Annexe A : Langage de Contrôle de Données

Création d'utilisateur

```
CREATE USER nom_d'utilisateur IDENTIFIED BY mot_de_passe ;
```

Exemple

```
CREATE USER utilisateur1 IDENTIFIED BY tp1 ;
```

- **Remarque importante**

Le mot de passe et le *nom_d'utilisateur* doivent commencer par un caractère alphabétique.

Modification de mot de passe

```
ALTER USER nom_d'utilisateur IDENTIFIED BY mot_de_passe ;
```

Exemple :

```
ALTER USER utilisateur1 IDENTIFIED BY test ;
```

Suppression d'utilisateur

```
DROP USER nom_d'utilisateur ;
```

Exemple

```
DROP USER utilisateur1 ;
```

Transmission de privilèges

Lorsqu'un utilisateur crée une table ou une vue, il en est le propriétaire exclusif. Les autres utilisateurs n'y ont pas accès. Il peut cependant en donner un droit d'utilisation à d'autres par la commande GRANT.

```
GRANT privilège1 , privilège2, ... | ALL PRIVILEGES
[ON table/vue ]
TO utilisateur1 , utilisateur2 , ... | PUBLIC
[ WITH GRANT OPTION ] ;
```

Cette instruction accorde des privilèges d'accès « *privilège i* » ou tous les privilèges (ALL PRIVILEGES) sur la table ou vue aux utilisateurs « *utilisateur i* » ou à tous les utilisateurs (PUBLIC).

La clause WITH GRANT OPTION permet aux utilisateurs ayant reçu les privilèges de les transmettre à leur tour à d'autres utilisateurs.

Exemple

```
GRANT ALL PRIVILEGES TO utilisateur1 ;
```

L‟utilisateur « utilisateur1 » peut tout faire, il peut même accéder aux tables des autres utilisateurs.

- **Exemples de Privilèges :**

SELECT : lecture, INSERT : insertion, UPDATE : mise à jour, DELETE : suppression, ALL : tous les privilèges, ALTER : destruction, INDEX : construction d‟index

Exemples :

- Pour accorder à tous un droit de lecture sur une table T d'un utilisateur U, le propriétaire U accordera les droits nécessaires.

```
GRANT SELECT ON nom_table TO PUBLIC;
```

- Si on veut accorder à l' *utilisateur1* un droit de lecture sur une table T d'un utilisateur U, le propriétaire U accordera les droits nécessaires.

```
GRANT SELECT ON film TO utilisateur1;
GRANT SELECT, INSERT ON seance TO utilisateur1;
```

- **Exemples de privilèges systeme :**

CREATE SESSION : “ce privilège est utilisé pour ouvir une session sous SQL*PLUS (permet donc d‟ouvrir une fenêtre SQL*Plus et de se connecter à Oracle”.
CREATE TABLE, CREATE USER, CREATE VIEW, CREATE SESSION
Exemple

```
GRANT CREATE SESSION TO utilisateur3;
```

Suppression de privilèges

Annexe B : *Langage de Définition de Données*

1. Les commentaires

- Pour insérer des commentaires: il faut les précéder par : /* … */. Exemple :

```
/*  commentaire TP BDD   * /
```

2. Les symboles [ ] et |

- Les crochets **[ ]** indiquent que la commande est optionnelle.

Exemple

L‟instruction :

```
CREATE [UNIQUE] INDEX nom_index ON nom_table (attribut [ASC|DESC], …);
```

Sous SQLPLUS avec le mot clé UNIQUE :

```
CREATE UNIQUE INDEX  nom_indx  ON nom_table  ( nom_etudiant ASC );
```

Ou bien sous SQLPLUS sans le mot clé UNIQUE :

```
CREATE INDEX nom_indx  ON  nom_table  (nom_etudiant DESC);
```

- Ce symbole « | » indique le choix.
  Exemple

CREATE INDEX *nom_index* ON *nom_table* (attribut [ASC | DESC], …);

Sous SQLPLUS : dans l‟ordre ASC

```
CREATE INDEX nom_index ON nom_table (attribut ASC);
```

Ou bien l‟ordre DESC

```
CREATE INDEX nom_index ON nom_table (attribut DESC);
```

3. Création d’une table

CREATE TABLE *nom_table* **(Liste d’attributs suivis de leurs types séparer par des virgules) ;**

Type des données :

- NUMBER(n) : Entier à n chiffres
- NUMBER(n, m) : Réel à n chiffres au total (virgule comprise), m après la virgule
- INTEGER
- VARCHAR2(n) : Chaîne de n caractères (entre ' ')
- DATE : Date au format „JJ-MM-AAAA‟

- Un attribut déclaré "NOT NULL" doit nécessairement être renseigné (rempli) lors de l'insertion d'un tuple.
- Ne pas mettre de virgule avant la dernière parenthèse fermante de l‟instruction CREATE.

Exemple

```
CREATE TABLE JOUE (
  NOM_ACTEUR      VARCHAR2(30)          NOT NULL ,
  TITRE           VARCHAR2(40)          NOT NULL
 ) ;
```

4. Définition de la contrainte "CHECK"

Cette contrainte est utilisée pour limiter les valeurs qu'un attribut peut contenir. Si l'on définit la contrainte CHECK sur un attribut seules certaines valeurs seront autorisées pour cet attribut. Si peut également définit la contrainte CHECK pour limiter les valeurs d'un attribut en se basant sur les valeurs d'un autre attribut.

Exemple

```
CREATE TABLE Films (
  TITRE            VARCHAR2(40)          NOT NULL ,
  DUREE            INTEGER CHECK ( DUREE>0 and DUREE <10) ,
  NATIONALITE      VARCHAR2(30) ,
  NOM_REALISATEUR  VARCHAR2(30) CHECK (NOM_REALISATEUR LIKE '%r')
 ) ;
```

5. Définition de la contrainte "DEFAULT"

La contrainte DEFAULT est utilisée pour insérer une valeur par défaut dans une colonne. La valeur par défaut sera ajoutée à tous les nouveaux tuples, si aucune autre valeur n'est spécifiée.

Exemple

```
CREATE TABLE Films (
  TITRE            VARCHAR2(40)          NOT NULL ,
  DUREE            INTEGER CHECK ( DUREE>0 and DUREE <10) ,
  NATIONALITE      VARCHAR2(30) DEFAULT 'Algérienne',
  NOM_REALISATEUR  VARCHAR2(30) check (NOM_REALISATEUR LIKE '%r')
 ) ;
```

6. Définitions des contraintes d'intégrité: Clé primaire

Il est préférable de nommer de la contrainte clé primaire de cette façon: PK_ *nom_table1*

```
CONSTRAINT nom_contrainte PRIMARY KEY (attribut_clé [, attribut_clé2, …])
```

7. Définitions des contraintes d'intégrité: Clé étrangère

Il est préférable de nommer la contrainte clé étrangère de cette façon: FK_*nom_table*1_ *nom_table2*

```
CONSTRAINT nom_contrainte FOREIGN KEY (attributs) REFERENCES table (attribut)
```

De plus, pour supprimer un tuple et que sa suppression se répercute dans les autres table liées par "REFERENCES" alors il faut ajouter "ON DELETE CASCADE". Avec cette fonction la suppression se fera dans toute les tables liées.

```
CONSTRAINT nom_contrainte FOREIGN KEY (attributs) REFERENCES table (attribut) ON DELETE CASCADE
```

L'instruction de création de table devient alors:

```
CREATE TABLE nom_table1 ( liste d'attributs avec leurs types séparer par des virgules
CONSTRAINT nom_contrainte PRIMARY KEY (liste1 d'attributs) ,
CONSTRAINT nom_contrainte FOREIGN KEY (liste2 d'attributs) REFERENCES
nom_table2 (liste2 d'attributs) ) ;
```

Exemple

```
CREATE TABLE JOUE (
  NOM_ACTEUR      VARCHAR2(30)      NOT NULL ,
  TITRE           VARCHAR2(40)      NOT NULL ,
  CONSTRAINT PK_JOUE PRIMARY KEY ( NOM_ACTEUR , TITRE ) ,
  CONSTRAINT FK_JOUE_FILM FOREIGN KEY (TITRE) REFERENCES FILM
(TITRE)
) ;
```

8. Synonyme d'une table

- **Création d'un Synonyme d'une table**

```
CREATE PUBLIC SYNONYM le_nom_du_synonyme FOR nom_table ;
```

Exemple

```
CREATE PUBLIC SYNONYM toto FOR system.film;
```

```
SELECT * FROM toto;
/****** identique à ****/
SELECT * FROM system.film ;
```

Exemple

```
GRANT SELECT ON system.film TO PUBLIC;
/*** ou bien ******/
GRANT SELECT ON toto TO PUBLIC;
```

9. Ajout d'attributs dans une table déjà créée

```
ALTER TABLE nom_table ADD ( attribut TYPE , …) ;
```

10. Modifications du type d'un attribut existant

```
ALTER TABLE nom_table MODIFY (attribut TYPE, …);
```

11. Suppression de contraintes

```
ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte ;
```

12. Description d'une table créée

- Utiliser DESC pour voir la description d"une table déjà créée.

```
DESC nom_table ;
```

Exemple

```
DESC film ;
```

13. Création d'un index

```
CREATE [ UNIQUE ] INDEX nom_index ON nom_table (attribut [ASC|DESC], …);
```

Tel que : UNIQUE → pas de double
ASC/DESC → ordre croissant ou décroissant

Exemple

```
CREATE INDEX JOUER_FK ON joue ( TITRE ASC ) ;
```

14. Suppression d'index ou de table

- **Pour supprimer une table**

```
DROP TABLE nom_table ;
```

- **Pour supprimer un index**

```
DROP INDEX nom_index ;
```

15. Ajout de tuples

```
INSERT INTO nom_table VALUES (valeur_de_attribut1, valeur_de_attribut2, …);

COMMIT ;  /*validation des insertions*/
```

- Il faut mettre les chaines de caractères entre côtes exemple „test BDD" .
- Utiliser le mot clé **NULL** pour laisser un attribut vide.
- Il ne faut pas oublier de valider les insertions en exécutant la commande « *commit* »
- Après chaque exécution l"instruction INSERT une réponse de SQL*Plus est affichée
    - **SQL> 1 Ligne créée**
- Après exécution de l"instruction COMMIT une réponse de SQL*Plus est affichée
    - **SQL> validation effectuée**

16. Mise à jour d'un attribut

```
UPDATE nom_table SET attribut = valeur [ WHERE condition ];
```

Exemple

```
UPDATE film SET duree = 60 WHERE titre = „Waterworld";
```

17. Suppression de tuples

```
DELETE FROM nom_table [ WHERE condition ];
```

Exemples

```
DELETE FROM film WHERE titre = „waterworld";
```

```
DELETE FROM film ;
```

18. Les Vues

Une vue est une table virtuelle calculée à partir d"autres tables grâce à une requête. L'intérêt des vues est de :

- Simplifier l"accès aux données en masquant les opérations de jointure.
- Sauvegarder des requêtes complexes.
- Présenter de mêmes données sous différentes formes adaptées aux différents usagers particuliers. Renforcer de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités

19. Définition d'une vue

CREATE VIEW *nom_vue* **AS** *requête*;

20. Problèmes de mise à jour, restrictions

La mise à jour de données via une vue pose des problèmes et la plupart des systèmes impose d'importantes restrictions.

- Le mot clé DISTINCT doit être absent.
- La clause FROM doit faire référence à une seule table.
- La clause SELECT doit faire référence directement aux attributs de la table concernée (pas d'attribut dérivé).
- Les clauses GROUP BY et HAVING sont interdites.

21. Catalogue du système

Contient sous forme relationnelle la définition de tous les objets créés par le système et les usagers. Ces tables sont accessibles avec SQL (en mode consultation uniquement).

```
USER_CATALOG (TABLE_NAME ,    TABLE_TYPE)
USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME, …)
USER_IND_COLUMNS (INDEX_NAME, TABLE_NAME, COLUMN_NAME, …)
ALL_TABLES (TABLE_NAME, OWNER, …)
ALL_USERS  (USERNAME , USER_ID ,  CREATED)
```

Annexe C : Langage de Manipulation de Données

Après la création des tables, il est alors possible de récupérer les données stockées grâce à la commande SELECT :

```
SELECT [ALL|DISTINCT] attribut(s) FROM nom_table [ WHERE condition]
   [GROUP BY attribut(s) [HAVING condition]]
          [ORDER BY attribut(s) [ASC|DESC]]   ;
```

- **Pour afficher tous les attributs et les tuples d'une table**

```
SELECT    *    FROM  nom_table ;
```

Exemple : afficher tout le contenu de la table film

```
SELECT  *   FROM   film;
```

Exemple : afficher deux attributs de la table film

```
SELECT  titre , nationalite   FROM   film;
```

- **Pour ordonner les tuples**
    - ORDER BY attribut ASC : ordre ascendant
    - ORDER BY attribut DESC : ordre descendant.

Exemple

```
SELECT   *   FROM    Film
ORDER   BY   nationalite   ASC;
```

- **Pour afficher un ensemble d'attribut d'une table (Projection)**

Exemple

```
SELECT   titre , nationalite   FROM   film;
```

- **Pour afficher un ensemble de tuples vérifiant une condition parmi les tuples d'une table. (Restriction)**

Exemple

```
SELECT   * FROM   film   WHERE   nationalite = 'US' ;
```

Dans le "WHERE" on peut :

1. Employer les opérateurs de comparaison (>,<, ≤, ≥, =) ;

2. Vérifier l‟appartenance d‟un attribut à un intervalle

```
WHERE attribut BETWEEN valeur1 AND valeur2  ;
```

3. Vérifier qu‟un attribut possède comme valeur une chaîne de caractère qui se termine par un ensemble de caractère précis (chaine) :

```
WHERE attribut LIKE  '%chaine'
```

4. Vérifier qu‟un attribut commence par un ensemble de caractère précis (chaine)

```
WHERE attribut LIKE 'chaine%'
```

5. Vérifier qu‟un attribut contient une chaine de caractère (chaine) :

```
WHERE attribut LIKE  '%chaine%'
```

6. Préciser qu‟un attribut appartient à un ensemble

Exemple : vérifier si la valeur de l'attribut est égale à x, y ou z

```
WHERE  attribut  IN  (x, y , z)
```

7. Utiliser la négation pour tous ces prédicats : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN.

```
WHERE attribut NOT LIKE  '%chaine%'
```

• **Fonctions d'agrégat :** Elles opèrent sur un ensemble de valeurs.

– AVG(), VARIANCE(), STDDEV() : moyenne, variance et écart-type des valeurs ;
– SUM() : somme des valeurs ;
– MIN(), MAX() : valeur minimum, valeur maximum ;
– COUNT() : nombre de valeurs.