Mémento

Oracle Database (SQL)

Version 2.0 (créé le 10/08/2023, modifié le 10/08/2024)

SQL (Structured Query Language) est un langage servant à exploiter des bases de données relationnelles.

Oracle Database est souvent utilisé pour les bases de données de sites web qui utilisent le langage PHP, et très souvent utilisé pour les bases de données des applications utilisant le langage Java.

Toutes les options facultatives sont représentées par des [ ].

Loric Informatique

Table des matières

1. Prise en main

1.1. Outils nécessaires

- SQL Developer (contenant Oracle si vous avez un serveur simple hébergé)

2. Bases

2.1. Syntaxe

```
requête1;
requête2;
requête3;
```

2.2. Les variables

2.2.1. Types de variables

2.2.1.1. Numériques

Type	Limites	Description
`NUMBER(X)`	Longueur inférieure ou égale à *X* chiffres (maximum : 255)	Nombre entier
`NUMBER(X, Y)`	Longueur inférieure ou égale à *X* chiffres dans la partie entière et inférieure ou égale à *Y* chiffres après la virgule (maximum : 255)	Nombre décimal

2.2.1.2. Caractères

Type	Limites	Description
`CHAR(X)`	Longueur fixée à *X* caractères (maximum : 255)	Chaîne de caractères entre ' '
`VARCHAR2(X)`	Longueur inférieure ou égale à *X* caractères (maximum : 255)	Chaîne de caractères entre ' '

2.2.1.3. Dates et heures

Type	Format	Description
`DATE`	`DD/MM/YYYY`	Date entre ' '

2.2.1.4. Autres types

Type	Description
`BOOLEAN`	Valeur pouvant être TRUE ou FALSE

2.3. Commentaires

```
--Commentaire tenant sur une ligne

/*
Commentaire pouvant être sur une ou plusieurs lignes
*/
```

2.4. Opérations mathématiques

Instruction	Description
`1 + 2`	Renvoie 3
`3 - 1`	Renvoie 2
`6 * 4`	Renvoie 24
`5.0 / 2.0`	Renvoie 2.5

`5 / 2`	Renvoie 2 (le quotient sans décimal)
`'2' || '7'`	Renvoie '27'

2.5. Opérateurs de conditions

Condition	**Description de ce que vérifie la condition**
a = *b*	*a* égal à *b* (seulement en contenu : 1='1')
a < *b*	*a* strictement inférieur à *b*
a > *b*	*a* strictement supérieur à *b*
a <= *b*	*a* supérieur ou égal à *b*
a != *b*	*a* n'est pas égal à *b* (seulement en contenu : 1='1')
a <> *b*	*a* n'est pas égal à *b* (seulement en contenu : 1='1')
a = ALL *b*	*a* est égale à toutes les valeurs dans *b* (qui peut être une sous-requête ou une liste sous forme : (*val1, val2…*))
a = ANY *b*	*a* est égale à au moins une valeur dans *b* (qui peut être une sous-requête ou une liste sous forme : (*val1, val2…*))
a IN *b*	*a* est présent dans *b* (qui peut être une sous-requête ou une liste sous forme : (*val1, val2…*))
a BETWEEN *b* AND *c*	*a* est compris entre *b* et *c*
a LIKE '*SousChaine*'	*a* contient la sous-chaîne (avec _ correspondant à 1 caractère et % correspondant à 0 ou plusieurs caractères)
a IS [NOT] NULL	Tester si une variable est nulle
a IS [NOT] UNKNOWN	Tester si une variable est définie

`OR`	À mettre entre deux conditions, permet d'avoir une des deux conditions qui doit être vraie
`AND`	À mettre entre deux conditions, permet d'avoir deux conditions qui doivent être vraie
`NOT condition`	Ne doit pas respecter la condition

3. Les fonctions

3.1. Les fonctions arithmétiques

Fonction	**Description**
`ROUND(n, d)`	Arrondit *n* au réel à *d* chiffres après la virgule ou à l'entier si *d* n'est pas renseigné
`TRUNC(n, d)`	Tronque *n* à *d* chiffres après la virgule ou à 0 si *d* n'est pas renseigné
`POWER(n, m)`	Renvoie *n* à la puissance *m* (si *n* est négatif, *m* doit être un entier)
`CEIL(n)`	Renvoie un entier directement supérieur ou égal à *n*
`FLOOR(n)`	Renvoie un entier directement supérieur ou égal à *n* (partie entière de *n*)
`ABS(n)`	Renvoie la valeur absolue de *n*
`MOD(n, m)`	Renvoie le reste de la division de *n* par *m*
`SQRT(n)`	Renvoie la racine carrée
`SIGN(n)`	Renvoie -1 si *n* est négatif, 1 si *n* est positif et 0 si *n* égal à 0

3.2. Les fonctions pour les chaînes de caractères

Fonction	Description
`LENGTH(chaine)`	Renvoie la longueur de la chaine
`SUBSTR(chaine, debut[, longueur])`	Renvoie la position (en commençant à une certaine position et allant jusqu'à la longueur fixée ou à la fin)
`UPPER(chaine)`	Convertit en majuscule
`LOWER(chaine)`	Convertit en minuscule
`INITCAP(chaine)`	Met en majuscule la première lettre et en minuscule les autres
`TRANSLATE(chaine, c1, c2)`	Remplace chaque caractère *c1* par *c2* dans la chaîne
`REPLACE(chaine, ch1, ch2)`	Remplace chaque chaine *ch1* par *ch2* dans la chaîne

3.3. Les fonctions pour les dates

3.3.1. Les masques pour les dates

Masque	Description
`'CC'`	Siècle (ex : 21)
`'YYYY'`	Année (ex : 2024)
`'Q'`	Numéro du trimestre dans l'année (ex : 3)
`'WW'`	Numéro de la semaine dans l'année
	Numéro du mois (ex : 5)
`'MM'`	Numéro du mois (ex : 05)
`'DDD'`	Numéro du jour dans l'année (ex : 185)
`'DD'`	Numéro du jour dans le mois (ex : 08)

'D'	Numéro du jour dans la semaine (ex : 5) avec dimanche égal à 0
'HH' ou 'HH12'	Heure sur 12 heures (ex : 09)
'HH24'	Heure sur 24 heures (ex : 21)
'MI'	Minutes (ex : 05)
'S'	Secondes (ex : 30)
'YEAR'	Année en lettres
'MONTH'	Mois en lettres
'MON'	Mois abrégé (3 lettres)
'DAY'	Jour en lettres
'DY'	Jour abrégé (3 lettres)

3.3.2. Les fonctions de manipulation des dates

Fonction	**Description**
SYSDATE ou CURRENT_DATE	Renvoie la date actuelle
CURRENT_TIMESTAMP ou SYSTIMESTAMP	Renvoie la date et l'heure actuelle
ADD_MONTHS(*date*, *nombre*)	Ajoute ou soustrait un certain nombre de mois
LAST_DAY(*date*)	Expression de type date qui a pour valeur la date du dernier jour du mois contenant la date
NEXT_DAY(*date*, *jour*)	Expression de type date qui a pour valeur la date du prochain jour de la semaine spécifié dans le jour
ROUND(*date*, *masque*)	Arrondit la date à la précision spécifiée
TRUNC(*date*, *masque*)	Tronque la date à la précision spécifiée
EXTRACT(DAY FROM *date*)	Extrait le jour à partir d'une date
EXTRACT(MONTH FROM *date*)	Extrait le mois à partir d'une date
EXTRACT(YEAR FROM *date*)	Extrait l'année à partir d'une date
TO_DATE(*chaine*, *masque*)	Convertit une chaîne de caractères en date (ex : TO_DATE('10/12/2024', 'DD/MM/YYYY'))

`TO_CHAR(date, masque)`	Convertit une date en chaîne de caractères (ex : `TO_CHAR(SYSDATE, 'DD/MM/YYYY')`)

3.4. Autres fonctions

Fonction	Description
`CHR(n)`	Retourne le caractère dont le code (ASCII ou EBCDIC) est égal à l'expression numérique entré en paramètre
`GREATEST(exp1, exp2…)`	Retourne la plus grande des valeurs des expressions arguments
`LEAST(exp1, exp2…)`	Retourne la plus petite des valeurs des expressions arguments
`COALESCE(exp1, exp2…)`	Retourne la première valeur différente de NULL des expressions arguments, s'il y en a une, et la valeur NULL s'il n'y en a pas

3.5. Les fonctions d'agrégat (pour des sélections uniquement, ne concernent pas les conditions)

Fonction	Description
`AVG(expression)`	Moyenne des valeurs d'une colonne
`SUM(expression)`	Somme des valeurs d'une colonne
`MIN(expression)`	La plus petite des valeurs d'une colonne
`VARIANCE(expression)`	La plus grande des valeurs d'une colonne
`STDDEV(expression)`	Écart-type ou déviation standard
`COUNT(*)`	Nombre de lignes
`COUNT(expression)`	Nombre de lignes ayant pour valeur non nulle

`COUNT(DISTINCT expression)`	Nombre de lignes ayant des valeurs distinctes non nulles

4. Requêtes d'initialisation de la base de données

4.1. Créer une table (tableau de valeurs)

```
CREATE TABLE [IF NOT EXISTS] table1 (
    attribut1 type1 [[CONSTRAINT nomContrainte1] contrainte1]
[GENERATED AS IDENTITY],
    attribut2 type2 [[CONSTRAINT nomContrainte2] contrainte2],
    attribut3 type3 [[CONSTRAINT nomContrainte3] contrainte3],
    [CONSTRAINT nomContrainte4] PRIMARY KEY (attribut1),
    [[CONSTRAINT nomContrainte5] FOREIGN KEY (attribut3) REFERENCES
table2(attribut1)]
);
```

ou

```
CREATE TABLE table1 (
    attribut1 type1 [CONSTRAINT nomContrainte1] PRIMARY KEY
[GENERATED AS IDENTITY],
    attribut2 type2 [[CONSTRAINT nomContrainte2] contrainte2],
    attribut3 type3 [[CONSTRAINT nomContrainte3] REFERENCES
table2(attribut1)]
);
```

`GENERATED AS IDENTITY` permet de créer automatiquement les valeurs des clés primaires si le type est un nombre entier.

4.2. Les contraintes

4.2.1. Liste des contraintes

Contrainte	Initiales nom contrainte	Description
`NOT NULL`	*nn_nomContrainte*	Permet de rendre obligatoire l'entrée d'une valeur dans un attribut
`CHECK(`*condition*`)`	*ck_nomContrainte*	Permet de vérifier la condition avant d'insérer un enregistrement
`UNIQUE`	*uq_nomContrainte*	Permet d'éviter l'insertion de plusieurs valeurs identiques
`DEFAULT` *valeurParDefaut*	*df_nomContrainte*	Permet de remplacer une valeur non renseignée par une autre
`PRIMARY KEY (`*attribut1*`)`	*pk_nomContrainte*	Permet de créer une clé primaire. Il s'agit généralement d'un identifiant, et sa valeur est unique
`FOREIGN KEY (`*attribut3*`) REFERENCES` *table2*`(`*attribut7*`)`	*fk_nomContrainte*	Permet de créer une clé étrangère en relation avec une autre table (déjà existante)

4.2.2. Ajouter une contrainte

```
ALTER TABLE table1
ADD CONSTRAINT nomContrainte1 contrainte1;
```

Il est fortement conseillé d'ajouter une contrainte en même temps que la création d'une table. Ces commandes peuvent être utile si une contrainte a été oubliée ou n'aurait pas dû être insérée.

4.2.3. Supprimer une contrainte

```
ALTER TABLE table1
DROP CONSTRAINT nomContrainte1;
```

4.2.4. Activer une contrainte d'intégrité

```
ALTER TABLE table1
ENABLE nomContrainte1;
```

4.2.5. Désactiver une contrainte d'intégrité

```
ALTER TABLE table1
DISABLE nomContrainte1;
```

4.3. Copier une table

```
CREATE TABLE table2 AS (SELECT * FROM table1);
```

4.4. Renommer une table

```
RENAME ancienNom TO nouveauNom;
```

4.5. Supprimer une table

```
DROP TABLE [IF EXISTS] table1 [CASCADE CONSTRAINTS];
```

5. Requêtes de mise à jour du contenu de la table

5.1. Insérer un ou plusieurs enregistrement (ligne)

```
INSERT INTO table1 [(attribut1, attribut2…)] VALUES
(valeur1, valeur2…);
```

5.2. Mettre à jour un ou plusieurs enregistrement(s) respectant la condition

```
UPDATE table1
SET attribut1 = valeur1 [, attribut2 = valeur2…]
WHERE condition;
```

5.3. Supprimer un ou plusieurs enregistrement(s) respectant la condition

```
DELETE FROM table1
WHERE condition;
```

6. Requêtes de mise à jour de la structure de la table

6.1. Ajouter une colonne

```
ALTER TABLE table1
ADD attribut type;
```

6.2. Supprimer une colonne

```
ALTER TABLE table1
DROP [COLUMN] attribut;
```

6.3. Modifier le type d'une colonne

```
ALTER TABLE table1
MODIFY attribut nouveauType;
```

6.4. Modifier le nom d'une colonne

```
ALTER TABLE table1
RENAME ancienNom TO nouveauNom;
```

7. Requêtes de sélection de données

7.1. Syntaxe de base

```
SELECT attribut1, attribut2… FROM table1
[WHERE condition1]
[GROUP BY attribut1 [, attribut2…]
[HAVING condition2]]
[ORDER BY attribut1 [DESC], attribut2 [DESC]…];
```

7.2. La clause SELECT

7.2.1. Sélection de tous les enregistrements de la table

```
SELECT * FROM table1;
```

7.2.2. Sélection de certains attributs de la table

```
SELECT attribut1, attribut2 FROM table1;
```

7.2.3. Sélection d'un attribut de la table sans répétitions

```
SELECT DISTINCT attribut1 FROM table1;
```

7.2.4. Utilisation d'une fonction d'agrégat (ne renvoie qu'une seule valeur sans regroupement)

```
SELECT maFonction(attribut1) FROM table1;
```

7.2.5. Sélection des *n* premiers enregistrements de la table

```
SELECT * FROM table1
[ORDER BY attribut1 [DESC], attribut2 [DESC]…]
FETCH FIRST n ROW[S] ONLY;
```

7.3. Les jointures (clause JOIN)

Les jointures permettent de sélectionner des attributs de plusieurs tables différentes (les jointures peuvent s'accumuler).

7.3.1. Syntaxe de base

```
SELECT * FROM table1
JOIN table2 ON table1.attribut = table2.attribut
[JOIN table3 ON table2.attribut = table3.attribut
…]
[WHERE condition];
```

7.3.2. Jointure interne simple

```
SELECT table1.attribut2, table2.attribut3 FROM table1
[INNER] JOIN table2 ON table1.attribut1 = table2.attribut4;
```

Autre solution (à utiliser uniquement pour deux attributs identiques de deux tables différentes) :

```
SELECT table1.attribut2, table2.attribut3 FROM table1
[INNER] JOIN table2 USING(attribut1);
```

7.3.3. Jointure externe gauche (avec des enregistrements de la table de gauche non présents dans la table de droite)

```
SELECT table1.attribut1, table2.attribut3 FROM table1
LEFT [OUTER] JOIN table2 ON table1.attribut1 = table2.attribut2;
```

7.3.4. Jointure externe droite (avec des enregistrements de la table de droite non présents dans la table de gauche)

```
SELECT table1.attribut1, table2.attribut3 FROM table1
RIGHT [OUTER] JOIN table2 ON table1.attribut1 = table2.attribut2;
```

7.3.5. Jointure externe entière (avec des tous les enregistrements non présents dans une autre table)

```
SELECT table1.attribut1, table2.attribut3 FROM table1
FULL [OUTER] JOIN table2 ON table1.attribut1 = table2.attribut2;
```

7.3.6. Jointure naturelle (pour deux tables avec une colonne du même nom)

```
SELECT table1.attribut1, table2.attribut3 FROM table1
NATURAL JOIN table2;
```

7.3.7. Produit cartésien (retourne chaque ligne d'une table avec chaque ligne d'une autre table)

```
SELECT table1.attribut1, table2.attribut3 FROM table1
CROSS JOIN table2;
```

7.4. Les alias

7.4.1. Alias sur les tables (pour faciliter les jointures)

```
SELECT t1.attribut2, t2.attribut3 FROM table1 t1
JOIN table2 t2 ON t1.attribut1 = t2.attribut4;
```

7.4.2. Alias sur une fonction d'agrégat (pour nommer une colonne sans nom ou renommer une colonne)

```
SELECT maFonction(attribut1) [AS] nomColonne FROM table1;
```

7.5. La clause WHERE

7.5.1. Sélection des enregistrements de la table respectant la condition

```
SELECT * FROM table1
WHERE condition;
```

7.6. Les clauses GROUP BY et HAVING

7.6.1. Utilisation d'une fonction d'agrégat sur chaque groupe d'enregistrements de même(s) attribut(s)

```
SELECT maFonction(attribut1), attribut2, attribut3… FROM table1
GROUP BY attribut2, attribut3…;
```

7.6.2. Utilisation d'une fonction d'agrégat sur chaque groupe d'enregistrements de même(s) attribut(s) avec une condition avant et après regroupement

```
SELECT maFonction(attribut1), attribut2, attribut3… FROM table1
WHERE conditionAvantRegroupement
GROUP BY attribut2, attribut3…
HAVING conditionApresRegroupement;
```

7.7. La clause ORDER BY

7.7.1. Sélection des enregistrements de la table dans l'ordre croissant

```
SELECT * FROM table1
[WHERE condition]
ORDER BY attribut1 [ASC][, attribut2 [ASC]…];
```

7.7.2. Sélection des enregistrements de la table dans l'ordre décroissant

```
SELECT * FROM table1
[WHERE condition]
ORDER BY attribut1 DESC[, attribut2 DESC…];
```

7.8. Les requêtes multiples de sélection

7.8.1. Les sous-requêtes

Les sous-requêtes sont des requêtes effectuées dans d'autres requêtes. Elles remplacent souvent les tables, mais peuvent également remplacer les attributs uniquement si elles ne renvoient qu'une seule valeur.

7.8.2. Sélection des enregistrements d'un attribut présents dans la première table ou la deuxième ou les deux (union)

```
SELECT attribut1 FROM table1
UNION
SELECT attribut1 FROM table2;
```

7.8.3. Sélection des enregistrements d'un attribut présents dans la première table, mais pas la deuxième (différence)

```
SELECT attribut1 FROM table1
MINUS
SELECT attribut1 FROM table2;
```

7.8.4. Sélection des enregistrements d'un attribut présents dans la première table et la deuxième (intersection)

```
SELECT attribut1 FROM table1
INTERSECT
SELECT attribut1 FROM table2;
```

8. Les index

Les index permettent de gagner en temps d'exécution pour une sélection d'une colonne avec beaucoup de valeurs.

8.1. Créer un index

```
CREATE INDEX monIndex ON table1 [(attribut1[, attribut2…])];
```

8.2. Supprimer un index

```
DROP INDEX monIndex;
```

9. Les vues

Les vues permettent de réaliser des requêtes de sélections plus facilement. Elles se comportent comme des tables pour les requêtes de sélection.

9.1. Créer une vue

```
CREATE VIEW maVue AS SELECT * FROM table1;
```

9.2. Utiliser la vue

```
SELECT attribut1, attribut2… FROM maVue;
```

9.3. Supprimer la vue

```
DROP VIEW [IF EXISTS] maVue;
```

10. Les CTE (Common Table Expression)

Les CTE sont des vues provisoires, la syntaxe est la suivante :

```
WITH
myCTE1 AS (requete1),
myCTE2 AS (requete2)…
requeteFinale;
```

11. Autres requêtes de données

11.1. Requête de description d'une table

```
DESC table1;
```

ou

```
DESCRIBE table1;
```

11.2. Les transactions

11.2.1. Valider une transaction

```
COMMIT;
```

11.2.2. Annuler une transaction

```
ROLLBACK;
```

12. Programmation procédurale et structurée (PL/SQL)

12.1. Syntaxe du bloc principal

```
[DECLARE
    declarations;]
BEGIN
    instructions;
[EXCEPTION
    gestion_des_erreurs;]
END;
```

Note : Le bloc DECLARE peut être retiré si aucune déclaration n'a été faite. Le bloc EXCEPTION permet de gérer les erreurs du programme, il n'est donc pas obligatoire. Il est possible de faire plusieurs blocs BEGIN imbriqués.

12.2. Déclarations (dans le bloc DECLARE)

12.2.1. Variables de types simples

```
v_variable1 type1 := valeur;
v_variable2 type2;
v_variable3 type3;
```

12.2.2. Variables constantes

```
v_variable1 CONSTANT type1 := valeur;
```

12.2.3. Variables de types d'une colonne

```
v_variable1 table1.attribut1%TYPE;
```

12.2.4. Variables de types d'une ligne entière d'une table

```
v_variable1 table1%ROWTYPE;
```

12.2.5. Curseurs explicites

12.2.5.1. Curseur sans paramètres

```
CURSOR c_curseur IS (SELECT * FROM table1 WHERE condition [ORDER BY
attribut1 [DESC]]);
```

12.2.5.2. Curseur avec paramètres

```
CURSOR c_curseur(v_attribut1 typeSansParentheses) IS (SELECT * FROM
table1 WHERE attribut1 = v_attribut1 [ORDER BY attribut1 [DESC]]);
```

12.2.5.3. Curseurs avec verrouillage des données (pour une mise à jour de la table notamment)

```
CURSOR c_curseur IS (SELECT * FROM table1 WHERE condition [ORDER BY
attribut1 [DESC]]) FOR UPDATE;
```

Note : Pour désigner l'élément sélectionné par le curseur dans une condition, entrez : `CURRENT OF c_curseur`

12.2.6. Exceptions

12.2.6.1. Exception simple

```
NOM_EXCEPTION_PERSONNALISEE EXCEPTION;
```

12.2.6.2. Exception créée à l'aide d'un numéro d'exception

```
NOM_EXCEPTION_PERSONNALISEE EXCEPTION;
PRAGMA EXCEPTION_INIT(NOM_EXCEPTION_PERSONNALISEE,
numero_exception);
```

12.3. Opération d'affectation

12.3.1. Affectation simple

```
v_variable := valeur;
```

12.3.2. Affectation avec une requête de sélection (ne fonctionne que si la sélection ne renvoie qu'une seule valeur)

```
SELECT attribut1 INTO v_variable FROM table1 WHERE condition;
```

12.4. Conditions

Instruction	Description

`IF condition1 THEN`
`  instruction1;`
`END IF;`
	Si *condition1* est vraie, alors on exécute *instruction1*

`IF condition1 THEN`
`  instruction1;`
`ELSE`
`  instruction2;`
`END IF;`
	Si *condition1* est vraie, alors on exécute *instruction1*, sinon, on exécute *instruction2*

`IF condition1 THEN`
`  instruction1;`
`}`
`ELSIF condition2 THEN`
`  instruction2;`
`ELSE`
`  instruction3;`
`END IF;`
	Si *condition1* est vraie, alors on exécute *instruction1*, sinon, si *condition2* est vraie, on exécute *instruction2*, sinon, on exécute *instruction3*

12.5. Boucles

Instruction	Description

`FOR i IN 0..n LOOP`
`    instruction1;`
`END LOOP;`
	On répète *n* fois *instruction1* pour *i* allant de 0 à *n* compris (pas besoin de déclarer la variable, elle est déclarée implicitement et est détruite à la fin de la boucle)

`FOR elt IN (SELECT * FROM table1) LOOP`
`    instruction1;`
`END LOOP;`
	On parcourt la requête de sélection (ou une chaine de caractère) pour *elt* prenant tous les enregistrements de la requête (pas besoin de déclarer la variable, elle est déclarée implicitement et est détruite à la fin de la boucle)

`WHILE condition LOOP`
`    instruction1;`
`END LOOP;`
	On répète jusqu'à ce que condition soit fausse (peut ne pas être répété)

`LOOP`
`    instruction1;`
`    EXIT WHEN condition;`
`    instruction2;`
`END LOOP;`
	On répète jusqu'à ce que condition soit fausse (*instruction1* est forcément répété une fois)

`EXIT;`	Permet de sortir d'une boucle sans la terminer

12.6. Les curseurs implicites et explicites

12.6.1. Les curseurs implicites

Curseur	Description
`SQL%ROWCOUNT`	Nombre de lignes traitées par la requête précédentes de mise à jour de la table
`c_curseur%NOTFOUND`	Vérifier que le curseur est vide ou la dernière ligne est déjà atteinte

c_curseur%FOUND	Vérifier que le curseur n'est pas vide ou la dernière ligne n'est pas encore atteinte
c_curseur%ISOPEN	Vérifier que le curseur est ouvert

12.6.2. Les curseurs explicites

```
OPEN c_curseur[(parametres)];
LOOP
    FETCH c_curseur INTO v_variable;
    EXIT WHEN c_curseur%NOTFOUND;
    instructions;
END LOOP;
CLOSE c_curseur;
```

ou

```
FOR v_variable IN c_curseur[(parametres)] LOOP
    instructions;
END LOOP;
```

12.7. Les procédures

12.7.1. Créer une procédure

```
CREATE OR REPLACE PROCEDURE maProcedure [(parametreEntree1 IN type1,
parametreSortie1 OUT type2…)]
IS
    [déclarations]
BEGIN
    instructions;
EXCEPTION
    gestion_des_erreurs;
END;
```

Note : Les paramètres d'entrée et de sortie ne sont pas obligatoires. Une variable en paramètre peut être à la fois une entrée et une sortie (avec IN OUT). La gestion des exceptions n'est pas obligatoire.

12.7.2. Retourner une valeur de la procédure

```
parametreSortie1 := valeur;
```

12.7.3. Faire un appel à la procédure

```
maProcedure[(variable1, variable2)];
```

ou en dehors d'un bloc :

```
EXECUTE maProcedure[(variable1, variable2…)];
```

12.7.4. Supprimer une procédure

```
DROP PROCEDURE maProcedure;
```

12.8. Les fonctions

12.8.1. Créer une fonction

```
CREATE OR REPLACE FUNCTION maFontion [(parametre1 type1, parametre2
type2…)]
RETURN type3
IS
    [déclarations]
BEGIN
    instructions;
EXCEPTION
    gestion_des_erreurs;
END;
```

12.8.2. Retourner une valeur de la fonction

```
RETURN variable;
```

12.8.3. Faire un appel à la fonction

```
variable := maFontion[(variable1, variable2…)];
```

ou en dehors d'un bloc :

```
SELECT maFontion[(variable1, variable2…)] FROM dual;
```

12.8.4. Supprimer une fonction

```
DROP FUNCTION maFontion;
```

12.9. Les exceptions (dans le bloc EXCEPTION)

12.9.1. Exceptions prédéfinies

Nom exception	Erreur Oracle	Numéro exception	Description
(Sans nom)	ORA-00001	-803	Violation de contrainte unique
(Sans nom)	ORA-01400	-407	Violation de contrainte not null
NO_DATA_FOUND	ORA-01403	100	Aucune donnée trouvée
TOO_MANY_ROWS	ORA-01422	-1422	Plus d'une ligne de renvoyées
ZERO_DIVIDE	ORA-01476	-1476	Division par zéro
(Sans nom)	ORA-02290	-2290	Violation de contrainte check
STORAGE_ERROR	ORA-06500	-6500	Erreur de stockage

VALUE_ERROR	ORA-06502	-6502	Valeur incorrecte
CURSOR_ALREADY_OPEN	ORA-06511	-6511	Curseur déjà ouvert
COLLECTION_IS_NULL	ORA-06531	-6531	Collection nulle
(Sans nom)	ORA-00902	-104 ou -199	Type de donnée invalide
OTHER	ORA-42612	-84	Autres erreurs

12.9.2. Provoquer une erreur personnalisée

```
RAISE NOM_EXCEPTION_PERSONNALISEE;
```

12.9.3. Gestion des erreurs (dans le bloc EXCEPTION)

```
WHEN NOM_EXCEPTION1 THEN
    instructions1;
WHEN NOM_EXCEPTION2 THEN
    instructions2;
```

12.10. Les instructions de base

Instruction	Description
`SET SERVEROUTPUT ON;`	Activer les commandes d'affichage de messages dans la console (à exécuter 1 fois par fichier au démarrage de SQL Developer)
`DBMS_OUTPUT.PUT_LINE('texte');`	Affiche un texte dans la console avec un retour à la ligne