المدرسة العليا للتكنولوجيا – الصويرة
ⵜⵉⵎⵙⵙⵓⵔⵉⵜ ⵜⴰⵏⵎⵓⵔⵜ ⵏ ⵜⵙⵙⵏⵉⵡⵉⵏ - ⵜⴰⵡⵉⵔⴰ
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - ESSAOUIRA

Cours Oracle SQL

Filière : Informatique décisionnelle et sciences de données

Base de Données Avancées pour la Science de Données

Pr. Lamia ZIAD
École Supérieure de Technologie d'Essaouira

Contents

1. Introduction à SQL pour la Science des Données

Science des Données

SQL est un outil essentiel pour tout Data Scientist ou Data Analyst. Il permet :

- d'extraire des données depuis des systèmes transactionnels ou décisionnels,
- de nettoyer et préparer les données avant les étapes de Machine Learning,
- d'effectuer des transformations (Feature Engineering),
- de créer des vues et tables utilisées dans les pipelines ETL,
- de réaliser des analyses statistiques rapides directement sur le SGBD.

1.1 Présentation générale de SQL

Le SQL (Structured Query Language) est un langage normalisé pour interagir avec un SGBDR (Oracle, MySQL, PostgreSQL...). Dans les métiers de la Science des Données :

- Il sert à **structurer les datasets**.
- Il permet d'automatiser la **préparation des données**.
- Il est le point de départ de toute analyse ou entraînement de modèle.

SQL comprend 5 sous-langages :

- **DDL** : définition (CREATE, ALTER...)

- **DML** : manipulation (INSERT, UPDATE, DELETE)
- **DQL** : interrogation (SELECT)
- **TCL** : transactions (COMMIT, ROLLBACK)
- **DCL** : privilèges

Science des Données

Avant de construire un modèle prédictif, un Data Scientist doit :

- sélectionner les features,
- éliminer les valeurs aberrantes,
- agréger des données,
- fusionner plusieurs sources.

Toutes ces étapes peuvent être réalisées efficacement en SQL.

1.2 Environnement Oracle : iSQL*Plus

iSQL*Plus offre :

- un **workspace** pour écrire les requêtes,
- un bouton **Execute** pour les appliquer,
- un **output pane** pour afficher résultats et erreurs.

Attention

Une erreur dans une requête SQL peut fausser :

- une analyse statistique,
- un modèle prédictif,
- ou un pipeline de production.

1.3 L'instruction SELECT

La commande SELECT permet :

- d'extraire des données,
- de créer des features dérivées,
- de filtrer et trier les lignes,
- d'effectuer des calculs statistiques simples.

Syntaxe minimale :

```
SELECT colonne1, colonne2
FROM table;
```

Projection

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

Astuce

Toujours nommer explicitement les colonnes :
- meilleure lisibilité - meilleure performance - évite les colonnes inutiles dans les datasets ML

1.4 Alias et Calculs

```
SELECT salary,
salary*12 AS annual_salary,
UPPER(last_name) AS last_upper
FROM employees;
```

1.5 DISTINCT

```
SELECT DISTINCT job_id FROM employees;
```

1.6 Concaténation

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

1.7 Règles de base

- SQL est insensible à la casse,
- les chaînes sont entre apostrophes,
- chaque requête se termine par ; en Oracle.

2. Clause WHERE et Nettoyage de Données

Science des Données

La clause **WHERE** est l'un des outils les plus importants pour :

- nettoyer les données (**data cleaning**),
- détecter les valeurs aberrantes (**outliers**),
- filtrer pour créer des datasets cohérents,
- sélectionner un sous-ensemble pertinent avant un modèle ML,
- préparer les données pour l'EDA (Exploratory Data Analysis).

2.1 Rôle de la clause WHERE

Sans WHERE, la totalité des données brutes est renvoyée — ce qui est rarement souhaitable en Data Science.

Exemple simple :

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

Avec filtre :

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

Attention

Un dataset mal filtré peut entraîner :

- un modèle biaisé,
- une performance instable,
- une convergence difficile,
- des conclusions statistiques incorrectes.

2.2 Opérateurs de comparaison

Ces opérateurs sont essentiels pour structurer les filtres :

- = (égalité)
- <> (différence)
- >, < (supérieur, inférieur)
- >=, <= (supérieur ou égal, inférieur ou égal)

Exemple :

```
SELECT last_name
FROM employees
WHERE department_id = 50;
```

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

2.3 Comparaison textuelle

Les chaînes doivent être entourées d'apostrophes :

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

2.4 Opérateur BETWEEN

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

Très utilisé pour détecter les "ranges" cohérents dans les variables quantitatives.

2.5 Opérateur IN

Pour tester si une valeur appartient à une liste :

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

Science des Données

IN est très utilisé pour :

- filtrer des catégories,
- exclure certaines classes,
- gérer les niveaux d'une variable qualitative.

2.6 Recherche textuelle : LIKE

LIKE permet des recherches par motifs (patterns) :

- % : plusieurs caractères,
- _ : exactement un caractère.

```
SELECT last_name
FROM employees
WHERE last_name LIKE 'K%';      -- commence par K

SELECT last_name
```

```
FROM employees
WHERE last_name LIKE '%n';      -- finit par n

SELECT last_name
FROM employees
WHERE last_name LIKE '_a%';     -- 2e lettre = a
```

2.7 IS NULL / IS NOT NULL

En Data Science, NULL $\neq$ 0 $\neq$ " C'est un vrai signal de données manquantes.

```
SELECT *
FROM employees
WHERE commission_pct IS NULL;
```

Attention

Une mauvaise gestion de NULL peut fausser :

- les moyennes,
- les agrégations,
- les corrélations,
- la performance des modèles ML.

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 :

1. NOT
2. AND
3. OR

Exemple :

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

Astuce

Utiliser des parenthèses explicites :
- évite les erreurs de logique - rend les requêtes reproductibles - facilite l'audit des pipelines

2.10 Tri des résultats : ORDER BY

Tri ascendant

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

Tri descendant

```
ORDER BY salary DESC;
```

Tri multiple

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

Tri sur alias

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

Science des Données

ORDER BY est utile pour :

- classer les outliers,
- visualiser les valeurs extrêmes avant normalisation,
- vérifier les distributions de variables continues,
- explorer les quantiles avant transformation.

3. Fonctions SQL : Texte, Numériques, Dates, Conversion et Traitement des Valeurs Manquantes

Science des Données

[databox,title=Pourquoi ces fonctions sont essentielles en Science des Données ?] Les fonctions SQL sont indispensables pour :

- préparer et nettoyer les données,
- créer des **features** (variables dérivées),
- transformer des variables avant apprentissage,
- formater des dates, nombres et textes,
- traiter les valeurs manquantes,
- normaliser les données entrant dans un pipeline ML.

Elles constituent la base du **Feature Engineering** avant l'analyse statistique ou l'apprentissage automatique.

3.1 Catégories de fonctions SQL

- **Fonctions monolignes** : agissent sur une ligne et retournent une valeur (utiles pour créer des features).
- **Fonctions multilignes** : agrègent plusieurs lignes (voir Chapitre 5).

Les fonctions monolignes peuvent être utilisées dans :

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

3.2 Fonctions sur les chaînes de caractères

Les données textuelles sont très courantes en Science des Données (noms, villes, produits, commentaires, logs, catégories). La normalisation est essentielle pour éviter des doublons ou divergences.

UPPER, LOWER, INITCAP

- **UPPER** : convertit en majuscule,
- **LOWER** : convertit en minuscule,
- **INITCAP** : première lettre en majuscule.

```
SELECT UPPER(last_name), LOWER(first_name),
INITCAP('machine LEARNING')
FROM employees;
```

Science des Données

[databox,title=Utilisation Data Science] Ces fonctions servent à :

- standardiser les labels,
- nettoyer les valeurs catégorielles,
- éviter des doublons avant un encodage (*Label Encoding*, One-Hot...).

LENGTH : longueur d'une chaîne

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

Science des Données

[databox,title=Feature Engineering] La longueur d'un texte peut devenir une feature pour :

- classification (spam / non spam),
- détection de qualité de données,
- NLP (analyse d'avis clients).

SUBSTR : extraction de sous-chaîne

```
SELECT SUBSTR('DATA SCIENCE', 1, 4) FROM dual;  -- DATA
```

Très utile pour :

- extraire un code,
- isoler un préfixe produit,
- découper des champs mal formatés.

INSTR : position d'un caractère

```
SELECT INSTR('MACHINE LEARNING', 'E') FROM dual;
```

CONCAT et opérateur ||

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

3.3 Fonctions numériques

Indispensables pour le prétraitement avant un modèle de ML.

ROUND

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

TRUNC

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

MOD

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

Science des Données

[databox,title=Applications Data Science]

- création d'indicateurs numériques,
- extraction de la partie entière / décimale,
- classification en intervalles (binning),
- création de features : ratios, pourcentages, résidus.

3.4 Fonctions sur les dates

La manipulation des dates est l'un des aspects les plus fréquents en Data Science.

SYSDATE

```
SELECT SYSDATE FROM dual;
```

MONTHS_BETWEEN

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

Très utile pour :

- calculer l'ancienneté,

- construire des variables temporelles,
- effectuer des analyses cohorte.

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;
```

Science des Données

[databox,title=Applications typiques]

- trouver le début/fin de période,
- calculer des fenêtres temporelles,
- générer des index temporels pour séries chronologiques.

3.5 Fonctions de conversion

TO_CHAR

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

Pour convertir un nombre :

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

TO_NUMBER

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

TO_DATE

```
SELECT TO_DATE('2024-03-01', 'YYYY-MM-DD')
FROM dual;
```

Science des Données

[databox,title=Pourquoi c'est critique ?] Les conversions correctes évitent :

- des erreurs d'ETL,
- des incohérences temporelles,
- des corruptions de type (string vs number),
- des crashs de pipelines ML.

3.6 Traitement des valeurs NULL

Astuce

[warnbox,title=NULL = problème classique en Data Science] NULL signifie :

- valeur manquante,
- inconnue,
- non applicable.

Et non pas 0 ou ".

NVL

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

NVL2

```
SELECT NVL2(commission_pct,
'Has bonus', 'No bonus')
FROM employees;
```

COALESCE

Retourne la première valeur non-NULL.

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

Science des Données

[databox,title=Utilisation dans les modèles ML] Avant un apprentissage automatique, il est essentiel de :

- imputer les valeurs manquantes,
- appliquer une stratégie (0, moyenne, médiane, catégorie spéciale),
- garantir un dataset propre.

4. Les Jointures SQL : Un Outil Essentiel pour l'Enrichissement des Données en Data Science

Science des Données

[databox,title=Rôle des jointures en Science des Données] Les jointures permettent de :

- fusionner plusieurs sources de données,
- enrichir les tables (ajout d'attributs explicatifs),
- préparer des jeux de données complets avant le Machine Learning,
- relier logs, utilisateurs, transactions, événements, capteurs,
- construire rapidement des tables analytiques (features tables).

Sans jointures, aucune analyse multi-sources n'est possible.

4.1 Introduction

Une base relationnelle est souvent composée de nombreuses tables : clients, commandes, produits, départements, capteurs, etc.

Pour effectuer une analyse, il faut souvent relier plusieurs tables :

- enrichir les données d'un client avec son historique d'achats,
- relier les événements aux logs d'un système,
- joindre les tables de factures et de paiements,

- ajouter des variables explicatives provenant de sources externes.

C'est le rôle des **jointures SQL**.

4.2 Inner Join (Jointure Interne)

La jointure interne ne conserve que les lignes ayant une correspondance 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;
```

Science des Données

[databox,title=Utilisation en Data Science] L'INNER JOIN est utilisé pour :

- créer une table propre combinant plusieurs sources validées,
- préparer des données d'apprentissage sans valeurs manquantes,
- filtrer uniquement les enregistrements cohérents,
- fusionner plusieurs tables cibles dans un data warehouse.

Ancienne syntaxe Oracle (déconseillée)

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

4.3 Alias de tables

Ils simplifient l'écriture et améliorent la lisibilité des requêtes complexes.

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

Astuce

[tipbox,title=Astuce Data Engineering] Toujours nommer vos tables avec :

- e = employees
- p = products
- c = customers
- tr = transactions

C'est une convention largement utilisée en industrie.

4.4 Jointures sur plusieurs tables

```
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;
```

Science des Données

[databox,title=Cas typiques en Data Science]

- Clients + achats + produits + notes clients,
- Utilisateurs + logs + sessions + device,
- Capteurs + mesures + localisation + météo,
- Données bancaires + transactions + anomalies.

Enrichissement = augmentation de la qualité des modèles.

4.5 Les Jointures Externes (Outer Joins)

Les jointures externes sont essentielles lorsqu'on souhaite analyser **tous** les éléments d'une table, même s'ils ne correspondent à aucun enregistrement dans l'autre.

- **LEFT JOIN** : tous les éléments de gauche + correspondances

- **RIGHT JOIN** : tous les éléments de droite + correspondances
- **FULL JOIN** : toutes les lignes des deux tables

LEFT OUTER JOIN

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

Utilisation :

- garder tous les employés, même sans département,
- identifier les valeurs manquantes,
- effectuer des analyses de complétude.

Attention

[warnbox,title=Cas critique Data Science] Si une jointure LEFT génère beaucoup de NULL, cela indique souvent :

- des données mal liées,
- des valeurs manquantes à corriger,
- des anomalies utiles pour un modèle ML (fraude, churn...).

RIGHT OUTER JOIN

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

Utilisé pour lister :

- tous les départements, même sans employés.

FULL OUTER JOIN

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

Pratique pour :

- détecter les incohérences,
- fusionner des datasets non homogènes,
- analyses exploratoires (EDA).

4.6 Ancienne syntaxe Oracle (+)

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

Attention

[warnbox] Obsolète — à ne jamais utiliser dans des pipelines modernes.

4.7 Self Join : Auto-jointure

Utilisée pour l'analyse hiérarchique ou réseau.

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

Science des Données

[databox,title=Applications en Data Science]

- graphiques réseaux (relations),
- arbres hiérarchiques,
- analyse organisationnelle,
- détection de communautés.

4.8 Jointures avec Conditions Supplémentaires

```
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;
```

Astuce

[tipbox,title=Conseils pratiques] Toujours appliquer les conditions :

- après la jointure (WHERE),
- ou pendant la jointure (ON) selon le besoin analytique.

4.9 NATURAL JOIN et USING

NATURAL JOIN

Relie automatiquement toutes les colonnes portant le même nom.

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

Attention

[warnbox,title=Risque majeur] Une nouvelle colonne avec le même nom dans les 2 tables peut changer totalement le résultat sans prévenir.

USING

Plus sûr que NATURAL JOIN :

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

Science des Données

[databox,title=Recommandation Data Science] Toujours préférer :

- INNER JOIN + ON,
- LEFT JOIN + ON,
- JOIN ... USING,

pour garantir la reproductibilité.

5. Fonctions Multilignes, Agrégation, Group By & HAVING en Analyse de Données

Science des Données

[databox,title=Objectif du chapitre] Ce chapitre introduit les fonctions d'agrégation SQL indispensables en :

- analyse statistique,
- préparation des données,
- Data Profiling & EDA (Exploratory Data Analysis),
- génération de features agrégés pour les modèles ML,
- reporting analytique.

5.1 Introduction

Les fonctions multilignes (appelées **fonctions d'agrégation**) opèrent sur un ensemble de lignes et renvoient une seule valeur. Elles sont essentielles pour :

- décrire statistiquement un dataset,
- créer des indicateurs métier (KPI),
- analyser des comportements utilisateurs,
- produire des variables agrégées (mean, sum, count...) pour le Machine Learning,

- explorer rapidement de grands volumes de données.

Astuce

[tipbox,title=Rôle en Data Science] Les agrégations permettent :

- d'identifier des tendances,
- de repérer des anomalies ou outliers,
- d'enrichir les jeux de données (features engineering),
- de consolider des données brutes pour les modèles supervisés.

5.2 Fonctions d'Agrégation : Outils Statistiques de Base

Les cinq principales fonctions multilignes sont :

- **SUM(expr)** : somme (indicateur global),
- **AVG(expr)** : moyenne,
- **MIN(expr)** : minimum,
- **MAX(expr)** : maximum,
- **COUNT(expr)** : nombre de valeurs non NULL,
- **COUNT(*)** : nombre total de lignes.

Exemples essentiels

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

Science des Données

[databox,title=Applications scientifiques]

- SUM : consommation totale, trafic réseau, ventes, logs.
- AVG : température moyenne, churn moyen, durée de session.
- MIN/MAX : détecter extrêmes, anomalies, bornes.
- COUNT : volumétrie des données, disponibilité, sparsité.

5.3 Différence : COUNT(*) vs COUNT(colonne)

- **COUNT(*)** → compte toutes les lignes
- **COUNT(colonne)** → ignore les valeurs NULL

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

Attention

[warnbox,title=Cas important en Data Science] Les NULL sont omniprésents dans les données réelles. COUNT(colonne) est donc très utile pour :

- évaluer la qualité d'un dataset,
- mesurer le taux de complétude,
- détecter des colonnes peu exploitables pour le ML.

5.4 La clause GROUP BY : Outil majeur d'agrégation

GROUP BY regroupe les lignes selon une ou plusieurs colonnes.

Exemple simple : salaire moyen par département

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

Règle fondamentale

Toute colonne dans SELECT doit être :

- soit dans GROUP BY,
- soit dans une fonction d'agrégation.

Exemple multi-colonnes

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

Science des Données

[databox,title=Utilité pour la Science des Données] GROUP BY est utilisé pour :

- produire des statistiques descriptives,
- agréger des séries temporelles,
- calculer des features pour un modèle ML :
  - mean per user,
  - sum per session,
  - count per transaction type.
- créer des cubes analytiques multidimensionnels.

5.5 Ordre logique d'exécution SQL

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

6. ORDER BY

Astuce

[tipbox,title=Bonne pratique] Toujours penser à cet ordre pour comprendre :

- pourquoi HAVING agit après le regroupement,
- pourquoi WHERE ne peut pas filtrer une agrégation,
- pourquoi les alias parfois ne fonctionnent pas dans GROUP BY.

5.6 HAVING : Filtrer les groupes agrégés

HAVING filtre les groupes générés par GROUP BY.

Exemple : départements où le salaire moyen dépasse 5000

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

WHERE vs HAVING

- WHERE → filtre avant regroupement,
- HAVING → filtre après regroupement.

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

Science des Données

[databox,title=Cas réel en Data Science] HAVING permet de :

- trouver des utilisateurs actifs (COUNT(events) $>$ 10),
- repérer des catégories avec forte variance,
- identifier des clusters préliminaires,
- filtrer des valeurs extrêmes agrégées.

5.7 Effets des NULL dans les agrégations

Les valeurs NULL sont ignorées pour :

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

Exception : COUNT(*) compte tout.

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

Attention

[warnbox,title=Impact en Machine Learning] Les NULL faussent :

- les moyennes,
- les variances,
- les outliers,
- les distributions.

D'où l'usage de NVL/COALESCE avant agrégation.

5.8 Agrégation temporelle (Time Series Analysis)

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;
```

Applications :

- analyses saisonnières,
- séries temporelles pour la prédiction,
- forecast (prévisions),
- analyse de cohortes (cohort analysis).

5.9 Agrégations avancées pour la Data Science

Variance, écart-type, covariances

Oracle propose :

- **VAR_SAMP(col)**
- **STDDEV_SAMP(col)**
- **COVAR_SAMP(x, y)**

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

Science des Données

[databox,title=Usage dans la modélisation] Mesurer la variance permet :

- de détecter les colonnes instables,
- d'identifier des variables non discriminantes,
- de préparer le scaling des données,
- de repérer des patterns non linéaires.

Fenêtre glissante (WINDOW FUNCTIONS) — aperçu

Bien qu'elles soient dans le chapitre "Fonctions analytiques", ces fonctions sont essentielles en DS :

- ROW_NUMBER
- RANK
- LAG / LEAD
- AVG(...) OVER(...)

```
SELECT employee_id,
salary,
AVG(salary) OVER (PARTITION BY department_id)
FROM employees;
```

Astuce

[tipbox,title=Pourquoi c'est crucial ?] Les fonctions analytiques permettent :

- d'éviter des sous-requêtes coûteuses,
- d'obtenir des statistiques par groupe sans GROUP BY,
- de calculer des tendances,
- d'analyser des séries temporelles.

6. Sous-interrogations (Subqueries) : Analyse avancée pour la Science des Données

Science des Données

[databox,title=Objectif du chapitre] Les sous-interrogations — ou **subqueries** — permettent :

- d'extraire des informations complexes,
- de créer des filtres dynamiques,
- de travailler avec des agrégations intermédiaires,
- d'obtenir des données dérivées pour le Machine Learning,
- de remplacer certaines jointures avec logique analytique,
- d'optimiser des requêtes pour l'Exploratory Data Analysis (EDA).

6.1 Introduction

Une sous-interrogation est une requête imbriquée à l'intérieur d'une autre requête. Elle est exécutée en premier, et son résultat sert d'entrée à la requête principale.

Applications en Data Science :

- filtrer les utilisateurs au-dessus d'un seuil dynamique,
- comparer des valeurs individuelles au comportement global,
- extraire des cohortes de clients,

- identifier des anomalies (salary > moyenne du département),
- générer des features complexes (ratios, écarts, classements).

Positions possibles

Les sous-requêtes peuvent apparaître dans :

- SELECT,
- FROM (vue en ligne),
- WHERE,
- HAVING.

Attention

[warnbox,title=Important] Les sous-interrogations sont fondamentales en Data Science SQL car elles permettent :

- d'éviter des jointures lourdes,
- de structurer une requête complexe en étapes,
- de créer des mini-pipelines de transformation.

6.2 Sous-interrogations scalaires

Une sous-interrogation scalaire renvoie une seule valeur.

Elles sont très utilisées pour :

- comparer une ligne au global (ex : plus que la moyenne),
- extraire une information agrégée,
- mettre à jour un dataset selon une valeur calculée.

Exemple classique

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

Science des Données

[databox,title=Application Data Science] Trouver les valeurs extrêmes :

- utilisateurs très actifs,
- clients premium,
- anomalies de logs,
- employés “Above Benchmark”.

6.3 Sous-interrogations multivaluées

Ces sous-requêtes retournent un ensemble de valeurs.

IN : appartient à l’ensemble

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

Astuce

[tipbox,title=Application] Créer des sous-ensembles cohérents :

- employés d’une ville,
- clients d’un segment,
- machines dans un même datacenter.

ANY : vrai pour au moins une valeur

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

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

- équivalent à : salary > MIN(salaire du département 50)

ALL : vrai pour toutes les valeurs

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

- équivalent à : salary > MAX(salaire du département 50)

Science des Données

[databox,title=Usage en Data Science] **ANY/ALL** sont utiles pour :

- sélectionner les top performers,
- identifier des anomalies,
- définir des seuils intelligents,
- créer des règles de scoring.

6.4 Sous-interrogations corrélées

La sous-requête dépend de la ligne courante de la requête externe. Elle s'exécute une fois par ligne.

```
SELECT e.last_name, e.salary
FROM employees e
```

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

Attention

[warnbox,title=Modélisation avancée] Utile pour créer des features :

- position de l'utilisateur dans son groupe,
- écart par rapport à la moyenne,
- comportement relatif dans une catégorie.

6.5 Sous-interrogations dans FROM (views en ligne)

Elles permettent de structurer des requêtes complexes.

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

Science des Données

[databox,title=Pipeline analytique] C'est l'équivalent d'avoir une table temporaire pendant :

- un calcul de KPIs,
- un EDA,
- une préparation de données pour ML.

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
)
);
```

Astuce

[tipbox,title=Cas d'usage réel] Identifier :

- des départements supérieurs à la moyenne,
- des catégories dominantes,
- des clusters naturels,
- des anomalies structurelles.

6.7 Sous-requêtes dans SELECT

Une sous-requête dans SELECT doit renvoyer une seule valeur.

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

Applications :

- enrichissement de dataset,

- lookup d'attributs,
- construction d'un dataframe final en SQL.

6.8 Règles importantes

- toute sous-requête est entre parenthèses,
- une sous-requête scalaire doit renvoyer une seule ligne,
- ALL, ANY, IN → sous-requêtes multivaluées,
- une sous-requête corrélée ne peut pas s'exécuter seule,
- optimiser via indexes lorsqu'il y a fortes dépendances.

Attention

[warnbox,title=Performance en Big Data] Les sous-requêtes peuvent coûter cher si :

- la table est très volumineuse,
- les index manquent,
- la sous-requête est corrélée.

Toujours vérifier l'impact avant intégration dans un pipeline ETL/ELT.

6.9 Résumé

- **Scalaires** : une seule valeur → comparaison directe.
- **Multivaluées** : liste de valeurs → IN/ANY/ALL.
- **Corrélées** : dépend de chaque ligne → features avancées.
- **FROM** : pipeline analytique, vue en ligne.
- **HAVING** : filtrage analytique post-agrégation.
- **SELECT** : enrichissement de dataset.

7. Langage de Manipulation des Données (LMD) : INSERT, UPDATE, DELETE pour la Science des Données

Science des Données

[databox,title=Objectifs du chapitre] Ce chapitre présente les opérations de manipulation des données, essentielles en :

- ingestion de données (Data Ingestion),
- nettoyage et préparation (Data Cleaning),
- transformation (ETL / ELT),
- mise à jour de tables de travail,
- gestion des datasets utilisés pour le Machine Learning,
- correction d'anomalies ou de valeurs manquantes.

7.1 Introduction

Le Langage de Manipulation des Données (LMD) — ou DML (Data Manipulation Language) — regroupe les commandes SQL permettant de modifier le contenu des tables :

- **INSERT** : ajouter des données,
- **UPDATE** : modifier des données existantes,
- **DELETE** : supprimer des données.

Attention

[warnbox,title=Important en Data Science] Les instructions LMD jouent un rôle majeur dans :

- la préparation des datasets,
- la gestion des valeurs NULL,
- la correction des valeurs aberrantes,
- la construction de tables de features pour le ML,
- les opérations de DataOps et synchronisation des données.

Une modification n'est **définitive** qu'après un **COMMIT**. Avant cela :

- seul l'utilisateur courant voit les changements,
- un **ROLLBACK** peut annuler les modifications.

7.2 INSERT : Ingestion de nouvelles données

INSERT permet d'ajouter des lignes dans une table. C'est une opération courante lors de :

- l'ingestion de fichiers CSV/JSON,
- le chargement de données externes (API, logs),
- l'intégration de nouvelles observations,
- la préparation de tables d'entraînement ML.

Syntaxe complète

```
INSERT INTO table_name
VALUES (val1, val2, ...);
```

Syntaxe recommandée (insertion partielle)

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

Astuce

[tipbox,title=Pourquoi c'est essentiel en Data Engineering ?] **INSERT partiel** :

- évite les erreurs si la structure évolue,
- facilite le mapping avec des fichiers d'entrée,
- rend les pipelines plus robustes.

Exemple (ingestion d'une nouvelle observation)

```
INSERT INTO customers (customer_id, age, city)
VALUES (1001, 29, 'Casablanca');
```

INSERT avec sous-interrogation (ETL / ELT)

```
INSERT INTO high_salary_employees (employee_id, last_name, salary)
SELECT employee_id, last_name, salary
FROM employees
WHERE salary > 7000;
```

Science des Données

[databox,title=Cas d'usage industriel] Très utilisé pour :

- créer des tables intermédiaires,
- stocker des cohortes d'utilisateurs,
- sauvegarder des segments pour analyse ML,
- préparer des datasets de feature engineering.

7.3 UPDATE : Modifier des données

UPDATE modifie les lignes existantes. Utile pour :

- corriger les erreurs,
- mettre à jour des valeurs manquantes,

- appliquer des règles métier (scoring, étiquettes),
- synchroniser des datasets.

Syntaxe

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

Attention

[warnbox,title=Attention] Sans WHERE : **toutes les lignes seront modifiées.**

Exemple (nettoyage de données)

```
UPDATE customers
SET city = 'Unknown'
WHERE city IS NULL;
```

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
);
```

Science des Données

[databox,title=Usage en ML] Mettre à jour les données source permet de :

- appliquer un label,
- corriger une feature,
- normaliser les valeurs,
- assigner une cohorte.

7.4 DELETE : Suppression de données

DELETE supprime des lignes selon une condition.

Utile pour :

- retirer des anomalies,
- supprimer des doublons (dédupliqués par clés),
- nettoyer un dataset,
- gérer des pipelines incrémentaux.

Syntaxe

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

Attention

[warnbox,title=Sans WHERE] Toutes les lignes seront supprimées.

Exemple (suppression d'anomalies)

```
DELETE FROM transactions
WHERE amount < 0;
```

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 LMD nécessite **COMMIT** pour être permanente.
- **ROLLBACK** annule les modifications non validées.
- Un LMD erroné ne modifie aucune donnée.
- Les sous-requêtes permettent une manipulation conditionnelle puissante.

Mesurer l'impact d'une modification

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

Astuce

[tipbox,title=Bonnes pratiques Data Science] Avant de modifier un dataset :

- dupliquer la table (CTAS),
- vérifier la volumétrie,
- tester dans un environnement de staging,
- documenter les transformations appliquées.

7.6 TRUNCATE : vider rapidement une table

```
TRUNCATE TABLE table_name;
```

Différences avec DELETE

- TRUNCATE est beaucoup plus rapide,
- TRUNCATE fait un COMMIT implicite,
- TRUNCATE ne permet pas de ROLLBACK,
- DELETE peut utiliser WHERE (TRUNCATE non).

Science des Données

[databox,title=Usage en Data Engineering] TRUNCATE est utilisé dans :

- les jobs ETL quotidiens,
- le rechargement complet (full load),
- les environnements de test ML,
- les pipelines de feature engineering réinitialisés.

8. Transactions : COMMIT, ROLLBACK et SAVEPOINT pour la Data Science & Data Engineering

Science des Données

[databox,title=Objectifs du chapitre] Ce chapitre est essentiel pour :

- assurer la fiabilité des pipelines de données (ETL/ELT),
- garantir la cohérence des datasets,
- gérer les erreurs lors d'un traitement,
- comprendre le fonctionnement multi-utilisateur,
- appliquer un versioning logique sur les données,
- éviter les corruptions de données dans les systèmes analytiques.

8.1 Introduction

Une **transaction** est une séquence d'instructions SQL considérée comme une unité logique de travail.

Elle garantit :

- **Atomicité** : tout ou rien,
- **Cohérence** : les règles sont respectées,
- **Isolation** : chaque utilisateur travaille indépendamment,

- **Durabilité** : les changements validés sont persistants.

Ces propriétés sont connues sous l'acronyme **ACID**. Elles sont fondamentales dans les systèmes de bases de données utilisés en Data Engineering.

Début et fin d'une transaction

Une transaction commence :

- après un COMMIT ou ROLLBACK,
- ou automatiquement après la première instruction LMD (INSERT, UPDATE, DELETE).

Elle se termine lorsqu'un utilisateur exécute :

- **COMMIT** : valider les changements,
- **ROLLBACK** : annuler les changements,
- ou lorsqu'une commande LDD est lancée (CREATE, DROP, etc.).

8.2 COMMIT : Valider définitivement les changements

Astuce

[tipbox,title=Quand l'utiliser ?] **COMMIT** doit être utilisé lorsque l'on est certain que les modifications sont correctes. Très utile dans les pipelines ETL en fin de traitement.

COMMIT confirme toutes les modifications effectuées dans la transaction courante.
Effets d'un COMMIT :

- les changements deviennent permanents,
- les autres utilisateurs voient les données mises à jour,
- les verrous sur les lignes sont libérés.

Exemple

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

COMMIT;
```

Science des Données

[databox,title=Usage professionnel] Dans un pipeline DataOps :

- Extraction → Transformation → Chargement → **COMMIT**
- Empêche les corruptions de données,
- Finalise un lot de données après audit.

8.3 ROLLBACK : Annuler les changements

ROLLBACK restaure l'état de la base avant la transaction en cours.

- 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;  -- annule la suppression
```

Attention

[warnbox,title=Rôle dans la qualité des données] ROLLBACK est indispensable pour :

- corriger des erreurs d'ingestion,
- annuler un traitement partiellement réussi,
- garantir que les datasets restent cohérents,
- sécuriser un pipeline en cas d'échec d'une étape.

8.4 SAVEPOINT : Point de restauration partiel

SAVEPOINT permet de définir un point intermédiaire dans la transaction.

```
SAVEPOINT pointA;
```

On peut ensuite revenir à ce point :

```
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;

COMMIT;
```

Science des Données

[databox,title=Application Data Engineering] Les SAVEPOINT sont utiles dans :

- les transformations multi-étapes,
- les pipelines nécessitant du nettoyage progressif,
- les migrations de données délicates,
- les opérations sensibles (matching, enrichissement...).

8.5 Transactions implicites : COMMIT automatique

Certaines instructions entraînent un COMMIT automatique :

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

Attention

[warnbox,title=Attention, très important] Une commande LDD :

- valide implicitement toutes les transactions ouvertes,
- peut rendre impossible un retour en arrière.

Exemple dangereux

```
DELETE FROM employees WHERE salary < 2000;
DROP TABLE employees;   -- tout est validé automatiquement
```

8.6 Environnement multi-utilisateur

Dans un système multi-utilisateur (entreprise, cluster analytique...), Oracle garantit la cohérence des données.

Verrouillage des lignes

Lors d'une instruction DML :

- les lignes modifiées sont verrouillées,
- aucun autre utilisateur ne peut les modifier,
- les autres utilisateurs peuvent lire (lecture cohérente).

Isolation des transactions

Chaque utilisateur voit :

- ses propres changements non validés,
- les données validées des autres utilisateurs,
- mais pas les modifications non validées des autres.

Exemple explicatif

- A modifie un salaire → non visible par B,
- A exécute COMMIT → modification visible par B.

Science des Données

[databox,title=Cas réel en data science collaborative] Utile lorsqu'une équipe :

- prépare un dataset commun,
- teste différentes transformations,
- manipule simultanément une base de données partagée.

8.7 Bonnes pratiques professionnelles

- Toujours tester les insertions/mises à jour sur un échantillon.
- Utiliser SAVEPOINT lors de transformations risquées.
- Documenter tout changement sur les données.

- Ne jamais faire COMMIT automatiquement dans un script non maîtrisé.
- Toujours mesurer l'impact d'une transformation :

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

Astuce

[tipbox,title=En résumé] Les transactions sont le garant :

- de la qualité,
- de la cohérence,
- de la fiabilité

des données utilisées en Data Science.

9. Langage de Définition des Données (LDD) : CREATE, ALTER, DROP, TRUNCATE en Architecture de Données

Science des Données

[databox,title=Objectifs du chapitre] Ce chapitre explique comment créer, modifier et supprimer les objets d'une base de données, dans un contexte :

- de Data Engineering,
- de conception de schémas relationnels,
- d'architecture Data Warehouse / Data Lakehouse,
- de préparation des datasets pour Machine Learning,
- de gouvernance et qualité des données,
- de pipelines ETL/ELT modernes.

9.1 Introduction au LDD

Le Langage de Définition des Données (LDD — ou DDL : Data Definition Language) permet de manipuler la structure des objets d'une base Oracle :

- **CREATE** : créer une table, vue, index ou autre objet,
- **ALTER** : modifier un objet existant,
- **DROP** : supprimer un objet définitivement,

- **TRUNCATE** : vider une table de manière rapide.

Attention

[warnbox,title=IMPORTANT] Les instructions LDD provoquent un **COMMIT implicite** :

- toutes les transactions ouvertes sont automatiquement validées,
- tout retour arrière devient impossible.

9.2 Rôle du LDD dans un environnement Data Science

Les commandes LDD sont nécessaires pour :

- définir des tables de staging pour ETL/ELT,
- modéliser des schémas en étoile (data warehouse),
- créer des tables d'agrégation (fact tables),
- préparer des tables de features pour le Machine Learning,
- importer des datasets externes,
- gérer l'évolution des schémas (Schema Evolution).

9.3 CREATE TABLE : Création de tables structurées

Pour créer une table, il faut définir :

- les colonnes,
- les types de données,
- les contraintes,
- la clé primaire,
- les relations avec d'autres tables.

Syntaxe générale

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

Exemple (table pour dataset client)

```
CREATE TABLE customers (
customer_id   NUMBER(10) PRIMARY KEY,
age           NUMBER(3),
gender        CHAR(1),
city          VARCHAR2(50),
signup_date   DATE,
income        NUMBER(10,2)
);
```

Science des Données

[databox,title=Bonne pratique Data Science] Toujours prévoir :

- un identifiant unique stable (clé primaire),
- des colonnes propres (types adaptés),
- des contraintes pour éviter les valeurs erronées,
- des colonnes DATE/TIMESTAMP pour les analyses temporelles.

9.4 Types de données courants en Data Science

Types principaux

- **VARCHAR2(n)** : chaînes de caractères (text data),
- **CHAR(n)** : chaînes fixes,
- **NUMBER(p, s)** : données numériques (features),

- **DATE** : dates,
- **TIMESTAMP** : données temporelles précises,
- **CLOB** : texte volumineux (logs, JSON),
- **BLOB** : données binaires (images, audio).

Cas d'utilisation en Data Science

- TEXT / CLOB : NLP, prétraitement de texte,
- BLOB : Computer Vision, stockage binaire,
- NUMBER : features normales,
- DATE / TIMESTAMP : séries temporelles, cohortes.

9.5 ALTER TABLE : Faire évoluer un schéma

ALTER TABLE est essentiel pour :

- ajouter des colonnes lors du Feature Engineering,
- modifier les types selon l'évolution du dataset,
- introduire ou retirer des contraintes,
- renommer des colonnes pour plus de clarté,
- gérer les évolutions d'un schéma sur plusieurs versions.

Ajouter une colonne

```
ALTER TABLE customers
ADD (email VARCHAR2(80));
```

Modifier un type

```
ALTER TABLE customers
MODIFY (income NUMBER(12,2));
```

Supprimer une colonne

```
ALTER TABLE customers
DROP COLUMN gender;
```

Renommer une colonne

```
ALTER TABLE customers
RENAME COLUMN income TO annual_income;
```

Attention

[warnbox,title=Migration de schéma] ALTER TABLE est crucial lors :

- d'une migration d'architecture,
- de l'évolution d'un modèle ML (ajout de features),
- d'une correction de design de base.

9.6 Renommer une table

```
RENAME customers TO clients_data;
```

Science des Données

[databox,title=Cas réel] Lors de refactoring d'un entrepôt :

- renommer les tables pour harmoniser les conventions,
- clarifier des noms ambigus,
- basculer vers un nouveau modèle analytique.

9.7 DROP TABLE : Suppression définitive

```
DROP TABLE customers;
```

Attention

[warnbox,title=Attention] DROP TABLE = suppression permanente et irréversible !

DROP TABLE ... CASCADE CONSTRAINTS

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

Cette option supprime la table même si d'autres objets en dépendent.

9.8 TRUNCATE : Vider rapidement une table

```
TRUNCATE TABLE customers;
```

TRUNCATE vs DELETE

- TRUNCATE est beaucoup plus rapide,
- TRUNCATE fait un commit automatique,
- TRUNCATE ne permet pas de ROLLBACK,
- DELETE peut utiliser un WHERE (TRUNCATE non).

Science des Données

[databox,title=Utilisation en pipelines ETL] TRUNCATE est utilisé dans :

- les rechargements complets (Full Reload),
- la réinitialisation de tables de staging,
- les environnements temporaires pour ML,
- les jobs nocturnes de nettoyage.

9.9 CTAS : Create Table As Select

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

Syntaxe

```
CREATE TABLE users_2024 AS
SELECT user_id, city, age
FROM users
WHERE signup_date >= '01-JAN-2024';
```

Astuce

[tipbox,title=Usage en Data Science] CTAS est utilisé pour :

- créer des jeux de données d'entraînement ML,
- générer des tables intermédiaires pour EDA,
- faire du time-slicing de données,
- accélérer l'exploration en copiant des sous-ensembles.

Particularités

- les contraintes ne sont pas copiées,
- seuls les noms de colonnes et les données sont dupliqués,
- extrêmement performant pour créer de grands datasets.

9.10 Bonnes pratiques professionnelles

- Utiliser CTAS pour préparer rapidement un dataset.
- Documenter chaque évolution de schéma.
- Préférer ALTER TABLE plutôt que DROP/CREATE.
- Utiliser des types adaptés aux analyses (DATE, NUMBER, CLOB).
- Éviter DROP TABLE en production.
- Tester les modifications sur une copie avant déploiement.

Astuce

[tipbox,title=En résumé] LDD constitue la base :

- de la modélisation des données,
- de la préparation des architectures analytiques,
- de la gestion de la qualité des données,
- des pipelines modernes pour la Data Science.

10. Les Contraintes en Base de Données (PRIMARY, FOREIGN, CHECK, UNIQU NOT NULL)

Science des Données

[databox,title=Objectifs du chapitre] Dans ce chapitre, vous allez apprendre à :

- garantir l'intégrité et la qualité des données ;
- définir les clés primaires et étrangères ;
- restreindre la saisie grâce à CHECK et NOT NULL ;
- modéliser correctement les relations dans des schémas orientés Data Science ;
- comprendre l'importance des contraintes dans les pipelines ETL/ELT.

10.1 Introduction

Les contraintes jouent un rôle essentiel dans toute architecture de données :

- elles vérifient la validité des valeurs,
- empêchent les incohérences dans les datasets,
- assurent la cohérence des pipelines d'ingestion,
- garantissent l'intégrité référentielle dans les Data Warehouses,
- facilitent la qualité des données pour le Machine Learning.

Les principales contraintes Oracle sont :

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

Attention

[warnbox,title=Rôle dans la Science des Données] Les contraintes permettent de prévenir les problèmes fréquents :

- doublons,
- valeurs manquantes injustifiées,
- incohérences dans les référentiels,
- valeurs aberrantes invalides,
- pertes d'intégrité dans des tables de features.

10.2 Contrainte NOT NULL

Une colonne NOT NULL doit obligatoirement contenir une valeur.

Syntaxe

```
age NUMBER NOT NULL
```

Ajout via ALTER TABLE

```
ALTER TABLE customers
MODIFY (age NOT NULL);
```

Astuce

[tipbox,title=Utilité en Data Science] On utilise NOT NULL pour :

- éviter des valeurs manquantes dans les features critiques ;
- imposer une présence obligatoire (identifiants, dates) ;
- garantir que les datasets utilisés pour ML sont propres.

10.3 Contrainte UNIQUE

Cette contrainte garantit qu'aucune valeur dupliquée n'existe dans la colonne.

Syntaxe

```
email VARCHAR2(80) UNIQUE
```

Contrainte nommée

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

Science des Données

[databox,title=Applications pratiques] La contrainte UNIQUE s'applique dans :

- les datasets clients (emails, numéros de téléphone),
- les tables d'utilisateurs (login, identifiant externe),
- la déduplication de données en Data Engineering.

10.4 PRIMARY KEY (Clé primaire)

Une clé primaire identifie de manière unique une ligne.

Propriétés

- UNIQUE,

- NOT NULL,
- indexée automatiquement,
- sert de référence pour les FOREIGN KEYS.

Exemple

```
CONSTRAINT cust_pk PRIMARY KEY (customer_id)
```

Astuce

[tipbox,title=Bonne pratique Data Engineering] Toujours utiliser une clé primaire :

- stable,
- non réutilisée,
- non modifiable,
- idéalement un entier numérique.

10.5 FOREIGN KEY : Clé étrangère

La FOREIGN KEY relie deux tables et garantit que les valeurs existent dans la table parent.

Syntaxe

```
CONSTRAINT cust_city_fk
FOREIGN KEY (city_id)
REFERENCES cities(city_id)
```

Science des Données

[databox,title=Importance dans la Data Science] La clé étrangère :

- assure la cohérence entre référentiels (produits, villes, clients),
- évite les valeurs orphelines,
- facilite les jointures dans les analyses et modèles ML,
- sécurise les pipelines ETL.

Options ON DELETE

```
ON DELETE CASCADE     -- supprime aussi les enregistrements enfants
ON DELETE SET NULL    -- remplace la FK par NULL
```

Attention

[warnbox,title=Cas d'usage réel] **ON DELETE CASCADE** :

- utile pour nettoyage automatique ;
- dangereux si mal utilisé.

10.6 Contrainte CHECK

CHECK impose une condition logique.

Exemples

```
age NUMBER CONSTRAINT age_ck CHECK (age >= 0)
```

```
CHECK (gender IN ('M','F'))
```

Astuce

[tipbox,title=Importance dans la qualité des données] CHECK permet :

- d'éviter des valeurs absurdes (salaire négatif),
- limiter les catégories à des valeurs autorisées,
- garantir la propreté dans les datasets ML.

10.7 Déclaration inline / out-of-line

Inline (dans la définition de colonne)

```
customer_id NUMBER PRIMARY KEY
```

Out-of-line (séparée)

```
CONSTRAINT cust_pk PRIMARY KEY (customer_id)
```

Astuce

[tipbox,title=Bonne pratique] Toujours nommer vos contraintes :

- facilite debugging,
- rend les schémas lisibles,
- simplifie ALTER TABLE.

10.8 Ajouter une contrainte après création

```
ALTER TABLE customers
ADD CONSTRAINT income_ck CHECK (income >= 0);
```

10.9 Désactiver temporairement une contrainte

```
ALTER TABLE customers DISABLE CONSTRAINT cust_pk;
ALTER TABLE customers ENABLE  CONSTRAINT cust_pk;
```

Attention

[warnbox,title=Attention en production] Désactiver des contraintes peut entraîner :

- des erreurs dans les pipelines,
- des données incohérentes pour le Machine Learning,
- des jointures invalides.

10.10 Suppression d'une contrainte

```
ALTER TABLE customers
DROP CONSTRAINT income_ck;
```

10.11 Contraintes différables : DEFERRABLE

Certaines contraintes peuvent être vérifiées uniquement à la fin de la transaction.

```
ALTER TABLE customers
ADD CONSTRAINT salary_ck
CHECK (annual_income > 0)
DEFERRABLE INITIALLY DEFERRED;
```

Science des Données

[databox,title=Utilisation en Data Engineering] Pratique pour :

- charger des données en plusieurs étapes,
- réaliser des migrations de schémas,
- effectuer des traitements complexes avec dépendances.

Astuce

[tipbox,title=Résumé général] Les contraintes assurent :

- la cohérence structurelle,
- la qualité des datasets,
- la fiabilité des pipelines,
- l'intégrité des modèles ML.

Elles sont indispensables dans tout système moderne orienté Science des Données.