Cours Oracle SQL

Base de données Avancées

Pr. Lamia ZIAD
École Supérieure de Technologie d'Essaouira

Contents

Chapter 1

Introduction à SQL & Instruction SELECT

1.1 Présentation générale de SQL

SQL (Structured Query Language) est le langage normalisé utilisé pour communiquer avec un système de gestion de base de données relationnelle (SGBDR). Il permet :

- la définition des objets : tables, vues, index, contraintes (DDL),
- la manipulation des données : INSERT, UPDATE, DELETE (DML),
- le contrôle des transactions : COMMIT, ROLLBACK, SAVEPOINT (TCL),
- la gestion des droits et privilèges (DCL).

Dans ce cours Oracle SQL, nous nous concentrerons principalement sur :

- l'interrogation des données (SELECT),
- les filtres (WHERE),
- les expressions et fonctions,
- les jointures,
- les sous-interrogations,
- le LDD, LMD et les contraintes,
- le contrôle des transactions.

1.2 Environnement Oracle : iSQL*Plus

iSQL*Plus est l'interface Web permettant d'exécuter des commandes SQL dans Oracle. Elle comporte trois zones :

- **Workspace** : zone où l'utilisateur saisit ses instructions SQL,
- **Execute** : bouton permettant l'exécution de la commande,
- **Output Pane** : zone affichant le résultat ou les messages d'erreur.

Lorsque l'utilisateur exécute une commande :

- si la commande est correcte : le résultat apparaît immédiatement,
- si une erreur est détectée : Oracle précise le type d'erreur et la ligne concernée.

1.3 L'instruction SELECT

SELECT est la commande la plus utilisée dans SQL. Elle permet d'afficher des données provenant d'une ou plusieurs tables.

Syntaxe minimale

```
SELECT colonne1, colonne2
FROM nom_table;
```

- **SELECT** détermine les colonnes à afficher ;
- **FROM** indique la ou les tables où se trouvent les données.

1.4 Projection : choix des colonnes

La projection consiste à sélectionner uniquement les colonnes nécessaires.

```
SELECT last_name FROM employees;
SELECT last_name, salary FROM employees;
SELECT * FROM employees;
```

L'utilisation de `*` doit être évitée en production car elle peut nuire aux performances et à la lisibilité.

1.5 Alias de colonnes

Les alias permettent de renommer une colonne dans le résultat.

```
SELECT last_name AS Nom, salary AS Salaire FROM employees;
SELECT last_name AS "Nom Employé" FROM employees;
```

1.6 Expressions et calculs

SQL autorise l'utilisation d'opérations arithmétiques dans SELECT.

```
SELECT first_name, salary, salary*12 AS salaire_annuel FROM employees;
SELECT 10 + 5 * 2 FROM dual;     -- 20
SELECT (10 + 5) * 2 FROM dual;   -- 30
```

1.7 DISTINCT : éliminer les doublons

```
SELECT DISTINCT department_id FROM employees;
```

1.8 Concaténation

Oracle utilise l'opérateur `||` :

```
SELECT first_name || ' ' || last_name AS full_name
FROM employees;
```

1.9 Règles importantes

- SQL n'est pas sensible à la casse pour les mots-clés,
- les chaînes doivent être entourées d'apostrophes simples,
- chaque instruction doit se terminer par un point-virgule.

Chapter 2

Clause WHERE & ORDER BY

2.1 Rôle de la clause WHERE

La clause WHERE permet de filtrer les lignes renvoyées par une requête SQL. Sans WHERE, toutes les lignes de la table sont retournées.

Exemples :

```
SELECT last_name, salary
FROM employees;
```

```
SELECT last_name, salary
FROM employees
WHERE salary > 3000;
```

2.2 Opérateurs de comparaison

Les opérateurs compatibles avec la clause WHERE sont :

- = (égal)
- <> (différent)
- >, < (supérieur, inférieur)
- >=, <= (supérieur ou égal, inférieur ou égal)

Exemples :

```
SELECT last_name FROM employees WHERE department_id = 50;
SELECT last_name FROM employees WHERE salary <= 4000;
```

2.3 Comparaison de valeurs textuelles

Les chaînes de caractères doivent être entourées de guillemets simples.

```
SELECT * FROM employees WHERE last_name = 'King';
SELECT * FROM employees WHERE job_id = 'IT_PROG';
```

2.4 Opérateur BETWEEN

L'opérateur BETWEEN permet de tester si une valeur se trouve dans un intervalle inclusif.

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 3000 AND 6000;
```

Equivalent à :

```
salary >= 3000 AND salary <= 6000
```

2.5 Opérateur IN

IN teste si une valeur appartient à une liste.

```
SELECT last_name, department_id
FROM employees
WHERE department_id IN (10, 20, 30);
```

2.6 Opérateur LIKE

LIKE permet la recherche par motif (pattern).

- % : chaîne de caractères variable
- _ : un seul caractère

Exemples :

```
SELECT last_name FROM employees WHERE last_name LIKE 'K%';
SELECT last_name FROM employees WHERE last_name LIKE '%n';
SELECT last_name FROM employees WHERE last_name LIKE '_a%';
```

2.7 IS NULL / IS NOT NULL

Pour tester les valeurs NULL :

```
SELECT * FROM employees WHERE commission_pct IS NULL;
SELECT * FROM employees WHERE commission_pct IS NOT NULL;
```

Attention : L'expression `= NULL` est incorrecte.

2.8 Opérateurs logiques

AND

```
SELECT * FROM employees
WHERE department_id = 50 AND salary > 3000;
```

OR

```
SELECT * FROM employees
WHERE job_id = 'IT_PROG' OR job_id = 'ST_CLERK';
```

NOT

```
SELECT * FROM employees
WHERE NOT job_id = 'IT_PROG';
```

2.9 Priorité des opérateurs

Ordre d'évaluation par défaut :

1. NOT
2. AND
3. OR

On peut modifier l'ordre avec des parenthèses.

```
SELECT * FROM employees
WHERE job_id = 'IT_PROG'
OR (department_id = 50 AND salary > 3000);
```

2.10 ORDER BY : tri des résultats

ORDER BY permet de trier les résultats :

Tri ascendant (par défaut)

```
SELECT last_name, salary
FROM employees
ORDER BY salary;
```

Tri descendant

```
ORDER BY salary DESC;
```

Tri sur plusieurs colonnes

```
ORDER BY department_id, salary DESC;
```

2.11 Tri sur alias

Un alias défini dans SELECT peut être utilisé dans ORDER BY.

```
SELECT last_name, salary*12 AS annual_salary
FROM employees
ORDER BY annual_salary DESC;
```

Chapter 3

Fonctions SQL : Texte, Nombres, Dates, Conversion, NULL

3.1 Introduction aux fonctions SQL

Une fonction SQL permet de transformer ou manipuler une valeur. Il existe deux grandes catégories :

- **Fonctions monolignes** : agissent sur une seule ligne et renvoient une valeur,
- **Fonctions multilignes** : agissent sur plusieurs lignes (voir Chapitre 5).

Les fonctions monolignes peuvent être utilisées dans :

- SELECT,
- WHERE,
- ORDER BY,
- GROUP BY,
- HAVING.

Ce chapitre traite uniquement des fonctions monolignes.

3.2 Fonctions sur les chaînes de caractères

UPPER, LOWER, INITCAP

- **UPPER** : met en majuscule,
- **LOWER** : met en minuscule,
- **INITCAP** : met en majuscule la première lettre de chaque mot.

```
SELECT UPPER(last_name), LOWER(first_name), INITCAP('oracle SQL')
FROM employees;
```

LENGTH

Retourne la longueur d'une chaîne.

```
SELECT last_name, LENGTH(last_name)
FROM employees;
```

SUBSTR

Extrait une sous-chaîne.

```
SUBSTR(texte, position, longueur)
```

```
SELECT SUBSTR('INFORMATIQUE', 1, 4) FROM dual;   -- INFO
SELECT SUBSTR(last_name, 2, 3) FROM employees;
```

INSTR

Retourne la position d'un caractère ou d'une sous-chaîne.

```
SELECT INSTR('PROGRAMMATION', 'A') FROM dual;   -- 6
```

CONCAT et opérateur ||

```
SELECT CONCAT(first_name, last_name) FROM employees;
SELECT first_name || ' ' || last_name FROM employees;
```

L'opérateur `||` est préférable car il est plus flexible.

3.3 Fonctions numériques

ROUND

Arrondit un nombre.

```
SELECT ROUND(45.926, 2) FROM dual;   -- 45.93
```

TRUNC

Tronque un nombre (supprime les décimales sans arrondir).

```
SELECT TRUNC(45.926, 2) FROM dual;   -- 45.92
```

MOD

Retourne le reste d'une division.

```
SELECT MOD(10, 3) FROM dual;   -- 1
```

3.4 Fonctions sur les dates

Oracle stocke une date avec un composant **date** + **heure**. Fonctions utiles :

SYSDATE

Renvoie la date et l'heure du serveur Oracle.

```
SELECT SYSDATE FROM dual;
```

MONTHS_BETWEEN

Calcule le nombre de mois entre deux dates.

```
SELECT MONTHS_BETWEEN(SYSDATE, hire_date)
FROM employees;
```

ADD_MONTHS

```
SELECT ADD_MONTHS(SYSDATE, 6) FROM dual;
```

NEXT_DAY et LAST_DAY

```
SELECT NEXT_DAY(SYSDATE, 'MONDAY') FROM dual;
SELECT LAST_DAY(SYSDATE) FROM dual;
```

3.5 Fonctions de conversion

TO_CHAR

Convertit un nombre ou une date en texte.

```
SELECT TO_CHAR(hire_date, 'DD Month YYYY')
FROM employees;

SELECT TO_CHAR(salary, '999,999.99')
FROM employees;
```

TO_NUMBER

```
SELECT TO_NUMBER('5000') + 200 FROM dual;
```

TO_DATE

```
SELECT TO_DATE('2024-05-31', 'YYYY-MM-DD')
FROM dual;
```

3.6 Gestion des valeurs NULL

Les valeurs NULL représentent une **absence de valeur**, différente de zéro ou d'une chaîne vide.

NVL

Remplace une valeur NULL.

```
SELECT NVL(commission_pct, 0)
FROM employees;
```

NVL2

```
SELECT NVL2(commission_pct,
'A une commission',
'Pas de commission')
FROM employees;
```

COALESCE

Retourne la première valeur non-NULL d'une liste.

```
SELECT COALESCE(commission_pct, bonus, 0)
FROM employees;
```

Chapter 4

Les Jointures : INNER, OUTER, SELF JOIN, USING, NATURAL JOIN

4.1 Introduction aux jointures

Dans une base de données relationnelle, les informations sont réparties dans plusieurs tables afin de réduire la redondance et d'améliorer la cohérence. Les jointures permettent de :

- combiner des données provenant de plusieurs tables,
- relier des tables grâce aux clés primaires et étrangères,
- présenter une vue complète et cohérente des données.

Exemple classique :

- la table `EMPLOYEES` contient : nom, prénom, salaire, id département,
- la table `DEPARTMENTS` contient : id département, nom du département.

Pour afficher le nom de l'employé **et** son département, une jointure est nécessaire.

4.2 Inner Join (Jointure interne)

La jointure interne retourne uniquement les lignes qui ont des correspondances dans les deux tables.

Syntaxe ANSI recommandée

```
SELECT e.last_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

Ancienne syntaxe Oracle (WHERE)

```
SELECT e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

Cette syntaxe, bien que fonctionnelle, est moins lisible et rend les requêtes complexes plus difficiles à maintenir.

4.3 Alias de tables

Les alias permettent de simplifier la syntaxe et d'améliorer la lisibilité :

```
SELECT e.last_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

4.4 Jointures sur plusieurs tables

Il est possible de joindre 3 tables ou plus :

```
SELECT e.last_name, d.department_name, l.city
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations  l ON d.location_id   = l.location_id;
```

4.5 Jointures externes (OUTER JOIN)

Les jointures externes permettent d'afficher les lignes **même si aucune correspondance n'existe** dans l'autre table.

Trois types existent :

- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

LEFT OUTER JOIN

```
SELECT e.last_name, d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;
```

Affiche tous les employés, même sans département.

RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_name
FROM employees e
RIGHT JOIN departments d
ON e.department_id = d.department_id;
```

Affiche tous les départements, même sans employés.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON e.department_id = d.department_id;
```

Affiche toutes les lignes des deux tables.

4.6 Jointure externe Oracle : syntaxe (+)

Ancienne méthode Oracle (désormais obsolète) :

```
SELECT e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id(+);
```

Cette méthode est déconseillée dans de nouveaux développements.

4.7 Self Join (Auto-jointure)

Une auto-jointure permet de relier une table à elle-même. Très utilisée pour représenter des relations hiérarchiques (ex : employés → managers).

```
SELECT e.last_name AS employee,
m.last_name AS manager
FROM employees e
JOIN employees m
ON e.manager_id = m.employee_id;
```

4.8 Jointure avec conditions

Il est possible d'ajouter des conditions supplémentaires après la jointure :

```
SELECT e.last_name, d.department_name, e.salary
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.salary > 5000
ORDER BY d.department_name;
```

4.9 Natural Join et USING

NATURAL JOIN

Oracle relie automatiquement les colonnes de même nom dans les deux tables.

```
SELECT *
FROM employees
NATURAL JOIN departments;
```

Danger : si plusieurs colonnes portent le même nom, le résultat peut être imprévisible.

USING

Permet de préciser la colonne commune.

```
SELECT last_name, department_name
FROM employees
JOIN departments USING (department_id);
```

Chapter 5

Fonctions Multilignes, GROUP BY et HAVING

5.1 Introduction

Les fonctions multilignes (appelées aussi **fonctions d'agrégation**) permettent d'appliquer un calcul sur un groupe de lignes et de renvoyer une seule valeur en résultat. Elles sont indispensables lorsqu'on souhaite résumer ou regrouper des données.

Exemples d'utilisation typique :

- total des salaires,
- moyenne des salaires par département,
- plus ancienne ou plus récente date d'embauche,
- nombre d'employés dans chaque service.

Important : Les fonctions d'agrégations ignorent les valeurs NULL (sauf COUNT(*)).

5.2 Principales fonctions d'agrégation

- **SUM(expr)** : somme des valeurs,
- **AVG(expr)** : moyenne,
- **MIN(expr)** : minimum,
- **MAX(expr)** : maximum,

- **COUNT(expr)** : nombre de valeurs non NULL,
- **COUNT(*)** : nombre total de lignes.

Exemples :

```
SELECT SUM(salary) FROM employees;
SELECT AVG(salary) FROM employees;
SELECT MIN(hire_date), MAX(hire_date) FROM employees;
SELECT COUNT(*) FROM employees;
```

5.3 Différence entre COUNT(*) et COUNT(colonne)

- **COUNT(*)** compte toutes les lignes, y compris celles contenant des NULL,
- **COUNT(colonne)** ignore les lignes où la colonne contient NULL.

```
SELECT COUNT(*) FROM employees;              -- total employes
SELECT COUNT(commission_pct) FROM employees; -- seulement ceux avec commission
```

5.4 La clause GROUP BY

GROUP BY regroupe les lignes selon une ou plusieurs colonnes afin d'appliquer une fonction d'agrégation à chaque groupe.

Exemple : salaire moyen par département

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

Règles importantes :

- Toute colonne dans SELECT doit apparaître :
  - soit dans la clause GROUP BY,
  - soit dans une fonction d'agrégation.
- GROUP BY peut utiliser plusieurs colonnes.

Exemple :

```
SELECT department_id, job_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id;
```

5.5 Ordre logique d'exécution des clauses SQL

L'ordre dans lequel Oracle traite les clauses est :

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Cela permet de comprendre pourquoi HAVING agit **après** l'agrégation.

5.6 La clause HAVING

HAVING filtre les groupes après l'application du GROUP BY. Elle est utilisée uniquement avec les fonctions d'agrégation.

Exemple : départements où le salaire moyen dépasse 5000

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 5000;
```

Comparaison WHERE vs HAVING

- WHERE filtre les lignes (avant regroupement),
- HAVING filtre les groupes (après regroupement).

Exemple complet :

```
SELECT department_id, COUNT(*) AS nb_employes
FROM employees
WHERE salary > 3000
GROUP BY department_id
HAVING COUNT(*) >= 3
ORDER BY department_id;
```

5.7 NULL et fonctions d'agrégation

Les valeurs NULL sont ignorées dans :

- SUM
- AVG
- MIN
- MAX
- COUNT(colonne)

Exception : COUNT(*) compte toutes les lignes.
Exemple :

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

5.8 GROUP BY sur des dates

Exemple : nombre d'employés embauchés par année.

```
SELECT TO_CHAR(hire_date, 'YYYY') AS annee,
COUNT(*)
FROM employees
GROUP BY TO_CHAR(hire_date, 'YYYY')
ORDER BY annee;
```

Chapter 6

Les Sous-interrogations (Subqueries)

6.1 Introduction

Une sous-interrogation (subquery) est une requête imbriquée à l'intérieur d'une autre requête. Elle est exécutée en premier, et son résultat est utilisé par la requête principale.

Les sous-interrogations permettent de :

- comparer une valeur à un résultat calculé,
- travailler avec des valeurs agrégées,
- filtrer des résultats en utilisant d'autres tables,
- remplacer des jointures dans certains cas.

Position possible des sous-requêtes :

Elles peuvent être utilisées dans :

- SELECT
- FROM
- WHERE
- HAVING

Les plus courantes sont celles placées dans **WHERE** et **HAVING**.

6.2 Sous-interrogations scalaires

Une sous-interrogation **scalaire** renvoie une seule valeur. Elles sont souvent utilisées pour comparer une colonne avec un calcul.

Exemple :

```
SELECT last_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Interprétation : afficher les employés qui gagnent plus que la moyenne.

6.3 Sous-interrogations multivaluées (IN, ANY, ALL)

Ces sous-requêtes renvoient plusieurs lignes.

IN

```
SELECT last_name, department_id
FROM employees
WHERE department_id IN
(SELECT department_id
FROM departments
WHERE location_id = 1700);
```

ANY

La condition est vraie si la comparaison est vraie pour **au moins une valeur**.

```
SELECT last_name, salary
FROM employees
WHERE salary > ANY
(SELECT salary FROM employees WHERE department_id = 50);
```

Interprétation : salaire supérieur à *au moins* un salaire du département 50.

ALL

La condition doit être vraie pour **toutes les valeurs** retournées par la sous-requête.

```
SELECT last_name, salary
FROM employees
WHERE salary > ALL
(SELECT salary FROM employees WHERE department_id = 50);
```

Interprétation : salaire supérieur à *tous* les salaires du département 50.

6.4 Sous-interrogations corrélées

La sous-interrogation corrélée dépend de la ligne courante de la requête principale. Elle est exécutée une fois par ligne.

Exemple :

```
SELECT e.last_name, e.salary
FROM employees e
WHERE e.salary >
(SELECT AVG(salary)
FROM employees
WHERE department_id = e.department_id);
```

Interprétation : employés qui gagnent plus que la moyenne de leur département.

6.5 Sous-interrogations dans la clause FROM (Vue en ligne)

Une sous-requête peut être placée dans FROM : elle devient alors une **vue en ligne** (inline view).

```
SELECT department_id, avg_salary
FROM
(SELECT department_id,
AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id)
WHERE avg_salary > 5000;
```

Utilité : structurer une requête complexe en plusieurs parties plus lisibles.

6.6 Sous-interrogations dans HAVING

```
SELECT department_id, COUNT(*) AS nb
FROM employees
GROUP BY department_id
HAVING COUNT(*) >
(SELECT AVG(cnt)
FROM
(SELECT COUNT(*) AS cnt
FROM employees
GROUP BY department_id));
```

Interprétation : départements ayant plus d'employés que la moyenne des départements.

6.7 Sous-interrogations dans SELECT

Une sous-requête dans SELECT doit être scalaire.

```
SELECT last_name,
(SELECT department_name
FROM departments d
WHERE d.department_id = e.department_id) AS dept
FROM employees e;
```

6.8 Règles importantes

- Une sous-requête doit toujours être entre parenthèses.
- Les sous-requêtes scalaires doivent renvoyer une seule valeur.
- ALL, ANY et IN ne fonctionnent qu'avec des sous-requêtes multivaluées.
- Une sous-requête corrélée ne peut pas être exécutée seule.

Chapter 7

Le Langage de Manipulation des Données (LMD) : INSERT, UPDATE, DELETE

7.1 Introduction

Le Langage de Manipulation des Données (LMD ou DML : Data Manipulation Language) permet de modifier le contenu des tables d'une base de données.

Les instructions principales sont :

- **INSERT** : ajouter des lignes dans une table,
- **UPDATE** : modifier des lignes existantes,
- **DELETE** : supprimer des lignes.

Les instructions DML ne sont définitives qu'après un **COMMIT**. Tant que COMMIT n'est pas exécuté :

- les modifications ne sont visibles que par l'utilisateur courant,
- elles peuvent être annulées avec **ROLLBACK**.

7.2 INSERT : ajouter de nouvelles lignes

Syntaxe 1 : insertion complète

```
INSERT INTO table_name
```

```
VALUES (val1, val2, val3, ...);
```

Les valeurs doivent respecter l'ordre des colonnes dans la table.

Syntaxe 2 : insertion partielle

```
INSERT INTO table_name (col1, col2, col3)
VALUES (val1, val2, val3);
```

Avantage : on peut omettre certaines colonnes (si valeur par défaut ou NULL).

Exemples

```
INSERT INTO departments
(department_id, department_name, location_id)
VALUES (300, 'Support Technique', 1700);
```

INSERT *avec sous-interrogation*

```
INSERT INTO new_employees (employee_id, last_name, salary)
SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 50;
```

Permet de copier des lignes d'une table à une autre.

7.3 UPDATE : modifier des lignes existantes

Syntaxe

```
UPDATE table_name
SET col1 = val1,
col2 = val2
WHERE condition;
```

Attention : Sans clause WHERE, toutes les lignes seront modifiées.

Exemples :

```
UPDATE employees
SET salary = salary * 1.10
WHERE department_id = 80;
```

UPDATE basé sur une sous-requête

```
UPDATE employees
SET salary = salary + 500
WHERE employee_id IN
(SELECT employee_id
FROM employees
WHERE commission_pct IS NOT NULL);
```

7.4 DELETE : supprimer des lignes

Syntaxe

```
DELETE FROM table_name
WHERE condition;
```

Sans WHERE : toutes les lignes seront supprimées.
Exemples :

```
DELETE FROM employees
WHERE department_id = 190;
```

DELETE avec sous-interrogation

```
DELETE FROM employees
WHERE employee_id IN
(SELECT employee_id
FROM employees
WHERE salary < 2000);
```

7.5 Remarques importantes sur INSERT, UPDATE, DELETE

- Une instruction DML ne devient permanente qu'après COMMIT.
- Une instruction avec erreur ne modifie aucune donnée.
- Les sous-requêtes dans les instructions DML permettent des modifications conditionnelles complexes.

- On peut vérifier les lignes impactées avec :

```
SELECT COUNT(*) FROM employees;
```

7.6 TRUNCATE

Même si TRUNCATE fait partie du langage de définition de données (LDD), il est souvent présenté avec DELETE car il supprime des données.

```
TRUNCATE TABLE table_name;
```

Différences entre TRUNCATE et DELETE :

- TRUNCATE est plus rapide,
- TRUNCATE ne génère pas de rollback possible,
- TRUNCATE réinitialise l'espace alloué.

Chapter 8

Les Transactions : COMMIT, ROLLBACK et SAVEPOINT

8.1 Introduction

Une **transaction** représente une unité logique de travail composée d'une ou plusieurs instructions SQL. Les transactions permettent de garantir :

- la cohérence des données,
- la sécurité en cas d'erreurs,
- la possibilité d'annuler ou valider les modifications.

En Oracle, une transaction commence :

- soit après un COMMIT ou un ROLLBACK,
- soit dès la première instruction DML (INSERT, UPDATE, DELETE) exécutée.

Une transaction se termine lorsque l'utilisateur exécute :

- **COMMIT** : valider définitivement,
- **ROLLBACK** : annuler toutes les modifications,
- **DDL** (CREATE, ALTER, DROP) : COMMIT implicite.

8.2 COMMIT : valider la transaction

COMMIT enregistre définitivement toutes les modifications effectuées depuis le début de la transaction.

Effets d'un COMMIT :

- les changements deviennent permanents,
- les autres utilisateurs peuvent voir les modifications,
- les verrous posés sur les lignes sont libérés.

Exemple :

```
UPDATE employees
SET salary = salary + 300;

COMMIT;
```

8.3 ROLLBACK : annuler la transaction

ROLLBACK annule toutes les modifications effectuées depuis le début de la transaction ou depuis le dernier SAVEPOINT.

Effets du ROLLBACK :

- aucune modification n'est conservée,
- les verrous sont libérés,
- la table revient à l'état précédent.

Exemple :

```
DELETE FROM employees
WHERE department_id = 50;

ROLLBACK;  -- rétablir les données supprimées
```

8.4 SAVEPOINT : point de sauvegarde intermédiaire

SAVEPOINT permet de définir un point intermédiaire dans une transaction.

Syntaxe :

```
SAVEPOINT pointA;
```

On peut ensuite annuler uniquement une partie de la transaction :

```
ROLLBACK TO pointA;
```

Exemple complet

```
UPDATE employees SET salary = salary + 500
WHERE department_id = 80;

SAVEPOINT s1;

UPDATE employees SET salary = salary + 500
WHERE department_id = 50;

ROLLBACK TO s1;   -- annule seulement la deuxième modification

COMMIT;           -- valide la modification sur dept 80
```

8.5 Transactions implicites

Certaines instructions provoquent un COMMIT automatique :

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- TRUNCATE TABLE

Ces commandes font partie du LDD et valident (COMMIT) automatiquement toute transaction ouverte.

8.6 Comportement en environnement multi-utilisateur

Oracle garantit la cohérence même lorsque plusieurs utilisateurs accèdent aux mêmes données.

Verrouillage des lignes

Lorsqu'un utilisateur exécute une instruction DML :

- les lignes modifiées sont verrouillées,
- aucun autre utilisateur ne peut les modifier,
- la lecture reste possible (mode lecture cohérente).

Isolation des transactions

Chaque utilisateur travaille dans sa propre transaction. Tant que COMMIT n'est pas exécuté, les autres utilisateurs ne voient pas les modifications.

Exemple :

- utilisateur A augmente un salaire,
- utilisateur B consulte la table → ne voit aucune modification,
- A exécute COMMIT → B voit la mise à jour.

Chapter 9

Le Langage de Définition des Données (LDD) : CREATE, ALTER, DROP, TRUNCATE

9.1 Introduction au LDD

Le Langage de Définition des Données (LDD — Data Definition Language) permet de créer, modifier ou supprimer les objets d'une base Oracle.

Les principales instructions sont :

- **CREATE** : créer un objet,
- **ALTER** : modifier un objet existant,
- **DROP** : supprimer un objet,
- **TRUNCATE** : vider une table rapidement.

Les commandes LDD provoquent un **COMMIT implicite**. Toutes les transactions non validées sont automatiquement enregistrées.

9.2 CREATE TABLE

La création d'une table nécessite de définir :

- le nom de la table,

- les colonnes,
- les types de données,
- éventuellement des contraintes (PRIMARY KEY, UNIQUE, NOT NULL...).

Syntaxe générale

```
CREATE TABLE table_name (
column1 datatype constraint,
column2 datatype constraint,
...
);
```

Exemple

```
CREATE TABLE employees2 (
employee_id     NUMBER(6),
last_name       VARCHAR2(25) NOT NULL,
salary          NUMBER(8,2),
hire_date       DATE,
department_id   NUMBER(4)
);
```

9.3 Types de données les plus courants

- **VARCHAR2(n)** : chaîne variable (max. n caractères)
- **CHAR(n)** : chaîne fixe (n caractères)
- **NUMBER(p, s)** :
  - p = précision,
  - s = nombre de décimales.
- **DATE** : date + heure
- **CLOB** : texte volumineux
- **BLOB** : données binaires

9.4 ALTER TABLE : modifier une table

ALTER TABLE permet d'ajouter, modifier ou supprimer des colonnes et des contraintes.

Ajouter une colonne

```
ALTER TABLE employees2
ADD (email VARCHAR2(50));
```

Modifier une colonne

```
ALTER TABLE employees2
MODIFY (salary NUMBER(10,2));
```

Supprimer une colonne

```
ALTER TABLE employees2
DROP COLUMN email;
```

Renommer une colonne

```
ALTER TABLE employees2
RENAME COLUMN last_name TO family_name;
```

9.5 Renommer une table

```
RENAME employees2 TO staff;
```

9.6 DROP TABLE : supprimer la table

Supprime la table et toutes les données :

```
DROP TABLE employees2;
```

Attention : opération irréversible.

DROP TABLE ... CASCADE CONSTRAINTS

```
DROP TABLE departments CASCADE CONSTRAINTS;
```

Supprime la table même si d'autres objets dépendent d'elle.

9.7 TRUNCATE TABLE : vider la table

```
TRUNCATE TABLE employees2;
```

Différences TRUNCATE vs DELETE

- TRUNCATE est beaucoup plus rapide,
- TRUNCATE effectue un COMMIT implicite,
- DELETE permet un ROLLBACK, TRUNCATE non,
- DELETE peut avoir une clause WHERE, TRUNCATE non.

9.8 CREATE TABLE AS SELECT (CTAS)

Permet de créer une table en copiant la structure et/ou les données d'une autre table.

```
CREATE TABLE emp_copy AS
SELECT employee_id, last_name, salary
FROM employees;
```

Particularités

- les contraintes ne sont pas copiées,
- seules les données et les noms de colonnes sont dupliqués.

Chapter 10

Les Contraintes (Constraints)

10.1 Introduction

Les contraintes garantissent l'intégrité des données dans une base relationnelle. Elles empêchent l'insertion, la modification ou la suppression de données invalides.

Les contraintes Oracle sont :

- **PRIMARY KEY**
- **FOREIGN KEY**
- **UNIQUE**
- **CHECK**
- **NOT NULL**

Elles peuvent être créées :

- lors de la création de la table (définition inline),
- après la création (ALTER TABLE).

Les contraintes peuvent être :

- activées (ENABLE),
- désactivées (DISABLE),
- nommées explicitement.

10.2 La contrainte NOT NULL

Une colonne NOT NULL doit obligatoirement recevoir une valeur.

Définition lors de CREATE TABLE

```
last_name VARCHAR2(30) NOT NULL
```

Ajout via ALTER TABLE

```
ALTER TABLE employees
MODIFY (last_name NOT NULL);
```

10.3 La contrainte UNIQUE

La contrainte UNIQUE impose que les valeurs d'une colonne (ou groupe de colonnes) soient toutes différentes.

```
email VARCHAR2(50) UNIQUE
```

Contrainte nommée

```
CONSTRAINT email_uk UNIQUE (email)
```

Les valeurs NULL sont autorisées (une seule par colonne).

10.4 La clé primaire (PRIMARY KEY)

La clé primaire identifie de manière unique chaque ligne d'une table. Caractéristiques :

- unique,
- NOT NULL obligatoire,
- une seule PRIMARY KEY par table,
- peut être composée de plusieurs colonnes.

Exemple

```
CONSTRAINT emp_pk PRIMARY KEY (employee_id)
```

10.5 La clé étrangère (FOREIGN KEY)

La clé étrangère relie deux tables ensemble. Elle garantit l'intégrité référentielle.

Syntaxe

```
CONSTRAINT emp_dept_fk
FOREIGN KEY (department_id)
REFERENCES departments(department_id)
```

Restrictions

- on ne peut pas insérer une valeur inexistante dans la table parent,
- on ne peut pas supprimer une valeur parent utilisée dans la table enfant.

Options : ON DELETE

```
ON DELETE CASCADE      -- supprime aussi les enfants
ON DELETE SET NULL      -- met la FK à NULL
```

10.6 La contrainte CHECK

CHECK impose une condition logique à respecter.

```
salary NUMBER(8,2)
CONSTRAINT salary_ck CHECK (salary > 0)
```

Exemples :

```
CHECK (gender IN ('H','F'))
CHECK (hire_date >= '01-JAN-2000')
```

10.7 Contraintes dans CREATE TABLE

Contraintes en ligne (inline)

```
employee_id NUMBER CONSTRAINT emp_pk PRIMARY KEY
```

Contraintes hors ligne (out-of-line)

```
CONSTRAINT emp_dept_fk
FOREIGN KEY (department_id)
REFERENCES departments(department_id)
```

10.8 Ajout de contraintes après création

```
ALTER TABLE employees
ADD CONSTRAINT emp_salary_ck
CHECK (salary > 0);
```

10.9 Désactivation et activation des contraintes

```
ALTER TABLE employees DISABLE CONSTRAINT emp_pk;
ALTER TABLE employees ENABLE  CONSTRAINT emp_pk;
```

Désactiver une contrainte permet d'insérer des données temporairement incohérentes (à utiliser avec prudence).

10.10 Suppression d'une contrainte

```
ALTER TABLE employees
DROP CONSTRAINT emp_dept_fk;
```

10.11 Contraintes différables (DEFERRABLE)

Certaines contraintes peuvent être évaluées :

- immédiatement (IMMEDIATE),
- à la fin de la transaction (DEFERRED).

Syntaxe

```
ALTER TABLE employees
ADD CONSTRAINT emp_sal_ck
CHECK (salary > 0)
```

```
DEFERRABLE INITIALLY DEFERRED;
```