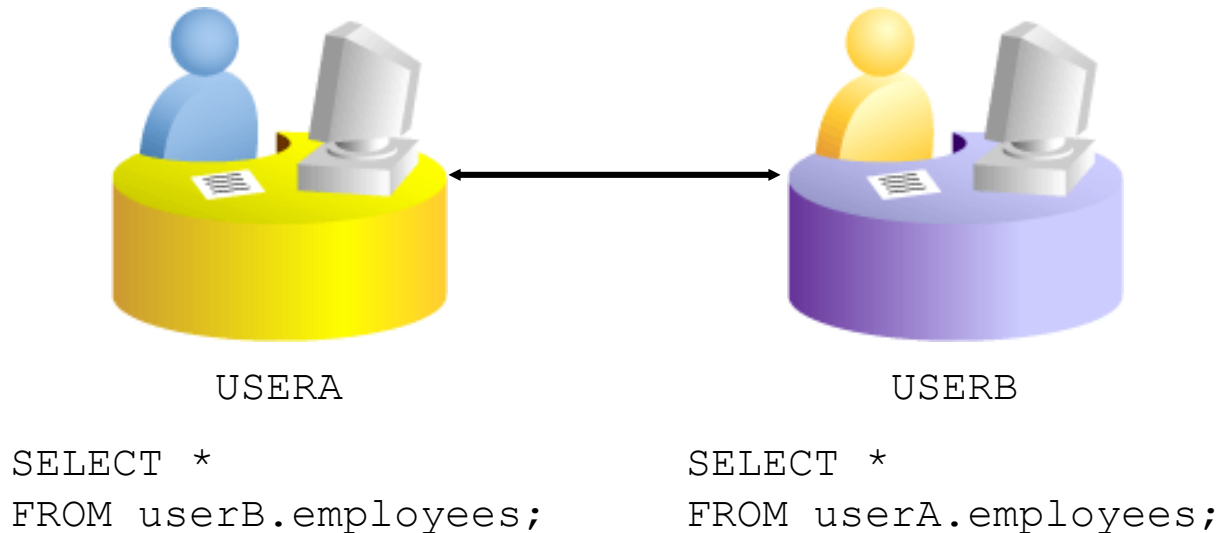
- Indiquez:

- Le nom de la table
- Le nom de la colonne, son type de données et sa taille.



# RÉFÉRENCER LES TABLES D'UN AUTRE UTILISATEUR

- Les tables appartenant à d'autres utilisateurs ne résident pas dans le schéma de l'utilisateur.
- Vous devez utiliser le nom du propriétaire comme préfixe pour ces tables.



# OPTION DEFAULT

- Indiquez une valeur par défaut pour une colonne lors de l'insertion.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Le type de données par défaut doit correspondre au type de colonne.

```
CREATE TABLE hire_dates  
  (id          NUMBER(8),  
   hire_date DATE DEFAULT SYSDATE);  
Table created.
```



# CRÉER DES TABLES

- Créez la table département

```
CREATE TABLE dept
  (deptno      NUMBER(2),
   dname       VARCHAR2(14),
   loc         VARCHAR2(13),
   create_date DATE DEFAULT SYSDATE);
```

Table created.

- Vérifiez la création de la table

```
DESCRIBE dept
```

# CRÉER DES TABLES À PARTIR D'AUTRES TABLES

```
CREATE TABLE [schema.]table  
(colonne type [default expr], ..... ) AS subquery
```

## Exemple

```
create table emp1 as select * from employees where  
    department_id=20  
select * from emp1;
```

# TYPES DE DONNÉES

Data Type	Description
<b>VARCHAR2</b> ( <i>size</i> )	Données de type caractère de longueur variable
<b>CHAR</b> ( <i>size</i> )	Données de type caractère de longueur fixe
<b>NUMBER</b> ( <i>p</i> , <i>s</i> )	Données numériques de longueur variable
<b>DATE</b>	Valeur de date et d'heure
<b>LONG</b>	Données de type caractère de longueur variable (jusqu'à 2 GB)
	.....

# INCLURE DES CONTRAINTES

- ◉ Les contraintes appliquent des règles au niveau table.
- ◉ Les contraintes empêchent la suppression d'une table s'il existe des dépendances.
- ◉ Les types de contrainte suivants sont pris en charge:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

# RÈGLES RELATIVES AUX CONTRAINTES

- ◉ Vous pouvez créer une contrainte à différents instants:
  - Lors de la création de la table
  - Après la création de la table
- ◉ Définissez une contrainte au niveau colonne ou au niveau table.

# DÉFINIR DES CONTRAINTES

- Syntaxe

```
CREATE TABLE [schema.] table
    (column datatype [DEFAULT expr]
    [column_constraint],
    ...
    [table_constraint] [,...]);
```

- Contrainte au niveau colonne

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Contrainte au niveau table

```
column, ...
[CONSTRAINT constraint_name] constraint_type
(column, ...),
```

# DÉFINIR DES CONTRAINTES

- Contrainte au niveau colonne:

```
CREATE TABLE employees(  
    employee_id  NUMBER(6)  
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,  
    first_name   VARCHAR2(20),  
    ...);
```

1

- Contrainte au niveau table

```
CREATE TABLE employees(  
    employee_id  NUMBER(6),  
    first_name   VARCHAR2(20),  
    ...  
    job_id       VARCHAR2(10) NOT NULL,  
    CONSTRAINT emp_emp_id_pk  
        PRIMARY KEY (EMPLOYEE_ID));
```

2

# CONTRAINTE NOT NULL

- Garantit que les valeurs NULL ne sont pas autorisées pour la colonne :

EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...

20 rows selected.



Contrainte NOT NULL (aucune ligne ne peut contenir de valeur NULL pour cette colonne)



Contrainte NOT NULL



Absence de NOT NULL contrainte (n'importe quelle ligne peut contenir une valeur NULL pour cette colonne)



# CONTRAINTE UNIQUE

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL
100	King	SKING
101	Kochhar	NKOCHHAR
102	De Haan	LDEHAAN
103	Hunold	AHUNOLD
104	Ernst	BERNST

...



INSERT INTO

208	Smith	JSMITH
209	Smith	JSMITH

← Autorisé

← Non autorisé :  
existe déjà

Contrainte UNIQUE

# CONTRAİNTE UNIQUE


- Définie au niveau table ou au niveau colonne

```
CREATE TABLE employees (  
    employee_id      NUMBER(6),  
    last_name        VARCHAR2(25) NOT NULL,  
    email            VARCHAR2(25),  
    salary           NUMBER(8,2),  
    commission_pct   NUMBER(2,2),  
    hire_date        DATE NOT NULL,  
    ...  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

# CONTRAINTE PRIMARY KEY

DEPARTMENTS

PRIMARY KEY



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

...

Non autorisé  
(valeur null)

↑ INSERT INTO



	Public Accounting		1400
50	Finance	124	1500

Non autorisé  
(50 existe déjà)

# CONTRAİNTE FOREIGN KEY

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

PRIMARY  
KEY



...

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60

FOREIGN  
KEY



INSERT INTO

200	Ford	9
201	Ford	60

Non autorisé  
(9 n'existe pas)



Autorisé



# CONTRAİNTE FOREIGN KEY

- Définie au niveau table ou au niveau colonne :

```
CREATE TABLE employees(  
    employee_id      NUMBER(6),  
    last_name        VARCHAR2(25) NOT NULL,  
    email            VARCHAR2(25),  
    salary           NUMBER(8,2),  
    commission_pct   NUMBER(2,2),  
    hire_date        DATE NOT NULL,  
    ...  
    department_id    NUMBER(4),  
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)  
        REFERENCES departments(department_id),  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

# CONTRAÎNTE FOREIGN KEY : MOTS CLÉS : ON DELETE

- ◉ FOREIGN KEY : définit la colonne dans la table enfant au niveau contrainte de table.
- ◉ REFERENCES : identifie la table et la colonne dans la table parent.
- ◉ ON DELETE CASCADE : supprime les lignes dépendantes dans la table enfant lorsqu'une ligne de la table parent est supprimée.
- ◉ ON DELETE SET NULL: convertit les valeurs des clés étrangères dépendantes en valeurs NULL. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL.
- ◉ SET DEFAULT place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé
- ◉ RESTRICT indique une erreur en cas d'effacement d'une valeur correspondant à la clé

# CONTRAİNTE CHECK

- Définit une condition à laquelle chaque ligne doit répondre.
- Les expressions suivantes ne sont pas autorisées:
  - Appels de fonction telle que SYSDATE
  - Interrogations qui font référence à d'autres valeurs dans d'autres lignes.

```
..., salary  NUMBER(2)  
    CONSTRAINT emp_salary_min  
        CHECK (salary > 0),...
```

# CREATE TABLE (1)

```
CREATE TABLE employees1
( employee_id      NUMBER(6)
  CONSTRAINT emp_employee_id1 PRIMARY KEY,
  first_name       VARCHAR2(20),
  last_name        VARCHAR2(25)
  CONSTRAINT emp_last_name_nn1 NOT NULL,
  email            VARCHAR2(25)
  CONSTRAINT emp_email_nn1    NOT NULL
  CONSTRAINT emp_email_uk1    UNIQUE,
  phone_number     VARCHAR2(20),
  hire_date        DATE
  CONSTRAINT emp_hire_date_nn1 NOT NULL,
  job_id           VARCHAR2(10)
  CONSTRAINT emp_job_nn1      NOT NULL,
  salary           NUMBER(8,2)
  CONSTRAINT emp_salary_ck1   CHECK (salary>0),
  commission_pct   NUMBER(2,2),
  manager_id       NUMBER(6),
  department_id    NUMBER(4)
  CONSTRAINT emp_dept_fk1     foreign KEY
REFERENCES dept (deptno));
```



# CREATE TABLE (2)

```
CREATE TABLE employees2
( employee_id      NUMBER(6),
  first_name       VARCHAR2(20),
  last_name        VARCHAR2(25) NOT NULL ,
  email            VARCHAR2(25) NOT NULL ,
  phone_number     VARCHAR2(20),
  hire_date        DATE NOT NULL ,
  job_id           VARCHAR2(10) NOT NULL ,
  salary           NUMBER(8,2),
  commission_pct   NUMBER(2,2),
  manager_id       NUMBER(6),
  department_id    NUMBER(4),
CONSTRAINT emp_employee_id2 PRIMARY KEY (employee_id),
CONSTRAINT emp_email_uk2    UNIQUE (email),
CONSTRAINT emp_salary_ck2   CHECK (salary>0),
CONSTRAINT emp_dept_fk2     FOREIGN KEY (department_id)
REFERENCES dept (deptno));
```

# CREATE TABLE (3)

## Table départements

```
CREATE TABLE dept
    (deptno      NUMBER(2),
     dname       VARCHAR2(14),
     loc         VARCHAR2(13),
     create_date DATE DEFAULT SYSDATE);
```

# L'INSTRUCTION ALTER TABLE

◉ Utilisez l'instruction ALTER TABLE pour:

- Ajouter une colonne
- Modifier une colonne existante
- Supprimer une colonne
- Ajouter une contrainte
- Supprimer une contrainte

# L'INSTRUCTION ALTER TABLE

- Utilisez l'instruction ALTER TABLE pour ajouter, modifier ou supprimer des colonnes.

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
MODIFY       (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
DROP         (column);
```

# AJOUTER UNE COLONNE

- Utilisez la clause ADD pour ajouter des colonnes.

```
ALTER TABLE employees  
ADD      (job_id VARCHAR2(9));  
Table altered.
```

- La nouvelle colonne devient la dernière colonne.

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
145	Russell	14000	01-OCT-96	
146	Partners	13500	05-JAN-97	
147	Errazuriz	12000	10-MAR-97	
148	Cambrault	11000	15-OCT-99	
149	Zlotkey	10500	29-JAN-00	

...

# MODIFIER UNE COLONNE

- Vous pouvez changer le type de données, la taille et la valeur par défaut d'une colonne.

```
ALTER TABLE employees  
MODIFY      (last_name VARCHAR2(30));  
Table altered.
```

- Une modification de la valeur par défaut affecte uniquement les insertions suivantes dans la table.

# SUPPRIMER UNE COLONNE

- Utilisez la clause DROP COLUMN pour supprimer les colonnes de la table dont vous n'avez plus besoin.

```
ALTER TABLE employees  
DROP COLUMN job_id;  
Table altered.
```

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
145	Russell	14000	01-OCT-96
146	Partners	13500	05-JAN-97
147	Errazuriz	12000	10-MAR-97
148	Cambrault	11000	15-OCT-99
149	Zlotkey	10500	29-JAN-00

# AJOUTER UNE CONTRAINTE

- ◉ Utilisez l'instruction ALTER TABLE pour:
  - Ajouter ou supprimer une contrainte, sans pour autant modifier sa structure.
  - Activer ou désactiver des contraintes
  - Ajouter une contrainte NOT NULL à l'aide de la clause MODIFY.

```
ALTER TABLE <table_name>  
ADD [CONSTRAINT <constraint_name>]  
type (<column_name>);
```



# AJOUTER UNE CONTRAINTE

- Ajouter une contrainte FOREIGN KEY à la table EMPLOYEES en indiquant qu'un manager doit déjà exister en tant qu'employé valide dans la table EMPLOYEES.

```
ALTER TABLE employees  
modify employee_id Primary Key;  
Table altered.
```

```
ALTER TABLE employees  
ADD CONSTRAINT emp_mgr_fk  
    FOREIGN KEY(manager_id)  
    REFERENCES employees(employee_id);  
Table altered.
```

# SUPPRIMER UNE CONTRAINTE

- Supprimer la contrainte manager de la table EMP

```
ALTER TABLE empLOYEES  
DROP CONSTRAINT emp_mgr_fk;  
Table altered.
```

- Supprimer la contrainte PRIMARY KEY sur la table DEPT et supprimer la contrainte FOREIGN KEY associée sur la colonne EMPLOYEES.DEPARTMENT\_ID

```
ALTER TABLE dept  
DROP PRIMARY KEY CASCADE;  
Table altered.
```

# L'INSTRUCTION DROP TABLE

- ⦿ Toutes les données de la table sont supprimées, ainsi que sa structure.
- ⦿ Toutes les transactions en cours sont validées.
- ⦿ Toutes les contraintes sont supprimées.
- ⦿ Vous ne pouvez pas annuler l'instruction DROP TABLE.

```
DROP TABLE dept [CASCADE CONSTRAINTS] ;  
Table dropped.
```

# GESTION DES VUES

## ◉ Création d'une Vue:

- Une vue est une vision partielle ou particulière des données d'une ou plusieurs tables de la base.
- Seule la définition de la vue est enregistrée dans la base, et pas les données de la vue. On peut parler de table virtuelle.

```
Create View vue(col1,col2,...) as select...  
WITH { READ ONLY | CHECK OPTION }  
[ CONSTRAINT constraint ] }
```

;

- **Exp.** CREATE VIEW emp2 (matr, NomE, dept) AS SELECT  
matr, nomE, dept FROM employees

## ◉ Suppression d'une Vue:

```
Drop view vue ;
```

# GESTION DES VUES

- Les vues créées avec l'option **WITH READ ONLY** peuvent être interrogées mais aucune opérations **DML** (UPDATE, DELETE, INSERT) peut être effectuées sur la Vue.
- L'option **WITH CHECK OPTION** ou **WITH CHECK OPTION CONSTRAINT** crée une contrainte de vérification sur la vue à partir de la clause **WHERE**.

Les vues créées avec l'option **WITH CHECK OPTION CONSTRAINT** empêche toutes mises à jour de la Vue si les conditions de la clause **WHERE** ne sont pas respectées.

```
SQL> CREATE VIEW v_emp_dept_10 (nom, métier, salaire, dpt) AS SELECT  
      ename, job, sal, deptno FROM emp WHERE deptno = 10 WITH CHECK  
      OPTION CONSTRAINT check_10;
```

```
SQL> INSERT INTO v_emp_dept_10 (nom, métier, salaire, dpt)  
      VALUES('Daniel','DBA',4000,78);
```

→ **ERREUR à la ligne 1 : ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE**

```
      INSERT INTO v_emp_dept_10 (nom, métier, salaire, dpt)  
      VALUES('Daniel','DBA',4000,10);
```

# INTÉRÊT DES VUES

La vue représente de cette façon une sorte d'intermédiaire entre la base de données et l'utilisateur. Cela a de nombreuses conséquences:

- une sélection des données à afficher
- une restriction d'accès à la table pour l'utilisateur, c'est-à-dire une sécurité des données accrue
- un regroupement d'informations au sein d'une entité. (faciliter la création de requêtes complexes)

→ *Une Vue est une table logique pointant sur une ou plusieurs tables ou vues et ne contient physiquement pas de données.*

*La structure d'une Vue est stockée dans le dictionnaire de données et peut contenir 1000 colonnes.*

# GESTION DES SÉQUENCES(1)

- Permet de définir une suite de nombres entiers, de clés uniques, de compteurs pour des tables.

```
CREATE SEQUENCE [schéma.]nomSéquence  
[INCREMENT BY entier ]  
[START WITH entier ]  
[ MAXVALUE entier ]  
[ MINVALUE entier ]  
[ { CYCLE | NOCYCLE } ]  
[ { CACHE entier | NOCACHE } ];
```

- L'interrogation d'une séquence se fait par l'utilisation des "pseudo-colonnes" CURRVAL et NEXTVAL. On parle de pseudo-colonne car cela se manipule un peu comme une colonne de table, mais ce n'est pas une colonne de table.
- La pseudo-colonne CURRVAL retourne la valeur courante de la séquence.
- La pseudo-colonne NEXTVAL incrémente la séquence et retourne la nouvelle valeur.

# GESTION DES SÉQUENCES(2)

SYNTAXE	DESCRIPTION
<b>INCREMENT BY</b>	intervalle entre les deux valeurs de la séquence (par défaut 1).
<b>START WITH</b>	première valeur de la séquence à générer.
<b>MAXVALUE</b>	valeur maximale de la séquence.
<b>MINVALUE</b>	valeur minimale de la séquence.
<b>CYCLE</b>	la séquence continuera de générer des valeurs après avoir atteint sa limite.
<b>NOCYCLE</b>	la séquence s'arrêtera de générer des valeurs après avoir atteint sa limite.
<b>CACHE</b>	les valeurs de la séquence seront mises en cache.
<b>NOCACHE</b>	les valeurs de la séquence ne seront pas dans le cache.



# GESTION DES INDEX(1)

**Index:** Structure de données permettant d'accélérer certains accès

Deux champs: la *clé d'index* et l'*adresse du bloc de données* contenant la clé.

**NB:** la clé est la valeur de un ou plusieurs attributs du fichier

- Creation d'un Index:

```
CREATE [UNIQUE] INDEX nom-index ON table(col1, col2,...)
```

- Suppression d'un index:

```
DROP INDEX nom_index [ON table]
```

- Un index est automatiquement supprimé dès qu'on supprime la table à laquelle il appartient.

# GESTION DES INDEX (2)

- On peut spécifier par l'option UNIQUE que chaque valeur d'index doit être unique dans la table.
- Un index peut être créé juste après la création d'une table ou sur une table contenant déjà des lignes.
- Un index peut porter sur plusieurs colonnes : la clé d'accès sera la concaténation des différentes colonnes.
- On peut créer plusieurs index indépendants sur une même table.

# GESTION DES INDEX(3)

- ◉ Renommer un Index:

```
ALTER INDEX index_name RENAME TO new_index_name;
```

# SYNTHÈSE

- ◉ Dans ce chapitre, vous avez appris à :
  - Répertorier les principaux objets de base de données.
  - utiliser l'instruction CREATE et inclure des contraintes.
  - Modifier la structure d'un objet.
  - Ajouter ou supprimer une contrainte à une table.
  - Supprimer une table.
  - Gérer les vues, les séquences, le synonymes et les index.