Langage SQL (PostgreSQL)

Samy AMRI

JaimeExcel.com

Mentions légales et droits d'auteur

© 2026 – Samy AMRI, fondateur de JaimeExcel.

Le présent support de formation est une œuvre protégée par le **Code de la propriété intellectuelle.**

Son acquisition confère à l'acquéreur un **droit d'utilisation personnel, non exclusif, non cessible et non transférable**, strictement limité à un usage privé ou interne, selon la qualité de l'acquéreur (particulier ou professionnel).

Toute reproduction, représentation, diffusion, mise à disposition, partage, revente, location, adaptation, traduction ou exploitation, totale ou partielle, par quelque procédé que ce soit, **sans autorisation écrite préalable de l'auteur**, est strictement interdite.

Sont notamment interdites :

- **La diffusion à des tiers, gratuite ou payante ;**
- **L'utilisation à des fins de formation, d'enseignement, de démonstration ou de transmission à des tiers ;**
- **L'intégration dans une plateforme numérique, LMS, intranet, extranet ou tout système de partage ;**
- **Toute exploitation commerciale directe ou indirecte.**

Toute utilisation non autorisée constitue un acte de contrefaçon, engageant la responsabilité civile et pénale de son auteur, conformément aux articles **L.335-2 et suivants du Code de la propriété intellectuelle.**

Les droits non expressément concédés demeurent la propriété exclusive de l'auteur.

Table des Matières

1 Introduction aux bases de données SQL 7

1.1 Ressources . . . 7
1.2 La base de données . . . 7
1.3 le Serveur . . . 7
1.4 Le Système de Gestion de Base de Données (SGBD) . . . 8
1.5 Le Modèle Relationnel . . . 8
1.6 Les contraintes d'intégrité . . . 9
1.6.1 Intégrité de domaine . . . 9
1.6.2 Intégrité d'entité . . . 9
1.6.3 Intégrité référentielle . . . 10
1.6.4 Intégrité transactionnelle . . . 10
1.7 Les clés : garantie de la cohérence et de l'unicité . . . 11
1.8 La normalisation des bases de données . . . 12
1.8.1 Pourquoi normaliser ? . . . 12
1.8.2 Première forme Normale (1NF) . . . 13
1.8.3 Deuxième forme normale (2NF) . . . 13
1.8.4 Troisième forme normale (3NF) . . . 15
1.9 Le langage SQL . . . 16
1.10 Installation de PostgreSQL . . . 16
1.11 Base de données de travail . . . 16

2 Extraire des données 18

2.1 SELECT … FROM . . . 18
2.2 WHERE . . . 19
2.2.1 Opérateurs de comparaisons . . . 19
2.2.2 AND & OR . . . 20
2.2.3 IN . . . 21
2.2.4 BETWEEN . . . 22
2.2.5 LIKE . . . 23
2.2.6 IS NULL / IS NOT NULL . . . 24
2.3 DISTINCT . . . 25
2.4 Exercices . . . 26
2.5 Solutions . . . 34

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

3 Ordonnancement et Agrégations 35

3.1 Fonctions d'agrégation . . . 36
3.1.1 SUM() . . . 36
3.1.2 COUNT() . . . 36
3.1.3 AVG() . . . 37
3.1.4 MAX() . . . 37
3.1.5 MIN() . . . 37
3.2 GROUP BY . . . 38
3.3 HAVING . . . 38
3.4 AS . . . 39
3.5 ORDER BY . . . 39
3.6 LIMIT . . . 41
3.7 OFFSET / FETCH . . . 42
3.8 PARTITION BY() . . . 43
3.9 ROW_NUMBER() . . . 45
3.10 RANK() . . . 47
3.11 Exercices . . . 50
3.12 Solutions . . . 57

4 Présenter les données 59

4.1 Opérations sur les chaines de caractère (varchar/nvarchar) . . . 59
4.1.1 LTRIM() . . . 59
4.1.2 RTRIM() . . . 59
4.1.3 TRIM() . . . 59
4.1.4 LENGTH() . . . 59
4.1.5 SUBSTRING() . . . 59
4.1.6 CONCAT() . . . 60
4.1.7 REPLACE() . . . 60
4.1.8 UPPER()/LOWER() . . . 60
4.2 Opérations sur les dates . . . 60
4.2.1 CURRENT_DATE . . . 60
4.2.2 EXTRACT(YEAR ou MONTH ou DAY FROM …) . . . 61
4.2.3 DATEADD() . . . 61
4.2.4 AGE() . . . 61
4.2.5 FORMAT() . . . 61
4.3 Conversion de données . . . 62
4.3.1 CAST() . . . 62

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

4.3.2 TRY_CAST()/TRY_CONVERT() . . . 62
4.3.3 Exemples de conversion de données . . . 62
4.4 CASE . . . 62
4.5 Exercices . . . 66
4.6 Solutions . . . 79

5 Interroger plusieurs tables 81
5.1 Jointures SQL . . . 82
5.2 Types de jointures . . . 82
5.2.1 INNER JOIN . . . 83
5.2.2 LEFT JOIN . . . 85
5.2.3 RIGHT JOIN . . . 87
5.2.4 FULL OUTER JOIN . . . 89
5.3 Combiner sans jointure . . . 90
5.3.1 UNION . . . 90
5.3.2 UNION ALL . . . 92
5.3.3 INTERSECT . . . 96
5.3.4 EXCEPT . . . 97
5.4 Exercices . . . 99
5.5 Solutions . . . 107

6 Sous-requêtes et vues 109
6.1 Sous-requêtes . . . 110
6.1.1 Sous-requêtes dans WHERE . . . 110
6.1.2 Sous-requêtes dans FROM (table dérivée) . . . 111
6.1.3 Sous-requêtes dans SELECT (calculs scalaires) . . . 111
6.2 Les cas d'utilisation . . . 112
6.2.1 Utiliser une fonction fenêtre dans un WHERE / HAVING . . . 112
6.2.2 Faire un TOP N par groupe (TOP ne partitionne pas) . . . 112
6.2.3 Filtrer sur un cumul, un rang ou un max partitionné . . . 112
6.2.4 Faire une comparaison ligne à ligne . . . 113
6.2.5 Sous-requêtes imbriquées vs corrélées . . . 113
6.3 Opérateurs utiles . . . 113
6.3.1 ANY . . . 113
6.3.2 ALL . . . 114
6.4 Vues . . . 114
6.4.1 Intérêt des vues . . . 114

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6.4.2 Syntaxe et exemples . . . 114
6.5 CTE . . . 115
6.5.1 CTE comme alternative à une sous-requête . . . 116
6.5.2 CTE avec plusieurs niveaux (chaînage) . . . 116
6.5.3 CTE et fonctions analytiques (PARTITION BY) . . . 117
6.5.4 CTE et TOP / OFFSET-FETCH . . . 117
6.5.5 Quand utiliser une CTE ? . . . 118
6.6 Exercices . . . 119
6.7 Solutions . . . 124

7 DDL : Création et Structure (CREATE TABLE, ALTER, DROP) 127
7.1 Manipulation de base de données . . . 128
7.1.1 CREATE DATABASE . . . 128
7.1.2 ALTER DATABASE . . . 128
7.1.3 DROP DATABASE . . . 129
7.2 Manipulation de tables . . . 129
7.2.1 CREATE TABLE . . . 129
7.2.2 Liste des principaux types de données sur PostgreSQL . . . 130
7.2.3 Tables temporaires . . . 132
7.2.4 ALTER TABLE . . . 132
7.2.5 DROP TABLE . . . 133
7.2.6 TRUNCATE TABLE . . . 133
7.3 Les contraintes (CONSTRAINT) sur PostgreSQL . . . 133
7.4 Manipulation d'index . . . 134
7.4.1 CREATE INDEX . . . 134
7.4.2 DROP INDEX . . . 135
7.5 Exercices . . . 135
7.6 Solutions . . . 137

8 DML : Ajouter / modifier / supprimer des données (INSERT, UPDATE, DELETE) 138
8.1 INSERT INTO . . . 139
8.2 UPDATE . . . 140
8.3 DELETE . . . 140
8.4 MERGE . . . 141
8.5 Exercices . . . 142
8.6 Solutions . . . 143

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

9 TCL (Transaction Control Language) : Transactions 145
9.1 TRANSACTION et COMMIT . . . 146
9.2 ROLLBACK . . . 146
9.3 Gestion des erreurs . . . 147
9.4 Exercices . . . 148
9.5 Solutions . . . 149

10 Fonctionnalités avancées 150
10.1 Procédures stockées . . . 151
10.2 Triggers . . . 153
10.3 Exercices . . . 156
10.4 Solutions . . . 158

11 DCL (Data Control Language) : Gestion des droits (GRANT, REVOKE) 162
11.1 Gestion des rôles . . . 163
11.1.1 Création d'un rôle . . . 163
11.1.2 Modifier un rôle . . . 163
11.1.3 Supprimer un rôle . . . 163
11.2 Droits et privilèges . . . 164
11.3 GRANT . . . 164
11.4 REVOKE . . . 164
11.5 Exercices . . . 165
11.6 Solutions . . . 166

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

1. Introduction aux bases de données SQL

1.1 Ressources

Ce document est un support de formation pour le langage SQL sur PostgreSQL.

Il est adapté pour un public débutant ou plus avancé qui voudrait approfondir certains sujets.

Il est possible de contacter l'auteur sur son site : Page contact.

1.2 La base de données

Une base de données est un système organisé permettant de stocker, structurer et exploiter des informations.
L'objectif d'une base de données est de garantir que les données soient :

- Fiables : les informations doivent être exactes et cohérentes
- Accessibles : disponibles au moment où l'utilisateur en a besoin
- Organisées : structurées de manière logique pour faciliter la consultation et l'analyse
- Sécurisées : protégées contre les accès non autorisés ou les pertes
- Partagées : l'accès simultané aux données par plusieurs utilisateurs est permis, des mécanismes de contrôle sont mis en oeuvre afin de garantir la cohérence des données accédées

Une base de données est un ensemble organisé de données stockées de manière structurée.
Elle contient des :

- Tables (clients, produits, commandes, etc.)
- Relations entre ces tables
- Contraintes (clés primaires, clés étrangères…)
- Vues, des index, des procédures, etc.

C'est l'équivalent d'un contenu ou d'un dossier de données structurées. La base de données contient uniquement les données et la structure.

1.3 le Serveur

Le serveur est le logiciel (et parfois la machine) qui exécute et gère les bases de données.

C'est lui qui :

- Stocke physiquement les fichiers de la base
- Reçoit les requêtes SQL
- Traite ces requêtes
- Renvoie les résultats
- Gère la sécurité (utilisateurs, permissions)
- Assure les performances, les sauvegardes, la concurrence, etc.

Le serveur, c'est le "moteur" : c'est lui qui fait fonctionner la base de données.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

1.4 Le Système de Gestion de Base de Données (SGBD)

Un Système de Gestion de Base de Données (SGBD) est un logiciel permettant de créer, organiser, manipuler et administrer une base de données.

Ses rôles principaux :

- Créer et structurer des bases de données
- Assurer l'accès concurrent à plusieurs utilisateurs
- Gérer la sécurité et les permissions
- Optimiser les performances d'accès aux données
- Garantir l'intégrité et la cohérence des informations

Parmi les SGBD les plus utilisés :

- **MySQL / MariaDB** (web, open-source)
- **PostgreSQL** (open-source, très robuste)
- **SQL Server** (Microsoft, usage entreprise)
- **Oracle Database** (entreprises avec grands volumes)
- **SQLite** (embedded, mobile, léger)

Nous utiliserons PostgreSQL dans cette formation.

1.5 Le Modèle Relationnel

Les bases de données relationnelles sont les plus répandues. Elles reposent sur le modèle relationnel, formalisé par Edgar F. Codd.

Le modèle relationnel repose sur 3 concepts clés :

- Les données organisées en Tables (=Relations) qui représentent une entité. Une table représente donc un ensemble d'informations du même type. Par exemple :
  - Clients
  - Produits
  - Commandes
  - Employés
- Colonnes (=Attributs) décrivant les propriétés de l'entité. Elles représentent une information spécifique
- Lignes (=Tuples) pour les occurrences uniques et correspondent à un nouvel enregistrement

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Les tables sont reliées entre elles pour constituer la base de données.

Le langage utilisé pour interagir avec ces données est SQL (Structured Query Language).

La table *Clients* suivante est un exemple de table :

client id	nom	email	Spécialité
1	Noether	noether@jaimeexcel.com	Mathématiques
2	Einstein	einstein@jaimeexcel.com	Physique
3	Codd	codd@jaimeexcel.com	Mathématiques

Chaque colonne correspond à une information particulière et souvent contrainte à un type de données (Texte, Nombre, Date…).

Chaque ligne correspond à un nouvel enregistrement, ici un nouveau client.

1.6 Les contraintes d'intégrité

Les contraintes d'intégrité sont des règles appliquées dans les bases de données relationnelles pour garantir que les données restent cohérentes, fiables et logiques. Elles jouent un rôle essentiel dans la qualité des données et dans le bon fonctionnement du système.

En d'autres termes, elles empêchent l'insertion de données incorrectes ou incohérentes.

1.6.1 Intégrité de domaine

Cette contrainte garantit que les valeurs d'une colonne appartiennent au bon type ou au bon ensemble de valeurs.

Elle s'appuie sur :

- les types de données (INT, VARCHAR, DATE, etc.)
- les contraintes CHECK
- les valeurs par défaut (DEFAULT)

Exemples :

- Une date de naissance doit être une date valide
- Le salaire doit être un nombre positif (CHECK (salaire > 0))
- Le statut d'une commande doit être "payée", "expédiée" ou "annulée" (liste fermée)

1.6.2 Intégrité d'entité

Elle garantit l'unicité des lignes dans une table.

Cela repose sur :

- Une clé primaire (PRIMARY KEY)
- Eventuellement une contrainte UNIQUE

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Exemples :

- Deux clients ne peuvent pas avoir le même identifiant.
- Un numéro de commande ne peut apparaître qu'une seule fois.

1.6.3 Intégrité référentielle

C'est l'une des notions les plus importantes.
Elle garantit la cohérence des relations entre les tables grâce aux clés étrangères (FOREIGN KEY ou FK).
Elle impose que :

- Une valeur dans une colonne référencée existe réellement dans la table parent
- On ne puisse pas supprimer une valeur utilisée ailleurs (sauf règles spécifiées)

Exemple :

- Dans une table Commandes, id_client doit correspondre à un client existant
- Impossible d'avoir une commande pour un client qui n'existe pas

Les actions possibles sur une clé étrangère :

- **ON DELETE RESTRICT** → interdiction de supprimer si des données liées existent
- **ON DELETE CASCADE** → suppression automatique des données liées
- **ON UPDATE CASCADE** → mise à jour automatique de la FK

1.6.4 Intégrité transactionnelle

Elle garantit que les opérations réalisées ensemble sont :

- Atomiques (tout ou rien)
- Cohérentes
- Isolées des autres transactions
- Persistantes

C'est le principe des propriétés **ACID** :

- Atomicité
- Cohérence
- Isolation
- Durabilité

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Exemple :

Un virement bancaire implique deux opérations : débiter + créditer. Elles doivent être exécutées ensemble ou pas du tout.

Les règles d'intégrité permettent de :

- Éviter les incohérences (ex. commandes sans client)
- Éviter les doublons (ex. deux employés avec le même id)
- Assurer la qualité et la fiabilité des données
- Améliorer la robustesse des applications
- Réduire les erreurs humaines
- Garantir que les règles métier sont respectées par le système

Sans contraintes, une base devient rapidement incohérente et inutilisable.

1.7 Les clés : garantie de la cohérence et de l'unicité

Les bases relationnelles s'appuient sur des clés pour assurer l'intégrité des données.

Clé primaire (Primary Key – PK)

- Identifie de manière unique chaque ligne
- Impossible à dupliquer ou à laisser vide
- Exemples : id_client, n° de commande

Clé étrangère (Foreign Key – FK)

- Crée une relation entre deux tables
- Référence la clé primaire d'une autre table

Par exemple :

Reprenons la table *Clients* vue plus haut :

client id	nom	email	Spécialité
1	Noether	noether@jaimeexcel.com	Mathématiques
2	Einstein	einstein@jaimeexcel.com	Physique
3	Codd	codd@jaimeexcel.com	Mathématiques

Ajoutons la table de *Commandes* :

Commande_id	Client_id	Produit	Quantité
1001	3	A	15
1002	1	A	12

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Commande_id	Client_id	Produit	Quantité
1003	1	B	18

La table *Commandes* contient Client_id dans une colonne FK (clé étrangère) pointant vers *Clients*.Client_id (cette syntaxe signifie la colonne Client_id appartenant à la table *Clients*)

1.8 La normalisation des bases de données

La normalisation est un ensemble de règles visant à organiser les données d'une base de données relationnelle de manière logique, cohérente et efficace.

L'objectif est de :

- Éliminer les redondances (éviter de stocker plusieurs fois la même information)
- Garantir l'intégrité des données
- Simplifier les mises à jour et éviter les anomalies
- Améliorer la clarté du modèle de données

Elle repose principalement sur la création de tables cohérentes et la définition de relations bien structurées entre elles.

1.8.1 Pourquoi normaliser ?

Sans normalisation, une base peut souffrir de problèmes tels que :

Redondance

La même donnée est répétée plusieurs fois, augmentant :

- Le risque d'incohérence
- Le volume de stockage
- Les coûts de maintenance

Anomalies

Lorsqu'une structure de table est mal conçue, certaines opérations deviennent problématiques :

- **Anomalie de mise à jour** : une information doit être mise à jour à plusieurs endroits
- **Anomalie d'insertion** : l'insertion d'une donnée est impossible sans une autre donnée
- **Anomalie de suppression** : la suppression d'une donnée entraîne la perte d'informations importantes

Pour guider la normalisation, on utilise des règles appelées formes normales. Les trois premières (1NF, 2NF, 3NF) sont les plus utilisées et suffisantes dans la grande majorité des cas.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

1.8.2 Première forme Normale (1NF)

La normalisation en première forme normale (1NF) respecte :

- L'unicité des lignes : idéalement avec une clé primaire pour identifiant unique
- Pas de colonnes dupliquées et chaque colonne a un type de données défini
- L'atomicité : chaque cellule de la colonne ne contient que des valeurs atomiques

Exemple violant la 1NF :

Formateur_id	Domaine	Nom	Pays	Ville
1	Mathématiques, Physique	Noether	Allemagne	Gottingen
2	Physique, Chimie	Curie	France	Paris
3	Physique	Planck	Allemagne	Berlin

Ici, la colonne Domaine contient plusieurs valeurs séparées par une virgule. L'atomicité n'est pas respectée.

Le processus de normalisation 1NF conduit au résultat suivant :

Formateur_id	Domaine	Nom	Pays	Ville
1	Mathématiques	Noether	Allemagne	Gottingen
1	Physique	Noether	Allemagne	Gottingen
2	Physique	Curie	France	Paris
2	Chimie	Curie	France	Paris
3	Physique	Planck	Allemagne	Berlin

L'atomicité est maintenant respectée.
En revanche, on remarque que les éléments de la colonne Formateur_id ne sont plus uniques.

2 solutions possibles à ce stade :

- Garder la table telle quelle et définir une clé composite (clé primaire composée de plusieurs colonnes) : ici par exemple {Formateur_id, Domaine} définissent de manière unique chaque ligne
- Ajouter une colonne en tant que clé primaire, nommée Enreg_ID par exemple à la place de la clé composite (solution recommandée)

L'atomicité rend les données plus propres et plus simples à lire, à interroger ou à filtrer.

Elle évite les anomalies d'insertion, de mise à jour ou de suppression car chaque élément se trouve dans sa propre ligne.

Elle réduit donc le risque d'incohérences et l'intégrité des données.

1.8.3 Deuxième forme normale (2NF)

La normalisation en deuxième forme normale (2NF) respecte :

- Les conditions de la 1NF (atomicité et clé primaire)

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

- L'élimination des dépendances partielles. Autrement dit, il doit y avoir une relation directe entre chaque colonne et la clé primaire (et non un sous-ensemble dans le cas d'une clé primaire composite)

Cela concerne surtout les clés primaires composites. Le problème se pose moins pour une clé primaire simple. On évite qu'une colonne dépende seulement d'une partie de la clé.

Reprenons la table normalisée (1NF) précédente :

Formateur_id	Domaine	Nom	Pays	Ville
1	Mathématiques	Noether	Allemagne	Gottingen
1	Physique	Noether	Allemagne	Gottingen
2	Physique	Curie	France	Paris
2	Chimie	Curie	France	Paris
3	Physique	Planck	Allemagne	Berlin

Si on considère la clé composite (Formateur_id + Domaine) par exemple, on constate que les 3 dernières colonnes dépendent en réalité uniquement de qui est le formateur (Formateur_id).

On voit bien ici que si l'on veut modifier le Nom, la Ville ou le Pays d'un client, il faudrait le faire dans plusieurs lignes à la fois.
Ce qui pose un problème de redondance et des anomalies potentielles (en cas d'erreur de saisie).

La solution est alors de séparer les données en plusieurs tables :

- Une Table *Formateurs* avec les colonnes qui dépendent directement de Formateur_id
- Une Table d'affectations avec sa clé primaire composite : Formateur_id et le Domaine

Formateurs :

Formateur_id	Nom	Pays	Ville
1	Noether	Allemagne	Gottingen
2	Curie	France	Paris
3	Planck	Allemagne	Berlin

Affectations :

Formateur_id	Domaine
1	Mathématiques
1	Physique
2	Physique
2	Chimie
3	Physique

Les dépendances partielles ont disparu.

La normalisation 2NF minimise les anomalies d'insertion, de mise à jour et de suppression donc conduit à une base plus fiable.

Elle réduit la redondance de données, nous n'avons plus à répéter toutes les données des formateurs plusieurs fois dans notre exemple. On a donc une optimisation du stockage et une simplification de la maintenance de la base de données.

Une base de données plus propre et efficace.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

1.8.4 Troisième forme normale (3NF)

La normalisation en troisième forme normale (3NF) respecte :

- Les conditions de la 1NF (atomicité et clé primaire)
- Les conditions de la 2NF (suppression des dépendances partielles sur une clé primaire composite)
- L'élimination des dépendances transitives. Autrement dit, chaque colonne doit dépendre directement de la clé primaire et de rien d'autre

Reprenons la table des formateurs ci-dessus (avec un formateur en plus) :

Formateur_id	Nom	Pays	Ville
1	Noether	Allemagne	Gottingen
2	Curie	France	Paris
3	Planck	Allemagne	Berlin
4	Heisenberg	Allemagne	Gottingen

La colonne Pays ne dépend pas directement de Formateur_iD mais plutôt de la colonne Ville. On a donc une dépendance transitive.

Le Pays dépendant directement de la Ville, on peut éviter la redondance de données en supprimant la dépendance transitive. Par ailleurs, on évite des anomalies potentielles (en cas d'erreur de saisie).

La solution est alors de séparer les données en plusieurs tables :

- Une Table *Villes* avec les colonnes Ville et Pays, ainsi qu'une clé primaire Ville_iD
- La Table *Formateurs* avec sa clé primaire et les colonnes dépendant directement de cette clé : Nom et la clé étrangère Ville_iD faisant le lien avec la table Villes

Villes :

Ville_id	Nom_ville	Pays
1	Gottingen	Allemagne
2	Paris	France
3	Berlin	Allemagne

Formateurs :

Formateur_id	Nom	Ville_id
1	Noether	1
2	Curie	2
3	Planck	3
4	Heisenberg	1

C'est la dernière étape de normalisation dans la majorité des cas.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

1.9 Le langage SQL

SQL est un langage normalisé pour interagir avec les bases de données relationnelles.

SQL est un langage déclaratif : on décrit le résultat attendu, et le SGBD détermine comment l'obtenir de manière optimale.

SQL est organisé en plusieurs catégories :

- DQL (Data Query Language) : interroger (SELECT)
- DML (Data Manipulation Language) : ajouter / modifier / supprimer (INSERT, UPDATE, DELETE)
- DDL (Data Definition Language) : création et structure (CREATE TABLE, ALTER, DROP)
- DCL (Data Control Language) : gestion des droits (GRANT, REVOKE)
- TCL (Transaction Control Language) : transactions (COMMIT, ROLLBACK)

1.10 Installation de PostgreSQL

Il existe de multiples options pour travailler sur PostgreSQL.

On peut télécharger PostgreSQL sur ce lien.

1.11 Base de données de travail

Dans le cadre de la formation, nous travaillerons sur la base de données JETHLON.

Cette base de données est téléchargeable au lien suivant.

Une fois le fichier DB_JETHLON téléchargé, il suffit de restaurer la base de données dans pgadmin (installé avec PosqtgreSQL).

Le schéma de la base de données se trouve dans la page suivante.

La base de données est celle d'un petit magasin de ventes d'articles de sport. Elle est constituée des tables suivantes :

- *Base N* : Table des ventes
- *Base N-1* : Table des ventes de l'année N-1
- *Base N-2* : Table des ventes de l'année N-2
- *Produits* : Table listant les produits
- *Sous categories* : Table listant les sous catégories de produit
- *Categories* : Table listant les catégories
- *Vendeurs* : Table listant les vendeurs
- *Magasins* : Table listant les magasins
- *Clients* : Table listant les clients
- *Geographie* : Table de géographie

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

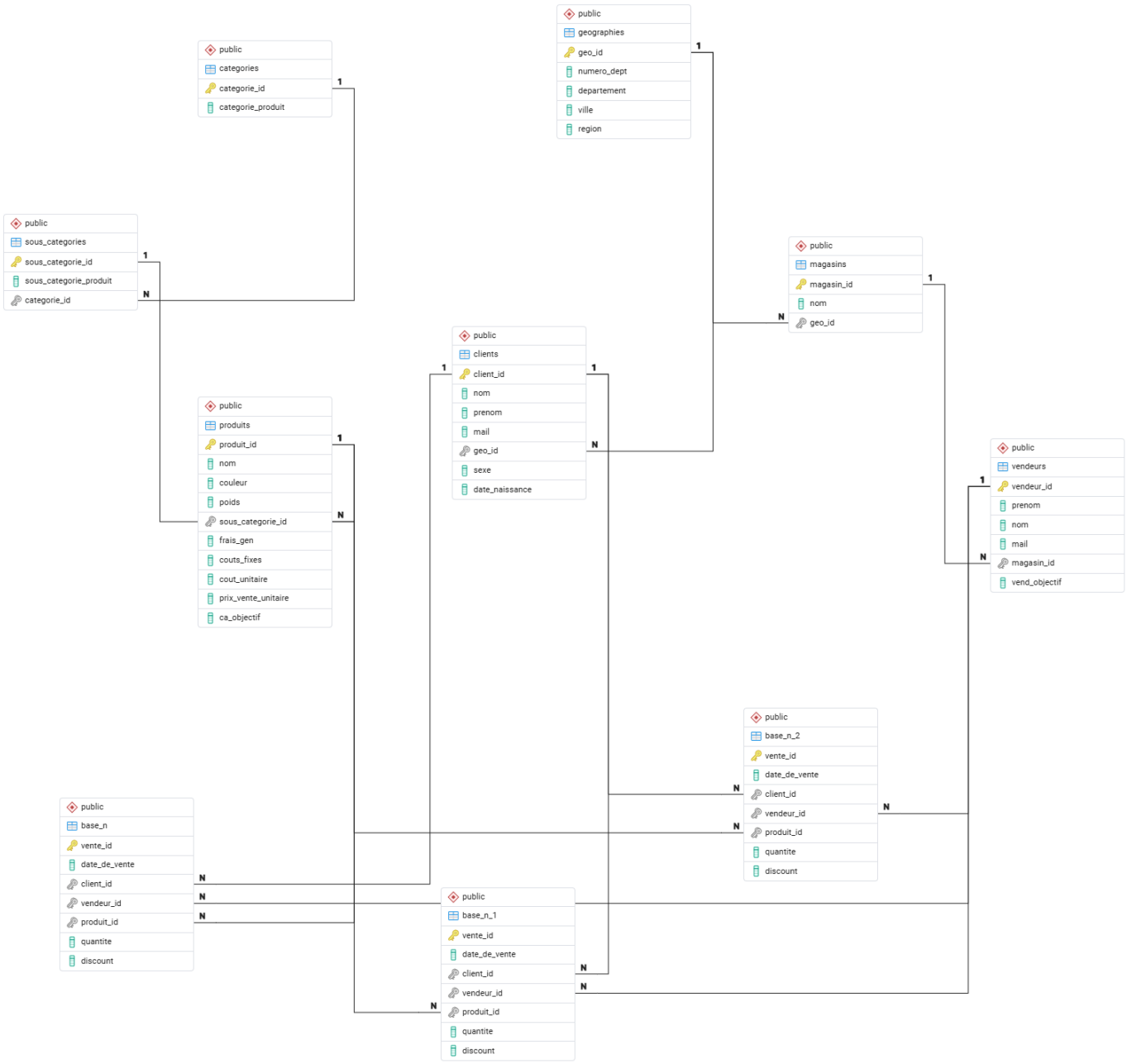

Figure 1: Schéma de la base JETHLON

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

2. Extraire des données

2.1 SELECT ... FROM

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

La syntaxe de base est la suivante :

```
SELECT nom_de_colonne1, nom_de_colonne2...
FROM nom_de_table
```

Si on prend la table *Vendeurs* de la base de données :

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
1	Niels	Bohr	niels.bohr@jaimeexcel.com	1	3800000
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
3	Emmy	Noether	emmy.noether@jaimeexcel.com	2	4030000
4	Isaac	Newton	isaac.newton@jaimeexcel.com	2	4050000
5	Marie	Curie	marie.curie@jaimeexcel.com	3	3900000
6	Erwin	Schrodingher	erwin.schrodingher@jaimeexcel.com	3	3800000

Si on veut avoir la liste des nom et prénom des vendeurs, on peut effectuer la requête suivante :

```
SELECT nom, prenom
FROM Vendeurs
```

Résultat :

nom	prenom
Bohr	Niels
Einstein	Albert
Noether	Emmy
Newton	Isaac
Curie	Marie
Schrodingher	Erwin

Il est possible de retourner automatiquement toutes les colonnes d'un tableau sans avoir à connaître le nom de toutes les colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère « * » (étoile). Il s'utilise de la manière suivante :

```
SELECT *
FROM Vendeurs
```

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
1	Niels	Bohr	niels.bohr@jaimeexcel.com	1	3800000

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
3	Emmy	Noether	emmy.noether@jaimeexcel.com	2	4030000
4	Isaac	Newton	isaac.newton@jaimeexcel.com	2	4050000
5	Marie	Curie	marie.curie@jaimeexcel.com	3	3900000
6	Erwin	Schrodingher	erwin.schrodingher@jaimeexcel.com	3	3800000

Il est déconseillé d'utiliser * quand c'est évitable.

2.2 WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées. C'est donc une sorte de filtre.

La syntaxe de base est la suivante :

```
SELECT nom_de_colonnes
FROM nom_de_table
WHERE condition
```

Si on reprend notre table *Vendeurs*, on remarque que les vendeurs appartiennent à des magasins (1,2 ou 3).

Pour obtenir la liste des vendeurs qui appartiennent au magasin_id 2, on écrit la requête suivante :

```
SELECT *
FROM Vendeurs
WHERE magasin_id = 2
```

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
3	Emmy	Noether	emmy.noether@jaimeexcel.com	2	4030000
4	Isaac	Newton	isaac.newton@jaimeexcel.com	2	4050000

2.2.1 Opérateurs de comparaisons

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques-uns des opérateurs les plus couramment utilisés.

Opérateur	Description
=	Egale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Opérateur	Description
BETWEEN	Valeurs comprises dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Attention : il y a des opérateurs qui n'existent pas dans les vieilles versions de système de gestion de bases de données (SGBD). De plus, il y a de nouveaux opérateurs non indiqués ici qui sont disponibles avec certains SGBD. N'hésitez pas à consulter la documentation de SQLServer pour voir ce qu'il vous est possible de faire.

2.2.2 AND & OR

Une requête SQL peut être restreinte à l'aide de la condition WHERE. Les opérateurs logiques AND et OR peuvent être utilisés au sein de la commande WHERE pour combiner des conditions.

Les opérateurs sont à ajoutés dans la condition WHERE. Ils peuvent être combinés à l'infini pour filtrer les données comme souhaités.

La syntaxe de base est la suivante :

- AND :

```
SELECT nom_de_colonnes
FROM nom_de_table
WHERE condition1 AND condition2
```

Par exemple, si on veut les vendeurs du magasin 1 et dont les objectifs de vente (vend_objectifs) sont supérieurs à 4000 000.

```
SELECT *
FROM Vendeurs
WHERE magasin_id = 1 AND vend_objectif > 4000000
```

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000

- OR :

```
SELECT nom_de_colonnes
FROM nom_de_table
WHERE condition1 OR condition2
```

Par exemple, si on veut les vendeurs du magasin 1 ou du magasin 3.

```
SELECT *
FROM Vendeurs
WHERE magasin_id = 1 OR magasin_id = 3
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
1	Niels	Bohr	niels.bohr@jaimeexcel.com	1	3800000
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
5	Marie	Curie	marie.curie@jaimeexcel.com	3	3900000
6	Erwin	Schrodingher	erwin.schrodingher@jaimeexcel.com	3	3800000

- Combinaison :

```
SELECT nom_de_colonnes
FROM nom_de_table
WHERE condition1 AND (condition2 OR condition3)
```

Ci-dessus, la requête filtre les résultats de la table si condition1 ET condition2 OU condition3 est vrai. Ne pas oublier les parenthèses quand c'est nécessaire.

Par exemple, si on veut les vendeurs avec un objectif de plus de 4000 000 et travaillant dans le magasin 1 ou du magasin 2.

```
SELECT *
FROM Vendeurs
WHERE vend_objectif > 4000000 AND (magasin_id = 1 OR magasin_id = 2)
```

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
3	Emmy	Noether	emmy.noether@jaimeexcel.com	2	4030000
4	Isaac	Newton	isaac.newton@jaimeexcel.com	2	4050000

2.2.3 IN

L'opérateur logique IN dans SQL s'utilise avec la commande WHERE pour vérifier si une colonne est égale à une valeur comprise dans un ensemble de valeurs déterminés. C'est une méthode simple pour vérifier si une colonne est égale à une valeur OU une autre valeur OU une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l'opérateur OR.

La syntaxe de base est la suivante :

```
SELECT nom_de_colonne
FROM nom_de_table
WHERE nom_de_colonne IN (valeur1, valeur2, valeur3...)
```

Entre les parenthèses il n'y a pas de limite du nombre d'arguments. Il est possible d'ajouter encore d'autres valeurs. Cette syntaxe peut être associée à l'opérateur NOT pour recherche toutes les lignes qui ne sont pas égales à l'une des valeurs stipulées.

En reprenant notre table *Vendeurs*, on peut vouloir afficher les lignes correspondant aux prénoms 'Albert', 'Erwin' ou 'Marie'. On a alors deux façons de faire :

- Avec OR :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT *
FROM Vendeurs
WHERE prenom = 'Albert' OR prenom = 'Erwin' OR prenom = 'Marie'
```

- Avec IN :

```sql
SELECT *
FROM Vendeurs
WHERE prenom IN ('Albert','Erwin','Marie')
```

Le résultat est évidemment le même :

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
5	Marie	Curie	marie.curie@jaimeexcel.com	3	3900000
6	Erwin	Schrodingher	erwin.schrodingher@jaimeexcel.com	3	3800000

2.2.4 BETWEEN

L'opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. L'exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

La syntaxe de base est la suivante :

```sql
SELECT nom_de_colonne
FROM nom_de_table
WHERE nom_de_colonne BETWEEN 'valeur1' AND 'valeur2'
```

La requête ci-dessus retournera toutes les lignes dont la valeur de la colonne « nom_colonne » sera comprise entre valeur1 et valeur2.

- Filtre entre 2 Dates :

On veut par exemple filtrer les ventes de la table *Base_N* ayant eu lieu entre le 01/05/2025 et le 05/05/2025 :

```sql
SELECT *
FROM Base_N
WHERE date_de_vente BETWEEN '2025-05-01' AND '2025-05-05'
```

vente_id	date_de_vente	client_id	vendeur_id	produit_id	quantite	discount
V2025000045	2025-05-05	15507	5	1	4	0.00
V2025000060	2025-05-05	20792	5	2	2	0.06
V2025000078	2025-05-02	260	3	2	3	0.01
V2025000197	2025-05-04	10114	1	4	1	0.08

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vente_id	date_de_vente	client_id	vendeur_id	produit_id	quantite	discount
V2025000219	2025-05-03	15606	4	4	3	0.10
V2025000242	2025-05-04	18710	4	5	3	0.00
V2025000265	2025-05-02	14847	5	5	3	0.04
V2025000484	2025-05-04	2784	4	9	1	0.01
V2025000499	2025-05-02	13319	1	9	3	0.03
V2025000651	2025-05-03	22814	6	12	2	0.07
V2025000672	2025-05-02	16229	2	12	2	0.01
V2025000674	2025-05-01	8978	3	12	1	0.00

On affiche ici les premiers résultats uniquement.

- Filtre entre 2 nombres :

On peut vouloir obtenir les résultats de la table des *Sous_categories* dont le numéro (sous_categorie_id) n'est pas compris entre 7 et 13 :

```
SELECT *
FROM Sous_categories
WHERE sous_categorie_id NOT BETWEEN 7 AND 13
```

sous_categorie_id	sous_categorie_produit	categorie_id
1	Velo electrique	1
2	Velo de Route	1
3	VTT	1
4	VTC	1
5	Batterie	2
6	Cables	2
14	T-shirt	4
15	Short	4
16	Sous-vêtement	4
17	Maillot de bain	5
18	Bonnet de bain	5
19	Montre connectee	6

2.2.5 LIKE

L'opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé permet de rechercher les enregistrements dont la valeur d'une colonne commence, finit ou contient une chaîne de caractère.

Il existe plusieurs cas de figure :

- LIKE '%abc' : Le caractère % est un caractère spécial qui remplace tous les autres caractères. Ainsi dans cet exemple, on recherche toutes les chaines de caractères qui se terminent par "abc".
- LIKE 'abc%' : Permet de rechercher toutes les chaines qui commencent par "abc".
- LIKE '%abc%': Permet de rechercher toutes les chaines qui contiennent "abc".
- LIKE 'la%age' : Permet de rechercher les chaines qui commencent par "la" et qui se terminent par "age", comme "langage" ou "lavage".

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

La syntaxe de base est la suivante :

```
SELECT nom_de_colonne
FROM nom_de_table
WHERE nom_de_colonne LIKE condition
```

Dans la table des *Produits*, on peut vouloir afficher les informations de tous les produits dont le nom commence par “VTT” :

```
SELECT nom, couleur, poids, cout_unitaire, prix_vente_unitaire
FROM Produits
WHERE nom LIKE 'VTT%'
```

nom	couleur	poids	cout_unitaire	prix_vente_unitaire
VTT1000	noir	14	850	2650
VTT1001	bleu	14	900	2700
VTT1002	rouge	14	900	2700
VTT1003	vert	14	900	2700
VTT1004	jaune	14	900	2700
VTT1005	rose	14	900	2700

On peut aussi vouloir les information sur ceux dont le nom finit par “VTT” cette fois ci :

```
SELECT nom, couleur, poids, cout_unitaire, prix_vente_unitaire
FROM Produits
WHERE nom LIKE '%VTT'
```

nom	couleur	poids	cout_unitaire	prix_vente_unitaire
Cables_VTT	noir	1	46.75	85
Freins_VTT	noir	1	5.50	10
Selle_VTT	noir	1	33.00	60

2.2.6 IS NULL / IS NOT NULL

Dans le langage SQL, l’opérateur IS permet de filtrer les résultats qui contiennent la valeur NULL. Cet opérateur est indispensable car la valeur NULL est une valeur inconnue et ne peut pas être filtrée par les opérateurs de comparaison (égal, inférieur, supérieur ou différent).

La syntaxe de base est la suivante :

```
SELECT *
FROM nom_de_table
WHERE nom_de_colonne IS NULL
```

ou

```
SELECT *
FROM nom_de_table
WHERE nom_de_colonne IS NOT NULL
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Nous pouvons afficher la liste des clients (table *Clients*) dont le mail est renseigné (mail non null donc) :

```
SELECT *
FROM Clients
WHERE mail IS NOT NULL
```

client_id	nom	prenom	mail	geo_id	sexe	date_naissance
1	nom1	prenom1	nom1.prenom1@gmail.com	11	homme	1991-05-06
2	nom2	prenom2	nom2.prenom2@gmail.com	51	femme	1975-09-04
3	nom3	prenom3	nom3.prenom3@gmail.com	65	homme	1991-10-26
4	nom4	prenom4	nom4.prenom4@gmail.com	86	femme	1997-07-11
5	nom5	prenom5	nom5.prenom5@gmail.com	83	femme	1967-05-10
6	nom6	prenom6	nom6.prenom6@gmail.com	103	homme	1966-07-07
7	nom7	prenom7	nom7.prenom7@gmail.com	38	femme	1989-09-29
8	nom8	prenom8	nom8.prenom8@gmail.com	34	femme	2001-01-21
9	nom9	prenom9	nom9.prenom9@gmail.com	4	homme	2000-09-18
10	nom10	prenom10	nom10.prenom10@gmail.com	66	femme	1999-05-03
11	nom11	prenom11	nom11.prenom11@gmail.com	31	homme	1976-06-25
12	nom12	prenom12	nom12.prenom12@gmail.com	21	femme	1967-12-14

On affiche seulement les premiers résultats ci-dessus.

2.3 DISTINCT

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.

La syntaxe de base est la suivante :

```
SELECT DISTINCT nom_de_colonne
FROM nom_de_table
```

Par exemple, pour afficher la liste des numéros de vendeurs qui ont vendu l'année N :

```
SELECT DISTINCT vendeur_id
FROM Base_N
```

vendeur_id
6
1
3
5
4
2

L'utilisation de la commande DISTINCT est donc très pratique pour éviter les résultats en doubles.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

2.4 Exercices

1. Afficher les colonnes nom, couleur et poids des produits de couleur “rouge”.

nom	couleur	poids
E_bike1002	rouge	25
Road_bike1002	rouge	8
VTT1002	rouge	14
Tshirt	rouge	0

2. Afficher les colonnes nom, couleur et poids des produits dont la couleur n’est pas le “bleu”.

nom	couleur	poids
E_bike1000	noir	28
E_bike1002	rouge	25
Road_bike1000	noir	8
Road_bike1002	rouge	8
Road_bike1003	vert	8
Road_bike1004	jaune	8
VTT1000	noir	14
VTT1002	rouge	14
VTT1003	vert	14
VTT1004	jaune	14
VTT1005	rose	14
VTC1000	noir	11
VTC1002	rose	11
Batterie1000	noir	2
Batterie1001	noir	2
Cables_Ebike	noir	1
Cables_Road_bike	noir	1
Cables_VTT	noir	1
Cables_VTC	noir	1
Freins_Ebike	noir	1
Freins_Road_bike	noir	1
Freins_VTT	noir	1
Freins_VTC	noir	1
Selle_Ebike	noir	1
Selle_Road_bike	noir	1
Selle_VTT	noir	1
Selle_VTC	noir	1
Combinaison	noir	1
Bike_shoes	noir	0
Oesics_run	noir	0
Oesics_run	blanc	0
Oesics_run	orange	0
Abidas_shoes	noir	0
Abidas_shoes	blanc	0
Abidas_shoes	orange	0
Mike_shoes	noir	0
Mike_shoes	blanc	0

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	couleur	poids
Mike_shoes	orange	0
Oesics_run_socks	noir	0
Oesics_run_socks	blanc	0
Abidas_run_socks	noir	0
Abidas_run_socks	blanc	0
Mike_run_socks	noir	0
Mike_run_socks	blanc	0
Semelle_running	noir	0
Tshirt	noir	0
Tshirt	blanc	0
Tshirt	jaune	0
Tshirt	rose	0
Tshirt	orange	0
Tshirt	rouge	0
Tshirt	vert	0
Short	noir	0
Short	blanc	0
Short	rose	0
Short	orange	0
SV_Homme	noir	0
SV_Femme	noir	0
Maillot_de_bain_Homme	noir	0
Maillot_de_bain_Homme	blanc	0
Maillot_de_bain_Femme	noir	0
Maillot_de_bain_Femme	blanc	0
Bonnet_de_bain	noir	0
Bonnet_de_bain	blanc	0
JE_watch	noir	0
JE_watch	blanc	0
JE_watch	vert	0
JE_watch	rose	0
JE_watch	gris	0

3. Extraire les ventes de *Base_N* avec quantité = 1 et pas de remise.

vente_id	date_de_vente	client_id	vendeur_id	produit_id	quantite	discount
V2025000055	2025-10-04	21418	1	2	1	0
V2025000083	2025-08-11	20669	4	2	1	0
V2025000187	2025-07-30	16846	3	4	1	0
V2025000208	2025-08-07	2113	4	4	1	0
V2025000239	2025-04-30	9898	1	5	1	0
V2025000268	2025-07-30	3216	4	5	1	0
V2025000290	2025-12-11	22833	4	6	1	0
V2025000352	2025-07-22	19909	5	7	1	0
V2025000456	2025-12-20	9535	3	9	1	0
V2025000463	2025-05-24	389	3	9	1	0
V2025000472	2025-10-08	2790	1	9	1	0
V2025000529	2025-07-12	14917	4	10	1	0
V2025000574	2025-11-07	18263	2	11	1	0
V2025000648	2025-09-26	11199	3	12	1	0

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vente_id	date_de_vente	client_id	vendeur_id	produit_id	quantite	discount
V2025000674	2025-05-01	8978	3	12	1	0
V2025000687	2025-07-22	12167	2	13	1	0
V2025000688	2025-11-18	7821	6	13	1	0
V2025000712	2025-09-10	9693	2	13	1	0
V2025000715	2025-11-29	13521	4	13	1	0
V2025000723	2025-12-10	1005	4	13	1	0

(Seuls les premiers résultats sont affichés ci-dessus).

4. Afficher les produits de couleur “rouge” ou “bleu”.

nom	couleur
E_bike1001	bleu
E_bike1002	rouge
Road_bike1001	bleu
Road_bike1002	rouge
VTT1001	bleu
VTT1002	rouge
VTC1001	bleu
Oesics_run	bleu
Abidas_shoes	bleu
Mike_shoes	bleu
Tshirt	bleu
Tshirt	rouge

5. Récupérer les clients vivant dans certains départements (38,69,75).

nom	prenom	geo_id
nom7	prenom7	38
nom47	prenom47	69
nom110	prenom110	38
nom131	prenom131	75
nom152	prenom152	38
nom232	prenom232	75
nom253	prenom253	75
nom256	prenom256	69
nom286	prenom286	38
nom345	prenom345	69
nom389	prenom389	75
nom392	prenom392	75
nom462	prenom462	75
nom509	prenom509	69
nom599	prenom599	75
nom718	prenom718	75
nom727	prenom727	75
nom771	prenom771	69
nom780	prenom780	69

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	prenom	geo_id
nom787	prenom787	75

(Seuls les premiers résultats sont affichés ci-dessus).

6. Lister les vendeurs qui ont une adresse mail valide (non vide).

nom	prenom
Bohr	Niels
Einstein	Albert
Noether	Emmy
Newton	Isaac
Curie	Marie
Schrodingher	Erwin

7. Récupérer la liste des produits dont le nom contient “VTC”.

produit_id	nom	couleur	poids
15	VTC1000	noir	11
16	VTC1001	bleu	11
17	VTC1002	rose	11
23	Cables_VTC	noir	1
27	Freins_VTC	noir	1
31	Selle_VTC	noir	1

8. Récupérer les clients dont l’email finit par “@gmail.com”

nom	prenom	mail
nom1	prenom1	nom1.prenom1@gmail.com
nom2	prenom2	nom2.prenom2@gmail.com
nom3	prenom3	nom3.prenom3@gmail.com
nom4	prenom4	nom4.prenom4@gmail.com
nom5	prenom5	nom5.prenom5@gmail.com
nom6	prenom6	nom6.prenom6@gmail.com
nom7	prenom7	nom7.prenom7@gmail.com
nom8	prenom8	nom8.prenom8@gmail.com
nom9	prenom9	nom9.prenom9@gmail.com
nom10	prenom10	nom10.prenom10@gmail.com
nom11	prenom11	nom11.prenom11@gmail.com
nom12	prenom12	nom12.prenom12@gmail.com
nom13	prenom13	nom13.prenom13@gmail.com
nom14	prenom14	nom14.prenom14@gmail.com
nom15	prenom15	nom15.prenom15@gmail.com
nom16	prenom16	nom16.prenom16@gmail.com
nom17	prenom17	nom17.prenom17@gmail.com
nom18	prenom18	nom18.prenom18@gmail.com

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	prenom	mail
nom19	prenom19	nom19.prenom19@gmail.com
nom20	prenom20	nom20.prenom20@gmail.com

(Seuls les premiers résultats sont affichés ci-dessus).

9. Lister les différentes couleurs de produit sans doublon.

couleur
gris
bleu
rouge
blanc
orange
vert
jaune
rose
noir

10. Afficher les produits dont le prix de vente est entre 1000 et 5000 (inclus).

nom	prix_vente_unitaire
Road_bike1000	3700
Road_bike1001	3900
Road_bike1002	3900
Road_bike1003	3900
Road_bike1004	3900
VTT1000	2650
VTT1001	2700
VTT1002	2700
VTT1003	2700
VTT1004	2700
VTT1005	2700

11. Afficher les produits qui ne sont pas dans les sous-catégories 1, 2 ou 3.

nom	sous_categorie_id
VTC1000	4
VTC1001	4
VTC1002	4
Batterie1000	5
Batterie1001	5
Cables_Ebike	6
Cables_Road_bike	6
Cables_VTT	6
Cables_VTC	6

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	sous_categorie_id
Freins_Ebike	7
Freins_Road_bike	7
Freins_VTT	7
Freins_VTC	7
Selle_Ebike	8
Selle_Road_bike	8
Selle_VTT	8
Selle_VTC	8
Combinaison	9
Bike_shoes	10
Oesics_run	11
Oesics_run	11
Oesics_run	11
Oesics_run	11
Abidas_shoes	11
Abidas_shoes	11
Abidas_shoes	11
Abidas_shoes	11
Mike_shoes	11
Mike_shoes	11
Mike_shoes	11
Mike_shoes	11
Oesics_run_socks	12
Oesics_run_socks	12
Abidas_run_socks	12
Abidas_run_socks	12
Mike_run_socks	12
Mike_run_socks	12
Semelle_running	13
Tshirt	14
Tshirt	14
Tshirt	14
Tshirt	14
Tshirt	14
Tshirt	14
Tshirt	14
Tshirt	14
Short	15
Short	15
Short	15
Short	15
SV_Homme	16
SV_Femme	16
Maillot_de_bain_Homme	17
Maillot_de_bain_Homme	17
Maillot_de_bain_Femme	17
Maillot_de_bain_Femme	17
Bonnet_de_bain	18
Bonnet_de_bain	18
JE_watch	19
JE_watch	19
JE_watch	19

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	sous_categorie_id
JE_watch	19
JE_watch	19

12. Récupérer les ventes réalisées entre le 01/05/2025 et 30/06/2025.

vente_id	date_de_vente
V2025000002	2025-05-10
V2025000004	2025-06-24
V2025000005	2025-05-27
V2025000012	2025-06-30
V2025000019	2025-06-16
V2025000023	2025-05-19
V2025000026	2025-06-17
V2025000038	2025-05-31
V2025000039	2025-05-27
V2025000041	2025-06-06
V2025000045	2025-05-05
V2025000047	2025-05-27
V2025000054	2025-06-16
V2025000057	2025-06-26
V2025000060	2025-05-05
V2025000066	2025-06-03
V2025000067	2025-06-10
V2025000078	2025-05-02
V2025000086	2025-05-10
V2025000098	2025-06-24
V2025000099	2025-06-12
V2025000104	2025-05-15
V2025000105	2025-06-30
V2025000106	2025-06-24
V2025000109	2025-05-16

(Seuls les premiers résultats sont affichés ci-dessus).

13. Lister les clients nés entre 1990 et 2000 et de sexe féminin.

nom	prenom	date_naissance
nom4	prenom4	1997-07-11
nom10	prenom10	1999-05-03
nom30	prenom30	2000-12-20
nom42	prenom42	1993-02-27
nom53	prenom53	1998-06-08
nom56	prenom56	1996-10-28
nom58	prenom58	1999-01-27
nom61	prenom61	1998-09-07
nom70	prenom70	1996-08-30
nom71	prenom71	1994-02-23

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	prenom	date_naissance
nom73	prenom73	1993-06-03
nom74	prenom74	1996-11-28
nom77	prenom77	1995-05-31
nom79	prenom79	1994-11-14
nom85	prenom85	1991-06-08
nom94	prenom94	1995-01-20
nom99	prenom99	1997-04-10
nom100	prenom100	1994-07-19
nom110	prenom110	1991-06-03
nom127	prenom127	1990-12-12
nom130	prenom130	1997-06-20
nom136	prenom136	1991-01-25
nom140	prenom140	1995-01-25
nom159	prenom159	1996-05-18
nom166	prenom166	1992-05-29

(Seuls les premiers résultats sont affichés ci-dessus).

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

2.5 Solutions

1. Afficher les colonnes nom, couleur et poids des produits de couleur “rouge”.

```sql
SELECT nom, couleur, poids
FROM Produits
WHERE couleur = 'rouge'
```

2. Afficher les colonnes nom, couleur et poids des produits dont la couleur n’est pas le “bleu”.

```sql
SELECT nom, couleur, poids
FROM Produits
WHERE couleur <> 'bleu'
```

3. Extraire les ventes de *Base_N* avec quantité = 1 et pas de remise.

```sql
SELECT *
FROM Base_N
WHERE quantite = 1 AND discount = 0
```

4. Afficher les produits de couleur “rouge” ou “bleu”.

```sql
SELECT nom, couleur
FROM Produits
WHERE couleur = 'rouge' OR couleur = 'bleu'
```

5. Récupérer les clients vivant dans certains départements (38,69,75).

```sql
SELECT nom, prenom, geo_id
FROM Clients
WHERE geo_id IN (38, 69, 75);
```

6. Lister les vendeurs qui ont une adresse mail valide (non vide).

```sql
SELECT nom, prenom
FROM Vendeurs
WHERE mail IS NOT NULL
```

7. Récupérer la liste des produits dont le nom contient “VTC”.

```sql
SELECT produit_id, nom, couleur, poids
FROM Produits
WHERE nom LIKE '%VTC%'
```

8. Récupérer les clients dont l’email finit par “@gmail.com”

```sql
SELECT nom, prenom, mail
FROM Clients
WHERE mail LIKE '%@gmail.com'
```

9. Lister les différentes couleurs de produit sans doublon.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT DISTINCT couleur
FROM Produits
```

10. Afficher les produits dont le prix de vente est entre 1000 et 5000 (inclus).

```sql
SELECT nom, prix_vente_unitaire
FROM Produits
WHERE prix_vente_unitaire BETWEEN 1000 AND 5000;
```

11. Afficher les produits qui ne sont pas dans les sous-catégories 1, 2 ou 3.

```sql
SELECT nom, sous_categorie_id
FROM Produits
WHERE sous_categorie_id NOT IN (1, 2, 3);
```

12. Récupérer les ventes réalisées entre le 01/05/2025 et 30/06/2025.

```sql
SELECT vente_id, date_de_vente
FROM Base_N
WHERE date_de_vente BETWEEN '2025-05-01' AND '2025-06-30';
```

13. Lister les clients nés entre 1990 et 2000 et de sexe féminin.

```sql
SELECT nom, prenom, date_naissance
FROM Clients
WHERE date_naissance BETWEEN '1990-01-01' AND '2000-12-31'
  AND sexe = 'femme';
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

3. Ordonnancement et Agrégations

3.1 Fonctions d'agrégation

Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :

- SUM() pour calculer la somme de plusieurs lignes. Elle permet par exemple de connaître le total de tous les achats d'un client.
- COUNT() pour compter le nombre de lignes concernées. Elle permet de savoir le nombre d'achats effectués par chaque client.
- AVG() pour calculer la moyenne d'un set de valeur. Elle permet de connaître le prix du panier moyen pour de chaque client.
- MAX() pour récupérer la plus haute valeur. Pratique pour savoir l'achat le plus cher.
- MIN() pour récupérer la plus petite valeur. Utile par exemple pour connaître la date du premier achat d'un client.

3.1.1 SUM()

La syntaxe de base est la suivante :

```
SELECT SUM(nom_de_colonne)
FROM nom_de_table
```

On peut par exemple calculer la quantité totale de produits achetés dans *Base_N* :

```
SELECT SUM(quantite)
FROM Base_N
```

sum
364604

3.1.2 COUNT()

La syntaxe de base est la suivante :

```
SELECT COUNT(nom_de_colonne)
FROM nom_de_table
```

On peut par exemple calculer le nombre de produits "rouge" dans la table *Produits* :

```
SELECT COUNT(produit_id)
FROM Produits
WHERE couleur = 'rouge'
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

count
4

3.1.3 AVG()

La syntaxe de base est la suivante :

```
SELECT AVG(nom_de_colonne)
FROM nom_de_table
```

On peut par exemple calculer le prix de vente unitaire moyen dans la table *Produits* :

```
SELECT AVG(prix_vente_unitaire)
FROM Produits
```

avg
899.4675

3.1.4 MAX()

La syntaxe de base est la suivante :

```
SELECT MAX(nom_de_colonne)
FROM nom_de_table
```

On peut par exemple calculer le prix de vente unitaire maximal dans la table *Produits* :

```
SELECT MAX(prix_vente_unitaire)
FROM Produits
```

max
7800

3.1.5 MIN()

La syntaxe de base est la suivante :

```
SELECT MIN(nom_de_colonne)
FROM nom_de_table
```

On peut par exemple calculer le prix de vente unitaire maximal dans la table *Produits* :

```
SELECT MIN(prix_vente_unitaire)
FROM Produits
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

min
6

3.2 GROUP BY

La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction d'agrégation sur un groupe de résultat. Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de regrouper les quantités vendues par vendeurs identiques et d'obtenir les quantités totales vendues pour chaque vendeur.

La syntaxe de base est la suivante :

```
SELECT nom_de_colonne1, fonction(nom_de_colonne2)
FROM nom_de_table
GROUP BY nom_de_colonne1
```

Si on reprend l'introduction, on peut donc afficher les quantités totales vendues par vendeur_id dans *Base_N* :

```
SELECT vendeur_id, SUM(quantite)
FROM Base_N
GROUP BY vendeur_id
```

vendeur_id	sum
6	59091
1	62389
3	61506
5	60922
4	61610
2	59086

Si on fait la somme, on retrouve la quantité totale calculée plus haut.

3.3 HAVING

La condition HAVING en SQL permet de filtrer suite à une agrégation. WHERE filtre uniquement des résultats présents dans la table tandis que HAVING permet de filtrer des résultats suite à un calcul d'agrégations.

La syntaxe de base est la suivante :

```
SELECT nom_de_colonne1, fonction(nom_de_colonne2)
FROM nom_de_table
GROUP BY nom_de_colonne1
HAVING fonction(nom_de_colonne2) opérateur valeur
```

Si dans le dernier exemple, on voulait garder seulement les vendeur_id qui ont vendu une quantité supérieure à 60 000 unités :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT vendeur_id, SUM(quantite)
FROM Base_N
GROUP BY vendeur_id
HAVING SUM(quantite) > 60000
```

vendeur_id	sum
1	62389
3	61506
5	60922
4	61610

3.4 AS

Dans certains exemples qui précèdent, on remarque que le nom de colonne est illisible, notamment quand on utilise une fonction. Il est heureusement possible de renommer une colonne dans une requête.

La syntaxe de base est la suivante :

```sql
SELECT nom_de_colonne1 as c1, nom_de_colonne2 as c2
FROM nom_de_table
```

Si on reprend l'exemple avec les quantités vendues par vendeur_id :

```sql
SELECT vendeur_id, SUM(quantite) AS Quantité_totale_vendue
FROM Base_N
GROUP BY vendeur_id
```

vendeur_id	quantité_totale_vendue
6	59091
1	62389
3	61506
5	60922
4	61610
2	59086

3.5 ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

La syntaxe de base est la suivante :

```sql
SELECT nom_de_colonne1, nom_de_colonne2
FROM nom_de_table
ORDER BY nom_de_colonne1
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus complexe ressemblerait alors cela :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
SELECT nom_de_colonne1, nom_de_colonne2
FROM nom_de_table
ORDER BY nom_de_colonne1 DESC, nom_de_colonne2 ASC
```

On peut par exemple afficher la liste des produits dans l'ordre croissant des numéros de sous_catégorie et l'ordre décroissant selon le prix de vente unitaire :

```
SELECT nom, sous_categorie_id, prix_vente_unitaire
FROM Produits
ORDER BY sous_categorie_id ASC, prix_vente_unitaire DESC
```

nom	sous_categorie_id	prix_vente_unitaire
E_bike1001	1	7800
E_bike1002	1	7700
E_bike1000	1	7500
Road_bike1003	2	3900
Road_bike1004	2	3900
Road_bike1001	2	3900
Road_bike1002	2	3900
Road_bike1000	2	3700
VTT1002	3	2700
VTT1001	3	2700
VTT1003	3	2700
VTT1004	3	2700
VTT1005	3	2700
VTT1000	3	2650
VTC1001	4	990
VTC1002	4	990
VTC1000	4	950
Batterie1001	5	990
Batterie1000	5	960
Cables_Ebike	6	406
Cables_Road_bike	6	90
Cables_VTT	6	85
Cables_VTC	6	75
Freins_Ebike	7	138
Freins_Road_bike	7	22
Freins_VTT	7	10
Freins_VTC	7	10
Selle_Ebike	8	150
Selle_Road_bike	8	90
Selle_VTT	8	60
Selle_VTC	8	60
Combinaison	9	108
Bike_shoes	10	85
Oesics_run	11	190
Oesics_run	11	190
Oesics_run	11	190
Oesics_run	11	190
Mike_shoes	11	185
Mike_shoes	11	185
Mike_shoes	11	185

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	sous_categorie_id	prix_vente_unitaire
Mike_shoes	11	185
Abidas_shoes	11	155
Abidas_shoes	11	155
Abidas_shoes	11	155
Abidas_shoes	11	155
Oesics_run_socks	12	6
Oesics_run_socks	12	6
Abidas_run_socks	12	6
Abidas_run_socks	12	6
Mike_run_socks	12	6
Mike_run_socks	12	6
Semelle_running	13	25
Tshirt	14	15
Tshirt	14	15
Tshirt	14	15
Tshirt	14	15
Tshirt	14	15
Tshirt	14	15
Tshirt	14	15
Tshirt	14	15
Short	15	19
Short	15	19
Short	15	19
Short	15	19
SV_Homme	16	25
SV_Femme	16	23
Maillot_de_bain_Homme	17	24
Maillot_de_bain_Homme	17	24
Maillot_de_bain_Femme	17	24
Maillot_de_bain_Femme	17	24
Bonnet_de_bain	18	12
Bonnet_de_bain	18	12
JE_watch	19	399
JE_watch	19	399
JE_watch	19	399
JE_watch	19	399
JE_watch	19	399

3.6 LIMIT

La clause LIMIT permet de limiter le nombre de lignes retournées par une requête.

La syntaxe de base est la suivante :

```
SELECT colonnes
FROM table
ORDER BY ...
LIMIT n
```

On peut par exemple afficher les 5 produits les plus chers :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT nom, sous_categorie_id, prix_vente_unitaire
FROM Produits
ORDER BY prix_vente_unitaire DESC
LIMIT 5
```

nom	sous_categorie_id	prix_vente_unitaire
E_bike1001	1	7800
E_bike1002	1	7700
E_bike1000	1	7500
Road_bike1001	2	3900
Road_bike1002	2	3900

3.7 OFFSET / FETCH

OFFSET et FETCH permettent de :

- Ignorer un certain nombre de lignes (OFFSET)
- Récupérer un nombre précis de lignes (FETCH)

Attention : OFFSET / FETCH ne fonctionnent qu'avec ORDER BY.

Sans ORDER BY, la requête est invalide.

La syntaxe de base est la suivante :

```sql
SELECT colonnes
FROM table
ORDER BY colonne
OFFSET n ROWS
FETCH NEXT m ROWS ONLY;
```

- n = nombre de lignes à ignorer
- m = nombre de lignes à récupérer

Par exemple, on peut afficher les 10 premiers produits (équivalent de TOP)

```sql
SELECT produit_id, nom
FROM Produits
ORDER BY produit_id
OFFSET 0 ROWS
FETCH NEXT 10 ROWS ONLY;
```

produit_id	nom
1	E_bike1000
2	E_bike1001
3	E_bike1002
4	Road_bike1000
5	Road_bike1001

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id	nom
6	Road_bike1002
7	Road_bike1003
8	Road_bike1004
9	VTT1000
10	VTT1001

On peut aussi afficher les 10 produits les plus chers à l'exception des 3 premiers (rangs 4 à 16) :

```
SELECT produit_id, nom, prix_vente_unitaire
FROM Produits
ORDER BY prix_vente_unitaire DESC
OFFSET 3 ROWS
FETCH NEXT 10 ROWS ONLY;
```

produit_id	nom	prix_vente_unitaire
7	Road_bike1003	3900
6	Road_bike1002	3900
5	Road_bike1001	3900
8	Road_bike1004	3900
4	Road_bike1000	3700
13	VTT1004	2700
12	VTT1003	2700
11	VTT1002	2700
10	VTT1001	2700
14	VTT1005	2700

OFFSET / FETCH appartiennent au standard SQL contrairement à TOP.

3.8 PARTITION BY()

Cette commande permet de créer des groupes logiques (partitions), comme ferait un GROUP BY… Mais sans regrouper les lignes : **on garde le détail.**

C'est donc une alternative à GROUP BY lorsqu'on veut agréger + conserver le détail.

La syntaxe de base est la suivante :

```
SELECT colonnes,
Fonction() OVER (PARTITION BY colonne)
FROM table
```

Si on veut la liste de nos produits tout en indiquant le prix moyen par sous_categorie, la requête serait la suivante :

```
SELECT nom, couleur, sous_categorie_id, prix_vente_unitaire,
AVG(prix_vente_unitaire) OVER (PARTITION BY sous_categorie_id) AS prix_moyen_par_sous_categorie
FROM Produits
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	couleur	sous_categorie_id	prix_vente_unitaire	prix_moyen_par_sous_categorie
E_bike1000	noir	1	7500	7666.6667
E_bike1001	bleu	1	7800	7666.6667
E_bike1002	rouge	1	7700	7666.6667
Road_bike1000	noir	2	3700	3860.0000
Road_bike1001	bleu	2	3900	3860.0000
Road_bike1002	rouge	2	3900	3860.0000
Road_bike1003	vert	2	3900	3860.0000
Road_bike1004	jaune	2	3900	3860.0000
VTT1000	noir	3	2650	2691.6667
VTT1001	bleu	3	2700	2691.6667
VTT1002	rouge	3	2700	2691.6667
VTT1003	vert	3	2700	2691.6667
VTT1004	jaune	3	2700	2691.6667
VTT1005	rose	3	2700	2691.6667
VTC1000	noir	4	950	976.6667
VTC1001	bleu	4	990	976.6667
VTC1002	rose	4	990	976.6667
Batterie1000	noir	5	960	975.0000
Batterie1001	noir	5	990	975.0000
Cables_Ebike	noir	6	406	164.0000
Cables_Road_bike	noir	6	90	164.0000
Cables_VTT	noir	6	85	164.0000
Cables_VTC	noir	6	75	164.0000
Freins_Ebike	noir	7	138	45.0000
Freins_Road_bike	noir	7	22	45.0000
Freins_VTT	noir	7	10	45.0000
Freins_VTC	noir	7	10	45.0000
Selle_Ebike	noir	8	150	90.0000
Selle_Road_bike	noir	8	90	90.0000
Selle_VTT	noir	8	60	90.0000
Selle_VTC	noir	8	60	90.0000
Combinaison	noir	9	108	108.0000
Bike_shoes	noir	10	85	85.0000
Oesics_run	noir	11	190	176.6667
Oesics_run	blanc	11	190	176.6667
Oesics_run	bleu	11	190	176.6667
Oesics_run	orange	11	190	176.6667
Abidas_shoes	noir	11	155	176.6667
Abidas_shoes	blanc	11	155	176.6667
Abidas_shoes	bleu	11	155	176.6667
Abidas_shoes	orange	11	155	176.6667
Mike_shoes	noir	11	185	176.6667
Mike_shoes	blanc	11	185	176.6667
Mike_shoes	bleu	11	185	176.6667
Mike_shoes	orange	11	185	176.6667
Oesics_run_socks	noir	12	6	6.0000
Oesics_run_socks	blanc	12	6	6.0000
Abidas_run_socks	noir	12	6	6.0000
Abidas_run_socks	blanc	12	6	6.0000
Mike_run_socks	noir	12	6	6.0000
Mike_run_socks	blanc	12	6	6.0000
Semelle_running	noir	13	25	25.0000

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	couleur	sous_categorie_id	prix_vente_unitaire	prix_moyen_par_sous_categorie
Tshirt	noir	14	15	15.0000
Tshirt	blanc	14	15	15.0000
Tshirt	bleu	14	15	15.0000
Tshirt	jaune	14	15	15.0000
Tshirt	rose	14	15	15.0000
Tshirt	orange	14	15	15.0000
Tshirt	rouge	14	15	15.0000
Tshirt	vert	14	15	15.0000
Short	noir	15	19	19.0000
Short	blanc	15	19	19.0000
Short	rose	15	19	19.0000
Short	orange	15	19	19.0000
SV_Homme	noir	16	25	24.0000
SV_Femme	noir	16	23	24.0000
Maillot_de_bain_Homme	noir	17	24	24.0000
Maillot_de_bain_Homme	blanc	17	24	24.0000
Maillot_de_bain_Femme	noir	17	24	24.0000
Maillot_de_bain_Femme	blanc	17	24	24.0000
Bonnet_de_bain	noir	18	12	12.0000
Bonnet_de_bain	blanc	18	12	12.0000
JE_watch	noir	19	399	399.0000
JE_watch	blanc	19	399	399.0000
JE_watch	vert	19	399	399.0000
JE_watch	rose	19	399	399.0000
JE_watch	gris	19	399	399.0000

Voici une petite comparaison entre **GROUP BY** et **PARTITION BY** à garder en tête :

Objectif	GROUP BY	PARTITION BY
Agréger	OUI	OUI
Conserver le détail	NON	OUI
Montrer tous les résultats de groupe	OUI	OUI
Montrer chaque ligne originale	NON	OUI

3.9 ROW_NUMBER()

ROW_NUMBER() assigne un numéro unique à chaque ligne dans une partition (ou sur l'ensemble), selon un ordre.

La syntaxe de base est la suivante :

```
SELECT colonnes,
ROW_NUMBER() OVER (ORDER BY colonne) as rang
FROM table
```

On peut par exemple numéroter les produits du plus cher au moins cher en leur affectant un rang :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
SELECT nom, couleur, poids, prix_vente_unitaire,
ROW_NUMBER() OVER (ORDER BY prix_vente_unitaire DESC) AS RangProduit
FROM Produits
```

nom	couleur	poids	prix_vente_unitaire	rangproduit
E_bike1001	bleu	25	7800	1
E_bike1002	rouge	25	7700	2
E_bike1000	noir	28	7500	3
Road_bike1001	bleu	8	3900	4
Road_bike1002	rouge	8	3900	5
Road_bike1003	vert	8	3900	6
Road_bike1004	jaune	8	3900	7
Road_bike1000	noir	8	3700	8
VTT1005	rose	14	2700	9
VTT1001	bleu	14	2700	10
VTT1002	rouge	14	2700	11
VTT1003	vert	14	2700	12
VTT1004	jaune	14	2700	13
VTT1000	noir	14	2650	14
VTC1002	rose	11	990	15
VTC1001	bleu	11	990	16
Batterie1001	noir	2	990	17
Batterie1000	noir	2	960	18
VTC1000	noir	11	950	19
Cables_Ebike	noir	1	406	20
JE_watch	blanc	0	399	21
JE_watch	noir	0	399	22
JE_watch	gris	0	399	23
JE_watch	rose	0	399	24
JE_watch	vert	0	399	25
Oesics_run	noir	0	190	26
Oesics_run	bleu	0	190	27
Oesics_run	blanc	0	190	28
Oesics_run	orange	0	190	29
Mike_shoes	orange	0	185	30
Mike_shoes	noir	0	185	31
Mike_shoes	blanc	0	185	32
Mike_shoes	bleu	0	185	33
Abidas_shoes	noir	0	155	34
Abidas_shoes	orange	0	155	35
Abidas_shoes	bleu	0	155	36
Abidas_shoes	blanc	0	155	37
Selle_Ebike	noir	1	150	38
Freins_Ebike	noir	1	138	39
Combinaison	noir	1	108	40
Cables_Road_bike	noir	1	90	41
Selle_Road_bike	noir	1	90	42
Bike_shoes	noir	0	85	43
Cables_VTT	noir	1	85	44
Cables_VTC	noir	1	75	45
Selle_VTT	noir	1	60	46
Selle_VTC	noir	1	60	47

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	couleur	poids	prix_vente_unitaire	rangproduit
SV_Homme	noir	0	25	48
Semelle_running	noir	0	25	49
Maillot_de_bain_Femme	blanc	0	24	50
Maillot_de_bain_Femme	noir	0	24	51
Maillot_de_bain_Homme	blanc	0	24	52
Maillot_de_bain_Homme	noir	0	24	53
SV_Femme	noir	0	23	54
Freins_Road_bike	noir	1	22	55
Short	rose	0	19	56
Short	noir	0	19	57
Short	blanc	0	19	58
Short	orange	0	19	59
Tshirt	vert	0	15	60
Tshirt	jaune	0	15	61
Tshirt	bleu	0	15	62
Tshirt	blanc	0	15	63
Tshirt	noir	0	15	64
Tshirt	rose	0	15	65
Tshirt	orange	0	15	66
Tshirt	rouge	0	15	67
Bonnet_de_bain	noir	0	12	68
Bonnet_de_bain	blanc	0	12	69
Freins_VTT	noir	1	10	70
Freins_VTC	noir	1	10	71
Abidas_run_socks	blanc	0	6	72
Oesics_run_socks	noir	0	6	73
Oesics_run_socks	blanc	0	6	74
Mike_run_socks	blanc	0	6	75
Mike_run_socks	noir	0	6	76
Abidas_run_socks	noir	0	6	77

Chaque rang est unique, même pour les produits ayant le même prix de vente.

3.10 RANK()

RANK() attribue un rang avec ex æquo et saute le rang suivant si égalité.

La syntaxe de base est la suivante :

```
SELECT colonnes,
RANK() OVER (ORDER BY colonne) as rang
FROM table
```

On peut par exemple numéroter les produits du plus cher au moins cher en leur affectant un rang en prenant en compte les égalités :

```
SELECT nom, couleur, poids, prix_vente_unitaire,
RANK() OVER (ORDER BY prix_vente_unitaire DESC) AS RangProduit
FROM Produits
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	couleur	poids	prix_vente_unitaire	rangproduit
E_bike1001	bleu	25	7800	1
E_bike1002	rouge	25	7700	2
E_bike1000	noir	28	7500	3
Road_bike1001	bleu	8	3900	4
Road_bike1002	rouge	8	3900	4
Road_bike1003	vert	8	3900	4
Road_bike1004	jaune	8	3900	4
Road_bike1000	noir	8	3700	8
VTT1005	rose	14	2700	9
VTT1001	bleu	14	2700	9
VTT1002	rouge	14	2700	9
VTT1003	vert	14	2700	9
VTT1004	jaune	14	2700	9
VTT1000	noir	14	2650	14
VTC1002	rose	11	990	15
VTC1001	bleu	11	990	15
Batterie1001	noir	2	990	15
Batterie1000	noir	2	960	18
VTC1000	noir	11	950	19
Cables_Ebike	noir	1	406	20
JE_watch	blanc	0	399	21
JE_watch	noir	0	399	21
JE_watch	gris	0	399	21
JE_watch	rose	0	399	21
JE_watch	vert	0	399	21
Oesics_run	noir	0	190	26
Oesics_run	bleu	0	190	26
Oesics_run	blanc	0	190	26
Oesics_run	orange	0	190	26
Mike_shoes	orange	0	185	30
Mike_shoes	noir	0	185	30
Mike_shoes	blanc	0	185	30
Mike_shoes	bleu	0	185	30
Abidas_shoes	noir	0	155	34
Abidas_shoes	orange	0	155	34
Abidas_shoes	bleu	0	155	34
Abidas_shoes	blanc	0	155	34
Selle_Ebike	noir	1	150	38
Freins_Ebike	noir	1	138	39
Combinaison	noir	1	108	40
Cables_Road_bike	noir	1	90	41
Selle_Road_bike	noir	1	90	41
Bike_shoes	noir	0	85	43
Cables_VTT	noir	1	85	43
Cables_VTC	noir	1	75	45
Selle_VTT	noir	1	60	46
Selle_VTC	noir	1	60	46
SV_Homme	noir	0	25	48
Semelle_running	noir	0	25	48
Maillot_de_bain_Femme	blanc	0	24	50
Maillot_de_bain_Femme	noir	0	24	50
Maillot_de_bain_Homme	blanc	0	24	50

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	couleur	poids	prix_vente_unitaire	rangproduit
Maillot_de_bain_Homme	noir	0	24	50
SV_Femme	noir	0	23	54
Freins_Road_bike	noir	1	22	55
Short	rose	0	19	56
Short	noir	0	19	56
Short	blanc	0	19	56
Short	orange	0	19	56
Tshirt	vert	0	15	60
Tshirt	jaune	0	15	60
Tshirt	bleu	0	15	60
Tshirt	blanc	0	15	60
Tshirt	noir	0	15	60
Tshirt	rose	0	15	60
Tshirt	orange	0	15	60
Tshirt	rouge	0	15	60
Bonnet_de_bain	noir	0	12	68
Bonnet_de_bain	blanc	0	12	68
Freins_VTT	noir	1	10	70
Freins_VTC	noir	1	10	70
Abidas_run_socks	blanc	0	6	72
Oesics_run_socks	noir	0	6	72
Oesics_run_socks	blanc	0	6	72
Mike_run_socks	blanc	0	6	72
Mike_run_socks	noir	0	6	72
Abidas_run_socks	noir	0	6	72

Cette fois-ci, les cas d'égalité sont conservés.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

3.11 Exercices

1. Afficher chaque vendeur avec la somme des quantités vendues.

vendeur_id	total_ventes
6	59091
1	62389
3	61506
5	60922
4	61610
2	59086

2. Calculer la moyenne du prix des produits par sous-catégorie.

sous_categorie_id	prix_moyen
11	176.6667
8	90.0000
19	399.0000
4	976.6667
14	15.0000
3	2691.6667
17	24.0000
13	25.0000
10	85.0000
9	108.0000
7	45.0000
1	7666.6667
5	975.0000
18	12.0000
2	3860.0000
16	24.0000
15	19.0000
6	164.0000
12	6.0000

3. Ne garder que les clients de la table *Base_N* ayant effectué plus de 5 achats.

client_id	nb_achats
8228	7
5436	6
12836	6
13763	6
12824	6
9142	6
9624	8
22513	6
3766	6
5596	6

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

client_id	nb_achats
16156	6
10078	8
12888	6
4994	6
8836	6
16366	6
2262	6
4644	6
22542	6
10202	6

(Seuls les premiers résultats sont affichés).

4. Utiliser LIMIT pour afficher les 5 meilleurs produits en quantité totale dans la table *Base_N*.

produit_id	total_vendu
72	25955
68	25163
71	25005
69	24945
67	24583

5. Indiquer les 3 produit_id les plus vendus en quantité dans la table *Base_N*.

produit_id	total_vendu
72	25955
68	25163
71	25005

6. Attribuer un numéro de vente chronologique à chaque client.

client_id	vente_id	date_de_vente	n_vente
1	V2025020154	2025-02-06	1
1	V2025020002	2025-05-03	2
1	TCL106	2026-01-13	3
1	V001	2026-01-13	4
1	V100	2026-01-14	5
1	T100	2026-01-20	6
2	V2025006338	2025-06-09	1
2	V2025045998	2025-06-28	2
2	V2025033058	2025-10-04	3
2	V002	2026-01-13	4
2	TCL107	2026-01-13	5
3	V2025032366	2025-02-12	1
3	V2025008240	2025-08-05	2

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

client_id	vente_id	date_de_vente	n_vente
3	V003	2026-01-13	3
3	TCL108	2026-01-13	4
4	V2025023937	2025-06-08	1
4	V2025023588	2025-08-18	2
4	V10352	2026-01-13	3
4	VPROC002	2026-01-14	4
5	V2025014420	2025-05-02	1
5	V2025029593	2025-09-04	2
5	V2025044908	2025-10-31	3
5	V2025036834	2025-12-07	4
6	V2025034645	2025-02-09	1
6	V2025016517	2025-03-09	2
6	V2025025178	2025-10-24	3
7	V2025035245	2025-01-11	1
7	V2025014383	2025-01-11	2
7	V2025025140	2025-09-01	3
7	V2025005103	2025-11-04	4
7	V2025016476	2025-12-03	5
8	V2025037640	2025-02-03	1
8	V2025010179	2025-06-18	2
8	V2025003882	2025-07-17	3
8	V2025034954	2025-11-04	4
9	V2025036297	2025-01-07	1
9	V2025039563	2025-05-02	2
9	V2025035255	2025-05-22	3
9	V2025026474	2025-07-16	4
9	V2025041259	2025-11-09	5

(Seuls les 40 premiers résultats sont affichés)

7. Indiquer le meilleur vendeur_id de la *Base_N* en termes de quantité totale vendue.

vendeur_id	total_vendu
1	62389

8. Indiquer le meilleur vendeur_id de la *Base_N* en termes de nombre de ventes effectués.

vendeur_id	nb_ventes
1	7903

9. Donner le classement des vendeurs dans la *Base_N* (quantité totale).

vendeur_id	total_ventes	rang
1	62389	1
4	61610	2

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vendeur_id	total_ventes	rang
3	61506	3
5	60922	4
6	59091	5
2	59086	6

10. Donner le produit le moins cher et le plus cher de chaque sous-catégorie.

sous_categorie_id	prix_min	prix_max
11	155	190
8	60	150
19	399	399
4	950	990
14	15	15
3	2650	2700
17	24	24
13	25	25
10	85	85
9	108	108
7	10	138
1	7500	7800
5	960	990
18	12	12
2	3700	3900
16	23	25
15	19	19
6	75	406
12	6	6

11. Moyenne, minimum et maximum des quantités par produit

produit_id	moyenne_qte	qte_min	qte_max
50	12.869245	5	20
49	12.781086	5	20
54	12.735144	5	20
46	12.709619	5	20
51	12.685817	5	20
55	12.678757	5	20
47	12.655518	5	20
66	12.641541	5	20
52	12.640232	5	20
60	12.614679	5	20
56	12.603636	5	20
68	12.587794	5	20
65	12.586854	5	20
59	12.578680	5	20
70	12.567775	5	20
72	12.556846	5	20
64	12.552316	5	20

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id	moyenne_qte	qte_min	qte_max
67	12.529562	5	20
61	12.517647	5	20
62	12.485166	5	20
63	12.451499	5	20
69	12.447605	5	20
71	12.427932	5	20
53	12.384476	5	20
58	12.380256	5	20
48	12.364273	5	20
57	12.240550	5	20
22	3.057391	1	5
39	2.944294	1	5
28	2.938614	1	5
38	2.899868	1	5
26	2.890909	1	5
43	2.890541	1	5
41	2.882506	1	5
34	2.869908	1	5
27	2.863727	1	5
37	2.863459	1	5
3	2.859649	-3	10
35	2.853521	1	5
32	2.851024	1	5
33	2.840220	1	5
44	2.831309	1	5
23	2.830739	1	5
18	2.828974	1	5
20	2.826642	1	5
36	2.824742	1	5
21	2.821306	1	5
42	2.817558	1	5
40	2.797619	1	5
1	2.785714	1	12
29	2.780488	1	5
19	2.774423	1	5
30	2.771318	1	5
45	2.763014	1	5
25	2.758929	1	5
9	2.758065	1	4
31	2.755639	1	5
24	2.738490	1	5
77	2.727742	1	5
75	2.704242	1	5
11	2.689655	1	4
6	2.672414	1	4
10	2.655172	1	4
76	2.653088	1	5
73	2.641026	1	5
12	2.639344	1	4
74	2.620269	1	5
7	2.612245	1	4
2	2.580645	-1	7

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id	moyenne_qte	qte_min	qte_max
16	2.540541	1	4
17	2.539683	1	4
5	2.474576	1	4
15	2.446429	1	4
8	2.385965	1	4
14	2.367347	1	4
4	2.262295	1	4
13	2.100000	1	4

12. Afficher la somme cumulée des ventes par vendeur par date de vente

vendeur_id	vente_id	date_de_vente	quantite	cumul_ventes
1	V2025029833	2025-01-01	16	194
1	V2025034558	2025-01-01	11	194
1	V2025039491	2025-01-01	10	194
1	V2025001588	2025-01-01	2	194
1	V2025044879	2025-01-01	2	194
1	V2025044815	2025-01-01	3	194
1	V2025035190	2025-01-01	8	194
1	V2025026468	2025-01-01	11	194
1	V2025038429	2025-01-01	13	194
1	V2025014222	2025-01-01	1	194
1	V2025005995	2025-01-01	4	194
1	V2025030696	2025-01-01	20	194
1	V2025029723	2025-01-01	7	194
1	V2025003426	2025-01-01	3	194
1	V2025026249	2025-01-01	5	194
1	V2025001807	2025-01-01	4	194
1	V2025044444	2025-01-01	1	194
1	V2025037824	2025-01-01	18	194
1	V2025015605	2025-01-01	4	194
1	V2025018057	2025-01-01	1	194
1	V2025026297	2025-01-01	16	194
1	V2025029669	2025-01-01	9	194
1	V2025006774	2025-01-01	4	194
1	V2025026278	2025-01-01	18	194
1	V2025007040	2025-01-01	3	194
1	V2025025174	2025-01-02	8	348
1	V2025024004	2025-01-02	12	348
1	V2025004184	2025-01-02	4	348
1	V2025029240	2025-01-02	5	348
1	V2025024407	2025-01-02	5	348
1	V2025023968	2025-01-02	15	348
1	V2025005208	2025-01-02	5	348
1	V2025045261	2025-01-02	1	348
1	V2025042036	2025-01-02	18	348
1	V2025000198	2025-01-02	3	348
1	V2025027805	2025-01-02	10	348
1	V2025013722	2025-01-02	5	348
1	V2025031070	2025-01-02	13	348

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vendeur_id	vente_id	date_de_vente	quantite	cumul_ventes
1	V2025030257	2025-01-02	19	348
1	V2025033680	2025-01-02	19	348
1	V2025020318	2025-01-02	12	348
1	V2025008303	2025-01-03	1	537
1	V2025036913	2025-01-03	14	537
1	V2025041477	2025-01-03	15	537
1	V2025037972	2025-01-03	10	537
1	V2025019168	2025-01-03	10	537
1	V2025033877	2025-01-03	9	537
1	V2025001946	2025-01-03	1	537
1	V2025003249	2025-01-03	2	537
1	V2025029274	2025-01-03	12	537

(Seuls les premiers résultats sont affichés).

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

3.12 Solutions

1. Afficher chaque vendeur avec la somme des quantités vendues.

```sql
SELECT vendeur_id, SUM(quantite) AS total_ventes
FROM Base_N
GROUP BY vendeur_id;
```

2. Calculer la moyenne du prix des produits par sous-catégorie.

```sql
SELECT sous_categorie_id, AVG(prix_vente_unitaire) AS prix_moyen
FROM Produits
GROUP BY sous_categorie_id;
```

3. Ne garder que les clients de la table *Base_N* ayant effectué plus de 5 achats.

```sql
SELECT client_id, COUNT(vente_id) AS nb_achats
FROM Base_N
GROUP BY client_id
HAVING COUNT(vente_id) > 5;
```

4. Utiliser LIMIT pour afficher les 5 meilleurs produits en quantité totale dans la table *Base_N*.

```sql
SELECT produit_id, SUM(quantite) AS total_vendu
FROM Base_N
GROUP BY produit_id
ORDER BY SUM(quantite) DESC
LIMIT 5;
```

5. Indiquer les 3 produit_id les plus vendus en quantité dans la table *Base_N*.

```sql
SELECT produit_id, SUM(quantite) AS total_vendu
FROM Base_N
GROUP BY produit_id
ORDER BY SUM(quantite) DESC
LIMIT 3;
```

6. Attribuer un numéro de vente chronologique à chaque client.

```sql
SELECT client_id,
    vente_id,
    date_de_vente,
    ROW_NUMBER() OVER (PARTITION BY client_id ORDER BY date_de_vente) AS n_vente
FROM Base_N;
```

7. Indiquer le meilleur vendeur_id de la *Base_N* en termes de quantité totale vendue.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT vendeur_id, SUM(quantite) AS total_vendu
FROM Base_N
GROUP BY vendeur_id
ORDER BY SUM(quantite) DESC
LIMIT 1;
```

8. Indiquer le meilleur vendeur_id de la *Base_N* en termes de nombre de ventes effectués.

```sql
SELECT vendeur_id, COUNT(vente_id) AS nb_ventes
FROM Base_N
GROUP BY vendeur_id
ORDER BY COUNT(vente_id) DESC
LIMIT 1;
```

9. Donner le classement des vendeurs dans la *Base_N* (quantité totale).

```sql
SELECT
    vendeur_id,
    SUM(quantite) AS total_ventes,
    RANK() OVER (ORDER BY SUM(quantite) DESC) AS rang
FROM Base_N
GROUP BY vendeur_id;
```

10. Donner le produit le moins cher et le plus cher de chaque sous-catégorie.

```sql
SELECT
    sous_categorie_id,
    MIN(prix_vente_unitaire) AS prix_min,
    MAX(prix_vente_unitaire) AS prix_max
FROM Produits
GROUP BY sous_categorie_id;
```

11. Moyenne, minimum et maximum des quantités par produit

```sql
SELECT
    produit_id,
    AVG(quantite) AS moyenne_qte,
    MIN(quantite) AS qte_min,
    MAX(quantite) AS qte_max
FROM Base_N
GROUP BY produit_id
ORDER BY moyenne_qte DESC;
```

12. Afficher la somme cumulée des ventes par vendeur

```sql
SELECT
    vendeur_id,
    vente_id,
    date_de_vente,
    quantite,
    SUM(quantite) OVER (PARTITION BY vendeur_id order by date_de_vente) AS cumul_ventes
FROM Base_N
LIMIT 50;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

4. Présenter les données

4.1 Opérations sur les chaines de caractère (varchar/nvarchar)

4.1.1 LTRIM()

La fonction supprime les espaces à gauche.

La syntaxe de base est la suivante :

```
SELECT LTRIM(nom) AS nom_sans_espaces_gauche
FROM Clients;
```

4.1.2 RTRIM()

La fonction supprime les espaces à droite.

La syntaxe de base est la suivante :

```
SELECT RTRIM(nom) AS nom_sans_espaces_droite
FROM Clients;
```

4.1.3 TRIM()

La fonction supprime les espaces à gauche et à droite.

La syntaxe de base est la suivante :

```
SELECT TRIM(nom) AS nom_sans_espaces
FROM Clients;
```

Toujours nettoyer les colonnes avant de faire des comparaisons, des concaténations ou des conversions.

4.1.4 LENGTH()

Utile pour détecter les espaces cachés ou vérifier la taille maximale d'une colonne.

La syntaxe de base est la suivante :

```
SELECT nom, LENGTH(nom) AS longueur_nom
FROM Clients;
```

4.1.5 SUBSTRING()

La fonction permet d'extraire une sous-chaîne.

La syntaxe de base est la suivante :

```
SELECT prenom, SUBSTRING(prenom,1,3) AS debut_prenom
FROM Clients;
```

L'exemple ci-dessus permet d'extraire à partir du premier caractère : **1**, les **3** premiers caractères.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

4.1.6 CONCAT()

Cette fonction permet de concaténer des chaines de caractère.

La syntaxe de base est la suivante :

```
SELECT CONCAT(TRIM(prenom), ' ', TRIM(nom)) AS nom_complet
FROM Clients;
```

On peut également utiliser l'opérateur + de la manière suivante :

```
SELECT TRIM(prenom) + ' ' + TRIM(nom) AS nom_complet
FROM Clients;
```

4.1.7 REPLACE()

La fonction permet de remplacer du texte.

La syntaxe de base est la suivante :

```
-- Remplacer un mot dans une colonne texte
SELECT REPLACE(nom, 'Durand', 'Dupont') AS nom_modifie
FROM Clients;
```

On peut aussi s'en servir pour remplacer des valeurs d'une colonne :

```
-- Corriger les virgules par des points dans une colonne importée
UPDATE Base_N_1
SET discount = REPLACE(discount, ',', '.');
```

4.1.8 UPPER()/LOWER()

Ces fonctions permettent respectivement en majuscules/minuscules.

La syntaxe de base est la suivante :

```
SELECT UPPER(nom) AS nom_maj, LOWER(prenom) AS prenom_min
FROM Clients;
```

4.2 Opérations sur les dates

4.2.1 CURRENT_DATE

La commande retourne la date et l'heure actuelles.

La syntaxe de base est la suivante :

```
SELECT CURRENT_DATE AS date_maintenant;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

4.2.2 EXTRACT(YEAR ou MONTH ou DAY FROM ...)

Cette fonction permettent d'obtenir selon l'argument à l'intérieur, respectivement l'année, le mois ou le jour.

La syntaxe de base est la suivante :

```
SELECT
    date_de_vente,
    EXTRACT(YEAR FROM date_de_vente) AS annee,
    EXTRACT(MONTH FROM date_de_vente) AS mois,
    EXTRACT(DAY FROM date_de_vente) AS jour
FROM Base_N;
```

4.2.3 DATEADD()

La fonction permet d'ajouter des jours, mois ou années à une date.

La syntaxe de base est la suivante :

```
-- Ajouter 30 jours aux ventes
SELECT DATEADD(day, 30, date_de_vente) AS vente_plus_30
FROM Base_N;

-- Ajouter 1 an à la date de naissance
SELECT DATEADD(year, 1, date_naissance) AS anniversaire_plus_1
FROM Clients;
```

4.2.4 AGE()

La fonction permet de calculer la différence entre dates. Elle retourne un intervalle de temps (par exemple 25 ans 3 mois et 12 jours).

Il faut l'utiliser avec EXTRACT(...) pour calculer un âge en années par exemple.

La syntaxe de base est la suivante :

```
-- Nombre de jours entre une vente et aujourd'hui
SELECT vente_id, EXTRACT(DAY FROM AGE(CURRENT_DATE, date_de_vente) AS jours_ecoules
FROM Base_N;

-- Âge du client en années
SELECT nom, prenom, EXTRACT(YEAR FROM AGE(CURRENT_DATE, date_de_naissance) AS age
FROM Clients;
```

4.2.5 FORMAT()

Elle permet de personnaliser le format en précisant le format souhaité.

La syntaxe de base est la suivante :

```
SELECT FORMAT(date_de_vente, 'dd/MM/yyyy') AS date_formatee
FROM Base_N;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
---Pour obtenir les résultats en lettre (janvier, fevrier, lun, mar...)
SELECT
    date_de_vente,
    FORMAT(date_de_vente, 'MMMM') AS mois,
    FORMAT(date_de_vente, 'ddd') AS jour_semaine
FROM Base_N;
```

4.3 Conversion de données

4.3.1 CAST()

La fonction permet de convertir les types de données.

La syntaxe de base est la suivante :

```sql
SELECT CAST(quantite AS decimal(5,2)) AS qte_decimal
FROM Base_N;
```

4.3.2 TRY_CAST()/TRY_CONVERT()

Ces fonctions permettent de ne pas renvoyer d'erreur mais NULL au cas où la conversion se passe mal.

La syntaxe de base est la suivante :

```sql
SELECT TRY_CAST(discount AS decimal(5,4)) AS discount_num
FROM Base_N_1;
```

4.3.3 Exemples de conversion de données

Par exemple, on peut convertir la colonne discount de la table *Base_N_1* en un nombre décimal de 4 chiffres après la virgule :

```sql
ALTER TABLE Base_N_1
ALTER COLUMN discount decimal(5,4);
```

Ou encore afficher la colonne date_de_vente en format date français :

```sql
UPDATE Base_N
SET date_de_vente = CONVERT(date, date_de_vente, 103);
```

4.4 CASE

CASE permet de créer des conditions dans une requête SQL, comme un IF dans d'autres langages de programmation.

- Il permet de catégoriser
- Transformer des valeurs
- Créer des colonnes calculées

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

- Gérer des exceptions

La syntaxe de base est la suivante :

```
CASE
    WHEN condition THEN résultat
    WHEN condition THEN résultat
    ELSE résultat_par_défault
END
```

Quelques exemples afin de mieux appréhender la syntaxe :

- Catégoriser les produits selon le prix

```
SELECT
    produit_id,
    prix_vente_unitaire,
    CASE
        WHEN prix_vente_unitaire < 10 THEN 'Bon marché'
        WHEN prix_vente_unitaire BETWEEN 10 AND 30 THEN 'Moyen'
        ELSE 'Cher'
    END AS categorie_prix
FROM Produits;
```

produit_id	prix_vente_unitaire	categorie_prix
1	7500	Cher
2	7800	Cher
3	7700	Cher
4	3700	Cher
5	3900	Cher
6	3900	Cher
7	3900	Cher
8	3900	Cher
9	2650	Cher
10	2700	Cher
11	2700	Cher
12	2700	Cher
13	2700	Cher
14	2700	Cher
15	950	Cher
16	990	Cher
17	990	Cher
18	960	Cher
19	990	Cher
20	406	Cher
21	90	Cher
22	85	Cher
23	75	Cher
24	138	Cher
25	22	Moyen
26	10	Moyen

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id	prix_vente_unitaire	categorie_prix
27	10	Moyen
28	150	Cher
29	90	Cher
30	60	Cher
31	60	Cher
32	108	Cher
33	85	Cher
34	190	Cher
35	190	Cher
36	190	Cher
37	190	Cher
38	155	Cher
39	155	Cher
40	155	Cher
41	155	Cher
42	185	Cher
43	185	Cher
44	185	Cher
45	185	Cher
46	6	Bon marché
47	6	Bon marché
48	6	Bon marché
49	6	Bon marché
50	6	Bon marché
51	6	Bon marché
52	25	Moyen
53	15	Moyen
54	15	Moyen
55	15	Moyen
56	15	Moyen
57	15	Moyen
58	15	Moyen
59	15	Moyen
60	15	Moyen
61	19	Moyen
62	19	Moyen
63	19	Moyen
64	19	Moyen
65	25	Moyen
66	23	Moyen
67	24	Moyen
68	24	Moyen
69	24	Moyen
70	24	Moyen
71	12	Moyen
72	12	Moyen
73	399	Cher
74	399	Cher
75	399	Cher
76	399	Cher
77	399	Cher

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

- Indiquer si un vendeur a un objectif élevé :

```
SELECT
    vendeur_id,
    vend_objectif,
    CASE
        WHEN vend_objectif > 4000000 THEN 'Objectif élevé'
        ELSE 'Objectif normal'
    END AS analyse_objectif
FROM Vendeurs;
```

vendeur_id	vend_objectif	analyse_objectif
1	3800000	Objectif normal
2	4200000	Objectif élevé
3	4030000	Objectif élevé
4	4050000	Objectif élevé
5	3900000	Objectif normal
6	3800000	Objectif normal

- Produits regroupés par catégorie de prix :

```
SELECT
    CASE
        WHEN prix_vente_unitaire < 10 THEN 'Bon marché'
        WHEN prix_vente_unitaire BETWEEN 10 AND 30 THEN 'Moyen'
        ELSE 'Cher'
    END AS categorie_prix,
    COUNT(*) AS nb_produits
FROM Produits
GROUP BY
    CASE
        WHEN prix_vente_unitaire < 10 THEN 'Bon marché'
        WHEN prix_vente_unitaire BETWEEN 10 AND 30 THEN 'Moyen'
        ELSE 'Cher'
    END;
```

categorie_prix	nb_produits
Bon marché	6
Moyen	24
Cher	47

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

4.5 Exercices

1. Afficher le nom du produit en MAJUSCULE.

produit_en_majuscule
E_BIKE1000
E_BIKE1001
E_BIKE1002
ROAD_BIKE1000
ROAD_BIKE1001
ROAD_BIKE1002
ROAD_BIKE1003
ROAD_BIKE1004
VTT1000
VTT1001
VTT1002
VTT1003
VTT1004
VTT1005
VTC1000
VTC1001
VTC1002
BATTERIE1000
BATTERIE1001
CABLES_EBIKE
CABLES_ROAD_BIKE
CABLES_VTT
CABLES_VTC
FREINS_EBIKE
FREINS_ROAD_BIKE
FREINS_VTT
FREINS_VTC
SELLE_EBIKE
SELLE_ROAD_BIKE
SELLE_VTT
SELLE_VTC
COMBINAISON
BIKE_SHOES
OESICS_RUN
OESICS_RUN
OESICS_RUN
OESICS_RUN
ABIDAS_SHOES
ABIDAS_SHOES
ABIDAS_SHOES
ABIDAS_SHOES
MIKE_SHOES
MIKE_SHOES
MIKE_SHOES
MIKE_SHOES
OESICS_RUN_SOCKS
OESICS_RUN_SOCKS

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_en_majuscule
ABIDAS_RUN_SOCKS
ABIDAS_RUN_SOCKS
MIKE_RUN_SOCKS
MIKE_RUN_SOCKS
SEMELLE_RUNNING
TSHIRT
TSHIRT
TSHIRT
TSHIRT
TSHIRT
TSHIRT
TSHIRT
TSHIRT
SHORT
SHORT
SHORT
SHORT
SV_HOMME
SV_FEMME
MAILLOT_DE_BAIN_HOMME
MAILLOT_DE_BAIN_HOMME
MAILLOT_DE_BAIN_FEMME
MAILLOT_DE_BAIN_FEMME
BONNET_DE_BAIN
BONNET_DE_BAIN
JE_WATCH
JE_WATCH
JE_WATCH
JE_WATCH
JE_WATCH

2. Afficher les noms de produits avec un espace propre (TRIM) autour.

produit_sans_espace
E_bike1000
E_bike1001
E_bike1002
Road_bike1000
Road_bike1001
Road_bike1002
Road_bike1003
Road_bike1004
VTT1000
VTT1001
VTT1002
VTT1003
VTT1004
VTT1005
VTC1000
VTC1001

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_sans_espace
VTC1002
Batterie1000
Batterie1001
Cables_Ebike
Cables_Road_bike
Cables_VTT
Cables_VTC
Freins_Ebike
Freins_Road_bike
Freins_VTT
Freins_VTC
Selle_Ebike
Selle_Road_bike
Selle_VTT
Selle_VTC
Combinaison
Bike_shoes
Oesics_run
Oesics_run
Oesics_run
Oesics_run
Abidas_shoes
Abidas_shoes
Abidas_shoes
Abidas_shoes
Mike_shoes
Mike_shoes
Mike_shoes
Mike_shoes
Oesics_run_socks
Oesics_run_socks
Abidas_run_socks
Abidas_run_socks
Mike_run_socks
Mike_run_socks
Semelle_running
Tshirt
Tshirt
Tshirt
Tshirt
Tshirt
Tshirt
Tshirt
Tshirt
Short
Short
Short
Short
SV_Homme
SV_Femme
Maillot_de_bain_Homme
Maillot_de_bain_Homme

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_sans_espace
Maillot_de_bain_Femme
Maillot_de_bain_Femme
Bonnet_de_bain
Bonnet_de_bain
JE_watch
JE_watch
JE_watch
JE_watch
JE_watch

3. Afficher les 3 premières lettres du prénom des vendeurs.

left
Nie
Alb
Emm
Isa
Mar
Erw

4. Afficher un indicateur email modifié (nom + '@formations.com') pour les vendeurs.

concat
Bohr@formation.com
Einstein@formation.com
Noether@formation.com
Newton@formation.com
Curie@formation.com
Schrodingher@formation.com

5. Afficher nom + prénom des clients dans une seule colonne (TOP 20).

nom_complet
prenom1 nom1
prenom2 nom2
prenom3 nom3
prenom4 nom4
prenom5 nom5
prenom6 nom6
prenom7 nom7
prenom8 nom8
prenom9 nom9
prenom10 nom10
prenom11 nom11
prenom12 nom12

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom_complet
prenom13 nom13
prenom14 nom14
prenom15 nom15
prenom16 nom16
prenom17 nom17
prenom18 nom18
prenom19 nom19
prenom20 nom20

6. Afficher la longueur du nom de produit.

nom	longueur
E_bike1000	10
E_bike1001	10
E_bike1002	10
Road_bike1000	13
Road_bike1001	13
Road_bike1002	13
Road_bike1003	13
Road_bike1004	13
VTT1000	7
VTT1001	7
VTT1002	7
VTT1003	7
VTT1004	7
VTT1005	7
VTC1000	7
VTC1001	7
VTC1002	7
Batterie1000	12
Batterie1001	12
Cables_Ebike	12
Cables_Road_bike	16
Cables_VTT	10
Cables_VTC	10
Freins_Ebike	12
Freins_Road_bike	16
Freins_VTT	10
Freins_VTC	10
Selle_Ebike	11
Selle_Road_bike	15
Selle_VTT	9
Selle_VTC	9
Combinaison	11
Bike_shoes	10
Oesics_run	10
Oesics_run	10
Oesics_run	10
Oesics_run	10
Abidas_shoes	12

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	longueur
Abidas_shoes	12
Abidas_shoes	12
Abidas_shoes	12
Mike_shoes	10
Mike_shoes	10
Mike_shoes	10
Mike_shoes	10
Oesics_run_socks	16
Oesics_run_socks	16
Abidas_run_socks	16
Abidas_run_socks	16
Mike_run_socks	14
Mike_run_socks	14
Semelle_running	15
Tshirt	6
Tshirt	6
Tshirt	6
Tshirt	6
Tshirt	6
Tshirt	6
Tshirt	6
Tshirt	6
Short	5
Short	5
Short	5
Short	5
SV_Homme	8
SV_Femme	8
Maillot_de_bain_Homme	21
Maillot_de_bain_Homme	21
Maillot_de_bain_Femme	21
Maillot_de_bain_Femme	21
Bonnet_de_bain	14
Bonnet_de_bain	14
JE_watch	8
JE_watch	8
JE_watch	8
JE_watch	8
JE_watch	8

7. Afficher l'année de naissance des clients (TOP 20).

annee
1991
1975
1991
1997
1967
1966
1989

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

annee
2001
2000
1999
1976
1967
2002
1979
1986
2005
2001
2005
2004
1965

8. Afficher le mois de vente (TOP 20).

mois_de_vente
10
5
8
6
5
1
10
8
1
8
7
6
2
9
10
10
3
7
6
12

9. Calculer l'âge des clients (TOP 20).

age
34
50
34
28
58
59
36

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

age
25
25
26
49
58
23
46
40
20
24
20
21
60

10. Afficher uniquement les ventes du mois de juin dans *Base_N* (TOP 20).

vente_id	date_de_vente	client_id	vendeur_id	produit_id	quantite	discount
V2025000004	2025-06-24	23556	2	1	4	0.05
V2025000012	2025-06-30	12045	6	1	3	0.00
V2025000019	2025-06-16	3849	1	1	2	0.06
V2025000026	2025-06-17	3002	2	1	2	0.03
V2025000041	2025-06-06	19836	4	1	1	0.05
V2025000054	2025-06-16	10799	2	1	4	0.07
V2025000057	2025-06-26	9164	6	2	2	0.05
V2025000066	2025-06-03	4083	3	2	2	0.00
V2025000067	2025-06-10	11335	6	2	4	0.00
V2025000098	2025-06-24	22474	4	2	4	0.06
V2025000099	2025-06-12	6711	5	2	2	0.02
V2025000105	2025-06-30	10756	5	2	3	0.02
V2025000106	2025-06-24	189	2	2	3	0.06
V2025000173	2025-06-10	16973	4	4	2	0.01
V2025000193	2025-06-12	9659	4	4	4	0.06
V2025000200	2025-06-26	18566	5	4	1	0.10
V2025000203	2025-06-04	1573	1	4	3	0.06
V2025000230	2025-06-03	8423	2	5	2	0.10
V2025000248	2025-06-01	3896	3	5	2	0.10
V2025000252	2025-06-20	5557	6	5	4	0.00

11. Afficher la colonne quantite en REAL (TOP 20).

quantite_en_float
4
4
3
4
1
1
3

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

quantite_en_float
2
1
1
3
3
1
1
1
3
2
4
2
2

12. Afficher le prix de vente unitaire en TTC avec une conversion.

pvu_en_ttc
9000.0
9360.0
9240.0
4440.0
4680.0
4680.0
4680.0
4680.0
3180.0
3240.0
3240.0
3240.0
3240.0
3240.0
1140.0
1188.0
1188.0
1152.0
1188.0
487.2
108.0
102.0
90.0
165.6
26.4
12.0
12.0
180.0
108.0
72.0
72.0
129.6
102.0

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

pvu_en_ttc
228.0
228.0
228.0
228.0
186.0
186.0
186.0
186.0
222.0
222.0
222.0
222.0
7.2
7.2
7.2
7.2
7.2
7.2
30.0
18.0
18.0
18.0
18.0
18.0
18.0
18.0
18.0
22.8
22.8
22.8
22.8
30.0
27.6
28.8
28.8
28.8
28.8
14.4
14.4
478.8
478.8
478.8
478.8
478.8

13. Dans la table *Clients*, afficher un statut selon l'âge :

- <25 = Jeune
- 25-60 = Adulte
- > 60 = Senior

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

age	categorie_age
34	Adulte
50	Adulte
34	Adulte
28	Adulte
58	Adulte
59	Adulte
36	Adulte
25	Adulte
25	Adulte
26	Adulte
49	Adulte
58	Adulte
23	Jeune
46	Adulte
40	Adulte
20	Jeune
24	Jeune
20	Jeune
21	Jeune
60	Adulte
23	Jeune
43	Adulte
36	Adulte
61	Senior
45	Adulte
23	Jeune
38	Adulte
21	Jeune
32	Adulte
25	Adulte

14. Dans la table *Produits*, Afficher une colonne “Produit : nom_du_produit — Cher/Moyen/Économique” selon le prix avec la règle suivante :

- >1000 = Cher
- 50-1000 = Moyen
- < 50 = Économique

libelle
Produit : E_bike1000 — Cher
Produit : E_bike1001 — Cher
Produit : E_bike1002 — Cher
Produit : Road_bike1000 — Cher
Produit : Road_bike1001 — Cher
Produit : Road_bike1002 — Cher
Produit : Road_bike1003 — Cher
Produit : Road_bike1004 — Cher
Produit : VTT1000 — Cher

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

libelle
Produit : VTT1001 — Cher
Produit : VTT1002 — Cher
Produit : VTT1003 — Cher
Produit : VTT1004 — Cher
Produit : VTT1005 — Cher
Produit : VTC1000 — Moyen
Produit : VTC1001 — Moyen
Produit : VTC1002 — Moyen
Produit : Batterie1000 — Moyen
Produit : Batterie1001 — Moyen
Produit : Cables_Ebike — Moyen
Produit : Cables_Road_bike — Moyen
Produit : Cables_VTT — Moyen
Produit : Cables_VTC — Moyen
Produit : Freins_Ebike — Moyen
Produit : Freins_Road_bike — Économique
Produit : Freins_VTT — Économique
Produit : Freins_VTC — Économique
Produit : Selle_Ebike — Moyen
Produit : Selle_Road_bike — Moyen
Produit : Selle_VTT — Moyen
Produit : Selle_VTC — Moyen
Produit : Combinaison — Moyen
Produit : Bike_shoes — Moyen
Produit : Oesics_run — Moyen
Produit : Oesics_run — Moyen
Produit : Oesics_run — Moyen
Produit : Oesics_run — Moyen
Produit : Abidas_shoes — Moyen
Produit : Abidas_shoes — Moyen
Produit : Abidas_shoes — Moyen
Produit : Abidas_shoes — Moyen
Produit : Mike_shoes — Moyen
Produit : Mike_shoes — Moyen
Produit : Mike_shoes — Moyen
Produit : Mike_shoes — Moyen
Produit : Oesics_run_socks — Économique
Produit : Oesics_run_socks — Économique
Produit : Abidas_run_socks — Économique
Produit : Abidas_run_socks — Économique
Produit : Mike_run_socks — Économique
Produit : Mike_run_socks — Économique
Produit : Semelle_running — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique
Produit : Tshirt — Économique

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

libelle
Produit : Short — Économique
Produit : Short — Économique
Produit : Short — Économique
Produit : Short — Économique
Produit : SV_Homme — Économique
Produit : SV_Femme — Économique
Produit : Maillot_de_bain_Homme — Économique
Produit : Maillot_de_bain_Homme — Économique
Produit : Maillot_de_bain_Femme — Économique
Produit : Maillot_de_bain_Femme — Économique
Produit : Bonnet_de_bain — Économique
Produit : Bonnet_de_bain — Économique
Produit : JE_watch — Moyen
Produit : JE_watch — Moyen
Produit : JE_watch — Moyen
Produit : JE_watch — Moyen
Produit : JE_watch — Moyen

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

4.6 Solutions

1. Afficher le nom du produit en MAJUSCULE.

```sql
SELECT UPPER(nom) as PRODUIT_EN_MAJUSCULE
FROM Produits;
```

2. Afficher les noms de produits avec un espace propre (TRIM) autour.

```sql
SELECT TRIM(nom) as produit_sans_espace
FROM Produits;
```

3. Afficher les 3 premières lettres du prénom des vendeurs.

```sql
SELECT LEFT(prenom, 3)
FROM Vendeurs;
```

4. Afficher un indicateur email modifié (nom + '@formations.com') pour les vendeurs.

```sql
SELECT CONCAT(nom, '@formation.com')
FROM Vendeurs;
```

5. Afficher nom + prénom des clients dans une seule colonne (TOP 20).

```sql
SELECT CONCAT(prenom, ' ', nom) AS nom_complet
FROM Clients
LIMIT 20;
```

6. Afficher la longueur du nom de produit.

```sql
SELECT nom, LENGTH(nom) AS longueur
FROM Produits;
```

7. Afficher l'année de naissance des clients (TOP 20).

```sql
SELECT EXTRACT(YEAR FROM date_naissance) AS annee
FROM Clients
LIMIT 20;
```

8. Afficher le mois de vente (TOP 20).

```sql
SELECT EXTRACT(MONTH FROM date_de_vente) as mois_de_vente
FROM Base_N
LIMIT 20;
```

9. Calculer l'âge des clients (TOP 20).

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT EXTRACT(YEAR FROM AGE(CURRENT_DATE, date_naissance)) AS age
FROM Clients
LIMIT 20;
```

10. Afficher uniquement les ventes du mois de juin dans *Base_N* (TOP 20).

```sql
SELECT *
FROM Base_N
WHERE EXTRACT(MONTH FROM date_de_vente) = 06
LIMIT 20;
```

11. Afficher la colonne quantite en FLOAT (TOP 20).

```sql
SELECT CAST(quantite as REAL) as quantite_en_float
FROM Base_N
LIMIT 20;
```

12. Afficher le prix de vente unitaire en TTC avec une conversion.

```sql
SELECT CAST(prix_vente_unitaire AS DECIMAL(10,2)) * 1.2 as pvu_en_TTC
FROM Produits;
```

13. Dans la table *Clients*, afficher un statut selon l'âge :

- <25 = jeune
- 25-50 = adulte
- > 60 = senior

```sql
SELECT
    EXTRACT(YEAR FROM AGE(CURRENT_DATE, date_naissance)) AS age,
    CASE
        WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, date_naissance)) < 25 THEN 'Jeune'
        WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, date_naissance)) <= 60 THEN 'Adulte'
        ELSE 'Senior'
    END AS categorie_age
FROM Clients
LIMIT 30;
```

14. Dans la table *Produits*, Afficher une colonne "Produit : nom_du_produit — Cher/Moyen/Économique" selon le prix avec la règle suivante :

- >1000 = Cher
- 50-1000 = Moyen
- < 50 = Économique

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT
    CONCAT(
        'Produit : ',
        nom,
        ' - ',
        CASE
            WHEN prix_vente_unitaire > 1000 THEN 'Cher'
            WHEN prix_vente_unitaire >= 50 THEN 'Moyen'
            ELSE 'Économique'
        END
    ) AS libelle
FROM Produits;;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5. Interroger plusieurs tables

5.1 Jointures SQL

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table.

Dans notre base de données, on a par exemple le prix de vente des produits dans la table *Produits* et les quantités vendues dans *Base_N*. Grâce à la jointure, il sera possible d'obtenir le chiffre d'affaires global en une seule requête.

5.2 Types de jointures

Il existe plusieurs types de jointures selon le résultat à obtenir. Par souci de pédagogie, nous ne les étudierons pas toutes. Nous nous concentrerons sur les types de jointures suivants :

- **INNER JOIN** : jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes.
- **LEFT JOIN** (ou LEFT OUTER JOIN) : jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifiée dans l'autre table.
- **RIGHT JOIN** (ou RIGHT OUTER JOIN) : jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifiée dans l'autre table.
- **FULL JOIN** (ou FULL OUTER JOIN) : jointure externe pour retourner les résultats quand la condition est vraie dans au moins une des 2 tables.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5.2.1 INNER JOIN

Cette jointure ne retourne que les lignes qui ont une correspondance des deux côtés.

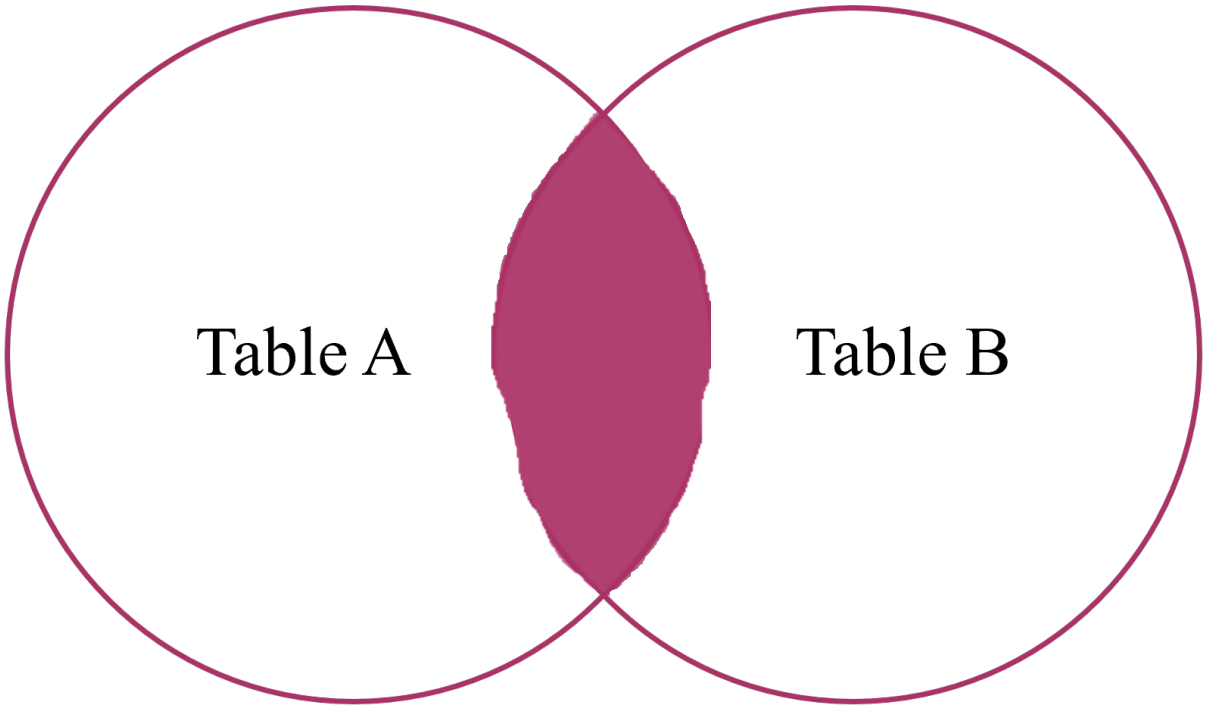

Figure 2: Jointure interne : INNER JOIN

La syntaxe de base est la suivante :

```
SELECT ...
FROM Table A
INNER JOIN Table B ON Table A.id = Table B.id
```

On peut par exemple afficher les ventes avec le nom des vendeurs (TOP 50) :

```
SELECT Base_N.vente_id,
    Base_N.produit_id,
    Base_N.quantite,
    Base_N.date_de_vente,
    Vendeurs.nom,
    Vendeurs.prenom
FROM Base_N
INNER JOIN Vendeurs
    ON Base_N.vendeur_id = Vendeurs.vendeur_id
    LIMIT 30;
```

vente_id	produit_id	quantite	date_de_vente	nom	prenom
V2025000001	1	4	2025-10-18	Schrodingher	Erwin
V2025000002	1	4	2025-05-10	Schrodingher	Erwin

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vente_id	produit_id	quantite	date_de_vente	nom	prenom
V2025000003	1	3	2025-08-11	Noether	Emmy
V2025000004	1	4	2025-06-24	Einstein	Albert
V2025000005	1	1	2025-05-27	Bohr	Niels
V2025000006	1	1	2025-01-09	Noether	Emmy
V2025000007	1	3	2025-10-29	Curie	Marie
V2025000008	1	2	2025-08-04	Schrodingher	Erwin
V2025000009	1	1	2025-01-17	Newton	Isaac
V2025000010	1	1	2025-08-08	Noether	Emmy
V2025000011	1	3	2025-07-21	Newton	Isaac
V2025000012	1	3	2025-06-30	Schrodingher	Erwin
V2025000013	1	1	2025-02-03	Schrodingher	Erwin
V2025000014	1	1	2025-09-10	Noether	Emmy
V2025000015	1	1	2025-10-10	Newton	Isaac
V2025000016	1	3	2025-10-07	Schrodingher	Erwin
V2025000017	1	2	2025-03-18	Curie	Marie
V2025000018	1	4	2025-07-03	Einstein	Albert
V2025000019	1	2	2025-06-16	Bohr	Niels
V2025000020	1	2	2025-12-06	Einstein	Albert
V2025000021	1	2	2025-09-04	Newton	Isaac
V2025000022	1	4	2025-07-05	Noether	Emmy
V2025000023	1	4	2025-05-19	Curie	Marie
V2025000024	1	2	2025-07-08	Schrodingher	Erwin
V2025000025	1	4	2025-07-17	Einstein	Albert
V2025000026	1	2	2025-06-17	Einstein	Albert
V2025000027	1	4	2025-08-07	Newton	Isaac
V2025000028	1	1	2025-01-03	Curie	Marie
V2025000029	1	2	2025-11-02	Curie	Marie
V2025000030	1	4	2025-10-28	Newton	Isaac

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5.2.2 LEFT JOIN

Cette jointure retourne toutes les lignes de gauche (A) + les correspondances de droite (B).

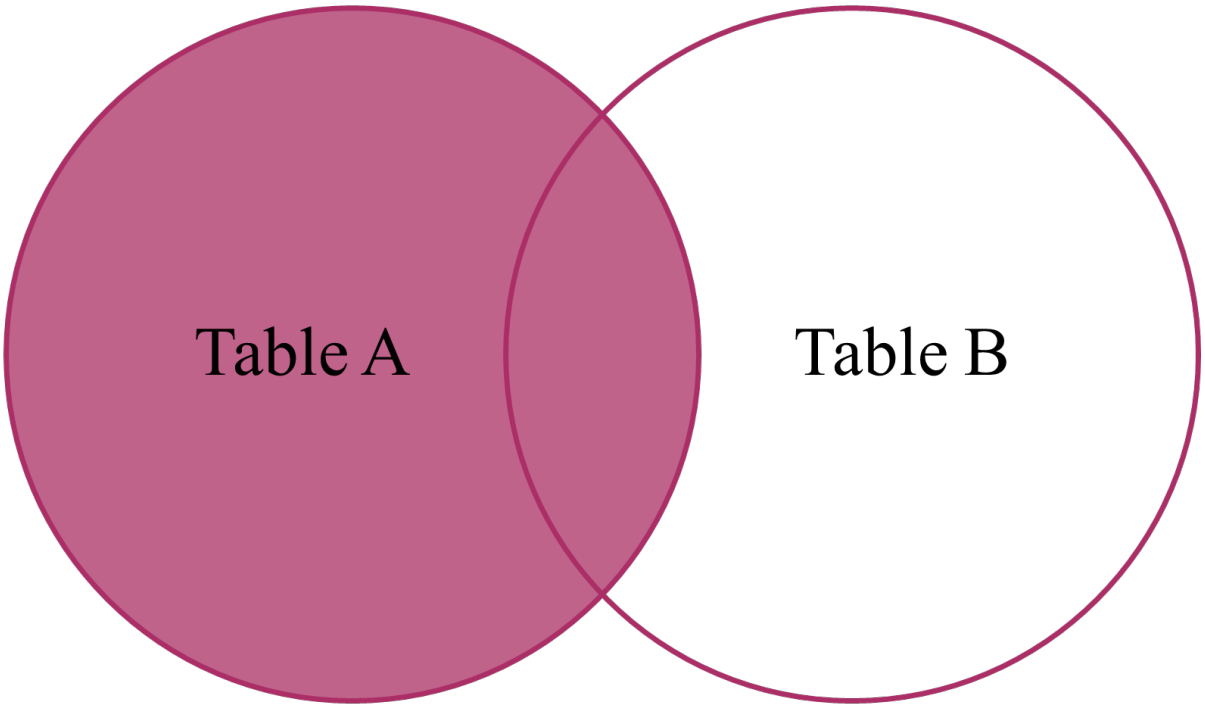

Figure 3: Jointure gauche : LEFT JOIN

La syntaxe de base est la suivante :

```
SELECT ...
FROM Table A
LEFT JOIN Table B ON Table A.id = Table B.id
```

On peut afficher les clients qui n'ont pas acheté durant l'année N-1 dans la table *Base_N_1* (TOP 50) :

```
SELECT Clients.client_id,
    Clients.nom
FROM Clients
LEFT JOIN Base_N_1
    ON Clients.client_id = Base_N_1.client_id
    WHERE Base_N_1.vente_id is null
    LIMIT 30
```

client_id	nom
21370	nom21370
17928	nom17928
22262	nom22262
16703	nom16703
14724	nom14724

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

client_id	nom
23772	nom23772
19915	nom19915
20566	nom20566
22391	nom22391
21905	nom21905
19491	nom19491
17116	nom17116
11257	nom11257
16828	nom16828
22378	nom22378
18562	nom18562
14619	nom14619
22383	nom22383
16533	nom16533
15709	nom15709
14291	nom14291
3367	nom3367
22883	nom22883
16106	nom16106
20943	nom20943
16278	nom16278
21854	nom21854
15280	nom15280
12648	nom12648
21273	nom21273

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5.2.3 RIGHT JOIN

C'est l'inverse de LEFT JOIN.

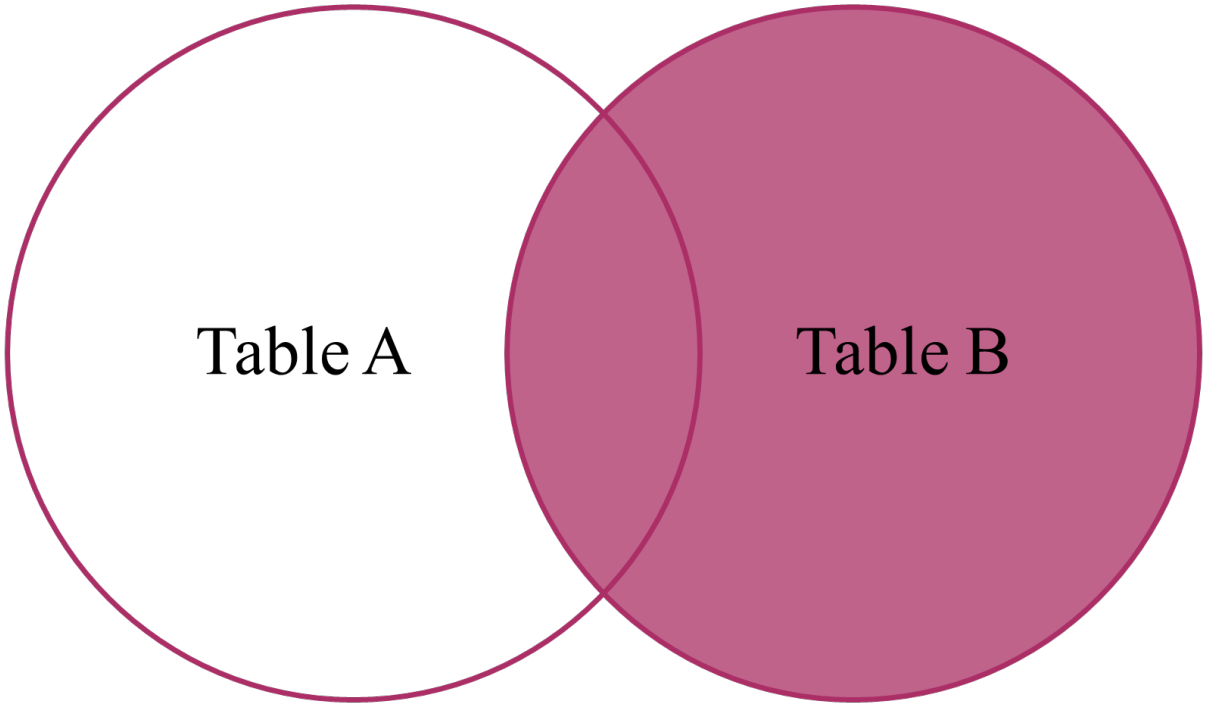

Figure 4: Jointure droite : RIGHT JOIN

La syntaxe de base est la suivante :

```
SELECT ...
FROM Table A
RIGHT JOIN Table B ON Table A.id = Table B.id
```

Pour illustrer la symétrie avec le LEFT JOIN, on va afficher la même chose avec le RIGHT JOIN. On va donc afficher les clients qui n'ont pas acheté à l'année N-1 dans la table *Base_N_1* :

```
SELECT
    Clients.client_id,
    Clients.nom
FROM Base_n_1
RIGHT JOIN Clients
    ON Clients.client_id = Base_N_1.client_id
    WHERE Base_N_1.vente_id is null
LIMIT 30
```

client_id	nom
21370	nom21370
17928	nom17928
22262	nom22262

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

client_id	nom
16703	nom16703
14724	nom14724
23772	nom23772
19915	nom19915
20566	nom20566
22391	nom22391
21905	nom21905
19491	nom19491
17116	nom17116
11257	nom11257
16828	nom16828
22378	nom22378
18562	nom18562
14619	nom14619
22383	nom22383
16533	nom16533
15709	nom15709
14291	nom14291
3367	nom3367
22883	nom22883
16106	nom16106
20943	nom20943
16278	nom16278
21854	nom21854
15280	nom15280
12648	nom12648
21273	nom21273

On retrouve le même résultat à condition d'inverser l'ordre des tables.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5.2.4 FULL OUTER JOIN

Elle retourne toutes les lignes même sans correspondance.

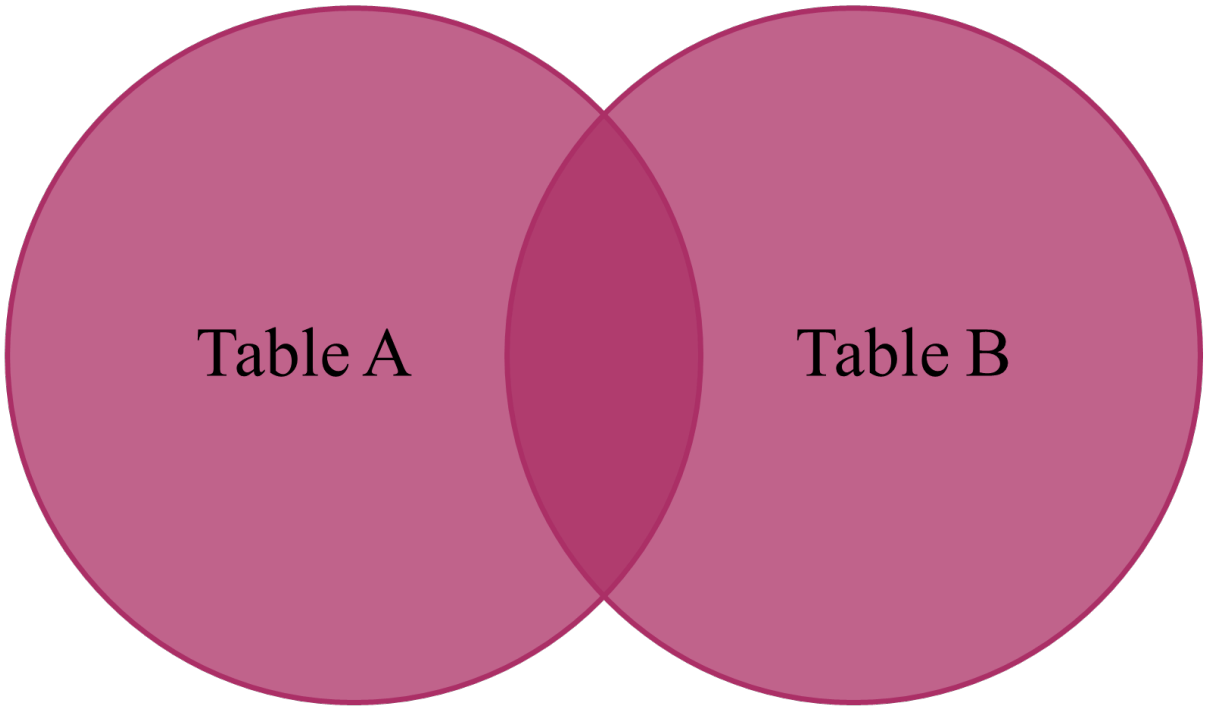

Figure 5: Jointure externe : OUTER JOIN

La syntaxe de base est la suivante :

```
SELECT ...
FROM Table A
FULL JOIN Table B ON Table A.id = Table B.id
```

On peut lister toutes les ventes et tous les produits même sans correspondance :

```
SELECT
    Produits.nom,
    Base_N.vente_id
FROM Produits
FULL JOIN Base_N
    ON Produits.produit_id = Base_N.produit_id;
```

nom	vente_id
E_bike1000	V2025000001
E_bike1000	V2025000002
E_bike1000	V2025000003
E_bike1000	V2025000004
E_bike1000	V2025000005

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	vente_id
E_bike1000	V2025000006
E_bike1000	V2025000007
E_bike1000	V2025000008
E_bike1000	V2025000009
E_bike1000	V2025000010
E_bike1000	V2025000011
E_bike1000	V2025000012
E_bike1000	V2025000013
E_bike1000	V2025000014
E_bike1000	V2025000015
E_bike1000	V2025000016
E_bike1000	V2025000017
E_bike1000	V2025000018
E_bike1000	V2025000019
E_bike1000	V2025000020
E_bike1000	V2025000021
E_bike1000	V2025000022
E_bike1000	V2025000023
E_bike1000	V2025000024
E_bike1000	V2025000025
E_bike1000	V2025000026
E_bike1000	V2025000027
E_bike1000	V2025000028
E_bike1000	V2025000029
E_bike1000	V2025000030

Cela peut potentiellement conduire à plusieurs valeurs vides ce qui est à éviter si possible. Privilégier une jointure INNER ou LEFT (RIGHT).

5.3 Combiner sans jointure

Des opérateurs permettent de combiner les résultats de plusieurs requêtes sans utiliser de jointures.

Ils s'utilisent **uniquement** quand :

- Les requêtes ont le même nombre de colonnes
- Les colonnes correspondantes ont des types compatibles

5.3.1 UNION

Cet opérateur permet de combiner les résultats de deux requêtes et supprime les doublons.

La syntaxe de base est la suivante :

```
SELECT ...
UNION -- Même nombre de colonnes en bas et en haut avec des types compatibles
SELECT ...
```

On peut par exemple afficher la liste des produits vendus les 2 dernières années (tables *Base_N* et *Base_N_1*) et afficher cela en une fois :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
SELECT DISTINCT
produit_id
from Base_N

UNION

SELECT DISTINCT
produit_id
from Base_N_1
```

produit_id
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

5.3.2 UNION ALL

Elle fait la même chose mais ne supprime pas les doublons. Son principal avantage réside dans le fait qu'elle est plus rapide.

```
SELECT ...
UNION ALL -- Même nombre de colonnes en bas et en haut avec des types compatibles
SELECT ...
```

La même requête que précédemment avec UNION ALL à la place donne le résultat suivant :

```
SELECT DISTINCT
produit_id
from Base_N
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
UNION ALL

SELECT DISTINCT
produit_id
from Base_N_1
```

produit_id
74
54
29
71
68
4
34
51
70
52
67
63
10
35
45
6
39
69
36
31
50
60
14
66
22
59
13
65
2
16
62
75
73
44
11
42
41
46
40
43
53
32
9
7
38

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id
15
48
26
12
72
57
24
61
19
77
25
30
21
49
47
3
17
37
28
20
33
1
76
5
18
64
55
27
23
56
58
8
74
54
29
71
68
4
34
51
70
52
67
63
10
35
6
39
69
36
31
50

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id
60
14
66
22
59
13
65
2
16
62
75
73
44
11
42
46
40
43
53
32
9
7
38
15
48
26
12
72
57
24
61
19
77
25
30
21
49
47
3
17
28
20
33
1
76
5
18
55
27
23
56
8

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Si un produit est vendu dans les deux années, il apparaît deux fois.

5.3.3 INTERSECT

Cet opérateur renvoie seulement les éléments communs entre deux requêtes.

La syntaxe de base est la suivante :

```
SELECT ...
INTERSECT -- Même nombre de colonnes en bas et en haut avec des types compatibles
SELECT ...
```

On peut vouloir afficher la liste des produits vendus l'année N **ET** l'année N-1 :

```
SELECT DISTINCT
produit_id
from Base_N

INTERSECT

SELECT DISTINCT
produit_id
from Base_N_1
```

produit_id
74
29
54
71
4
68
34
51
52
70
67
63
10
35
6
39
36
69
31
50
60
14
66
22
59
13
65

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit_id
2
16
62
75
73
11
44
42
40
46
43
32
53
7
9
38
15
12
26
48
72
24
57
19
61
77
25
30
21
49
47
3
17
20
28
33
1
76
5
18
55
27
23
56
8

5.3.4 EXCEPT

Cet opérateur renvoie ce qui est dans la première requête **mais pas dans la seconde**.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
SELECT ...
EXCEPT -- Même nombre de colonnes en bas et en haut avec des types compatibles
SELECT ...
```

Cette fois-ci, on peut vouloir afficher la liste des produits vendus l'année N **mais pas** l'année N-1 :

```
SELECT DISTINCT
produit_id
from Base_N

EXCEPT

SELECT DISTINCT
produit_id
from Base_N_1
```

produit_id
45
41
37
64
58

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5.4 Exercices

1. Afficher les produits avec leur sous-catégorie.

produit	sous_cat
E_bike1000	Velo electrique
E_bike1001	Velo electrique
E_bike1002	Velo electrique
Road_bike1000	Velo de Route
Road_bike1001	Velo de Route
Road_bike1002	Velo de Route
Road_bike1003	Velo de Route
Road_bike1004	Velo de Route
VTT1000	VTT
VTT1001	VTT
VTT1002	VTT
VTT1003	VTT
VTT1004	VTT
VTT1005	VTT
VTC1000	VTC
VTC1001	VTC
VTC1002	VTC
Batterie1000	Batterie
Batterie1001	Batterie
Cables_Ebike	Cables
Cables_Road_bike	Cables
Cables_VTT	Cables
Cables_VTC	Cables
Freins_Ebike	Freins
Freins_Road_bike	Freins
Freins_VTT	Freins
Freins_VTC	Freins
Selle_Ebike	Selle
Selle_Road_bike	Selle
Selle_VTT	Selle
Selle_VTC	Selle
Combinaison	Combinaison
Bike_shoes	Chaussure de velo
Oesics_run	Chaussure de course
Oesics_run	Chaussure de course
Oesics_run	Chaussure de course
Oesics_run	Chaussure de course
Abidas_shoes	Chaussure de course
Abidas_shoes	Chaussure de course
Abidas_shoes	Chaussure de course
Abidas_shoes	Chaussure de course
Mike_shoes	Chaussure de course
Mike_shoes	Chaussure de course
Mike_shoes	Chaussure de course
Mike_shoes	Chaussure de course
Oesics_run_socks	Chaussettes
Oesics_run_socks	Chaussettes

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit	sous_cat
Abidas_run_socks	Chaussettes
Abidas_run_socks	Chaussettes
Mike_run_socks	Chaussettes
Mike_run_socks	Chaussettes
Semelle_running	Semelles
Tshirt	T-shirt
Tshirt	T-shirt
Tshirt	T-shirt
Tshirt	T-shirt
Tshirt	T-shirt
Tshirt	T-shirt
Tshirt	T-shirt
Tshirt	T-shirt
Short	Short
Short	Short
Short	Short
Short	Short
SV_Homme	Sous-vêtement
SV_Femme	Sous-vêtement
Maillot_de_bain_Homme	Maillot de bain
Maillot_de_bain_Homme	Maillot de bain
Maillot_de_bain_Femme	Maillot de bain
Maillot_de_bain_Femme	Maillot de bain
Bonnet_de_bain	Bonnet de bain
Bonnet_de_bain	Bonnet de bain
JE_watch	Montre connectee
JE_watch	Montre connectee
JE_watch	Montre connectee
JE_watch	Montre connectee
JE_watch	Montre connectee

2. Afficher le total des ventes par produit.

produit	total_vendu
Tshirt	56840
Bonnet_de_bain	50960
Maillot_de_bain_Homme	49746
Maillot_de_bain_Femme	49515
Short	28980
Oesics_run_socks	14571
Abidas_run_socks	14358
Mike_run_socks	14054
JE_watch	10673
Abidas_shoes	8691
Oesics_run	8604
Mike_shoes	8308
SV_Homme	8043
SV_Femme	7547
Semelle_running	6535
Bike_shoes	2062

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit	total_vendu
Cables_VTT	1758
Cables_Road_bike	1642
Freins_VTT	1590
Batterie1001	1562
Cables_Ebike	1549
Freins_Road_bike	1545
Combinaison	1531
Freins_Ebike	1487
Selle_Ebike	1484
Selle_VTC	1466
Cables_VTC	1455
Selle_VTT	1430
Freins_VTC	1429
Batterie1000	1406
Selle_Road_bike	1368
VTT1000	171
E_bike1002	163
VTT1003	161
E_bike1001	160
VTC1002	160
E_bike1000	156
VTT1002	156
Road_bike1002	155
VTT1001	154
Road_bike1001	146
Road_bike1000	138
VTC1000	137
Road_bike1004	136
Road_bike1003	128
VTT1005	116
VTC1001	94
VTT1004	84

3. Top 3 des produits les plus vendus.

produit	total_vendu
Tshirt	56840
Bonnet_de_bain	50960
Maillot_de_bain_Homme	49746

4. Afficher les produits jamais vendus.

produit_id	nom

5. Calculer le chiffre d'affaires par vendeur.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vendeur	chiffre_affaires
Einstein	4902497
Newton	4736339
Schrodingher	4720590
Bohr	4671312
Curie	4588762
Noether	4445318

6. CA par produit avec un CASE (classification).

- >1000 000 = Top
- 500 000-1000 000 = Moyen
- < 500 000 = Faible

produit	ca	niveau
JE_watch	4258527	TOP
Oesics_run	1634760	TOP
Batterie1001	1546380	TOP
Mike_shoes	1536980	TOP
Batterie1000	1349760	TOP
Abidas_shoes	1347105	TOP
E_bike1002	1255100	TOP
E_bike1001	1248000	TOP
Maillot_de_bain_Homme	1193904	TOP
Maillot_de_bain_Femme	1188360	TOP
E_bike1000	1170000	TOP
Tshirt	852600	MOYEN
Cables_Ebike	628894	MOYEN
Bonnet_de_bain	611520	MOYEN
Road_bike1002	604500	MOYEN
Road_bike1001	569400	MOYEN
Short	550620	MOYEN
Road_bike1004	530400	MOYEN
Road_bike1000	510600	MOYEN
Road_bike1003	499200	FAIBLE
VTT1000	453150	FAIBLE
VTT1003	434700	FAIBLE
VTT1002	421200	FAIBLE
VTT1001	415800	FAIBLE
VTT1005	313200	FAIBLE
VTT1004	226800	FAIBLE
Selle_Ebike	222600	FAIBLE
Freins_Ebike	205206	FAIBLE
SV_Homme	201075	FAIBLE
Bike_shoes	175270	FAIBLE
SV_Femme	173581	FAIBLE
Combinaison	165348	FAIBLE
Semelle_running	163375	FAIBLE
VTC1002	158400	FAIBLE

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit	ca	niveau
Cables_VTT	149430	FAIBLE
Cables_Road_bike	147780	FAIBLE
VTC1000	130150	FAIBLE
Selle_Road_bike	123120	FAIBLE
Cables_VTC	109125	FAIBLE
VTC1001	93060	FAIBLE
Selle_VTC	87960	FAIBLE
Oesics_run_socks	87426	FAIBLE
Abidas_run_socks	86148	FAIBLE
Selle_VTT	85800	FAIBLE
Mike_run_socks	84324	FAIBLE
Freins_Road_bike	33990	FAIBLE
Freins_VTT	15900	FAIBLE
Freins_VTC	14290	FAIBLE

7. Classement des noms de produits par quantité vendue (RANK) (table *Base_N*).

produit	total	rang
Tshirt	56840	1
Bonnet_de_bain	50960	2
Maillot_de_bain_Homme	49746	3
Maillot_de_bain_Femme	49515	4
Short	28980	5
Oesics_run_socks	14571	6
Abidas_run_socks	14358	7
Mike_run_socks	14054	8
JE_watch	10673	9
Abidas_shoes	8691	10
Oesics_run	8604	11
Mike_shoes	8308	12
SV_Homme	8043	13
SV_Femme	7547	14
Semelle_running	6535	15
Bike_shoes	2062	16
Cables_VTT	1758	17
Cables_Road_bike	1642	18
Freins_VTT	1590	19
Batterie1001	1562	20
Cables_Ebike	1549	21
Freins_Road_bike	1545	22
Combinaison	1531	23
Freins_Ebike	1487	24
Selle_Ebike	1484	25
Selle_VTC	1466	26
Cables_VTC	1455	27
Selle_VTT	1430	28
Freins_VTC	1429	29
Batterie1000	1406	30
Selle_Road_bike	1368	31
VTT1000	171	32

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

produit	total	rang
E_bike1002	163	33
VTT1003	161	34
E_bike1001	160	35
VTC1002	160	35
E_bike1000	156	37
VTT1002	156	37
Road_bike1002	155	39
VTT1001	154	40
Road_bike1001	146	41
Road_bike1000	138	42
VTC1000	137	43
Road_bike1004	136	44
Road_bike1003	128	45
VTT1005	116	46
VTC1001	94	47
VTT1004	84	48

8. CA par catégorie de produit.

categorie	chiffre_affaires
Velo	9033660
Accessoire de velo	4885583
Chaussure	4694115
Tech	4258527
Natation	2993784
Accessoire de course	2199149

9. CA par magasin.

magasin	chiffre_affaires
JethlonLyon	9573809
JethlonMarseille	9309352
JethlonParis	9181657

10. Nombres de ventes par région (implantation des clients).

region	nb_ventes
Occitanie	6220
Nouvelle-Aquitaine	5777
Auvergne-Rhône-Alpes	5628
Grand Est	4701
Bourgogne-Franche-Comté	3801
Ile-de-France	3713
Centre-Val de Loire	2994
Provence-Alpes-Côte d'Azur	2968

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

region	nb_ventes
Hauts-de-France	2529
Normandie	2382
Pays de la Loire	2344
Bretagne	1764
Corse	1027
Autre	484
Guyane	72
Mayotte	68
Martinique	67
La Réunion	59
Guadeloupe	54

11. Age moyen des clients par région.

region	age_moyen
Hauts-de-France	43.56374
La Réunion	40.05556
Bretagne	42.96138
Mayotte	42.88889
Auvergne-Rhône-Alpes	43.80066
Ile-de-France	43.44773
Grand Est	44.18141
Guyane	41.94444
Corse	43.83209
Bourgogne-Franche-Comté	43.12120
Provence-Alpes-Côte d'Azur	43.77158
Autre	40.53846
Martinique	42.17647
Guadeloupe	46.74194
Normandie	43.59516
Nouvelle-Aquitaine	43.32586
Occitanie	43.82569
Pays de la Loire	43.07242
Centre-Val de Loire	44.11645

12. Les produits actifs dans *Base_N* mais pas dans *Base_N_1*.

produit_id
45
41
37
64
58

13. CA par vendeur et par année.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

vendeur_id	vendeur	année	ca
1	Bohr	2023	3248058
1	Bohr	2024	3921367
1	Bohr	2025	4502212
1	Bohr	2026	169100
5	Curie	2023	3492502
5	Curie	2024	3735086
5	Curie	2025	4588762
2	Einstein	2023	3438739
2	Einstein	2024	3914129
2	Einstein	2025	4755297
2	Einstein	2026	147200
4	Newton	2023	3311741
4	Newton	2024	3809093
4	Newton	2025	4736339
3	Noether	2023	3368392
3	Noether	2024	3897761
3	Noether	2025	4445318
6	Schrodingher	2023	3257730
6	Schrodingher	2024	3933463
6	Schrodingher	2025	4674390
6	Schrodingher	2026	46200

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5.5 Solutions

1. Afficher les produits avec leur sous-catégorie.

```
SELECT P.nom AS produit, SC.sous_categorie_produit AS sous_cat
FROM Produits P
INNER JOIN Sous_categories SC ON P.sous_categorie_id = SC.sous_categorie_id;
```

2. Afficher le total des ventes par produit.

```
SELECT P.nom AS produit,
       SUM(V.quantite) AS total_vendu
FROM Base_N V
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY P.nom
ORDER BY total_vendu DESC;
```

3. Top 3 des produits les plus vendus.

```
SELECT P.nom AS produit,
       SUM(V.quantite) AS total_vendu
FROM Base_N V
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY P.nom
ORDER BY total_vendu DESC
LIMIT 3;
```

4. Afficher les produits jamais vendus.

```
SELECT P.produit_id, P.nom
FROM Produits P
LEFT JOIN Base_N V ON P.produit_id = V.produit_id
WHERE V.vente_id IS NULL;
```

5. Calculer le chiffre d'affaires par vendeur.

```
SELECT VR.nom AS vendeur,
       SUM(V.quantite * P.prix_vente_unitaire) AS chiffre_affaires
FROM Base_N V
INNER JOIN Vendeurs VR ON V.vendeur_id = VR.vendeur_id
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY VR.nom
ORDER BY chiffre_affaires DESC;
```

6. CA par produit avec un CASE (classification).

- >1000 000 = Top
- 500 000-1000 000 = Moyen
- < 500 000 = Faible

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT P.nom AS produit,
       SUM(V.quantite * P.prix_vente_unitaire) AS CA,
       CASE
          WHEN SUM(V.quantite * P.prix_vente_unitaire) > 1000000 THEN 'TOP'
          WHEN SUM(V.quantite * P.prix_vente_unitaire) > 500000 THEN 'MOYEN'
          ELSE 'FAIBLE'
       END AS niveau
FROM Base_N V
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY P.nom
ORDER BY SUM(V.quantite * P.prix_vente_unitaire) DESC;
```

7. Classement des noms de produits par quantité vendue (RANK) (table *Base_N*).

```sql
SELECT P.nom AS produit,
       SUM(V.quantite) AS total,
       RANK() OVER(ORDER BY SUM(V.quantite) DESC) AS rang
FROM Base_N V
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY P.nom;
```

8. CA par catégorie de produit.

```sql
SELECT C.categorie_produit AS categorie,
       SUM(V.quantite * P.prix_vente_unitaire) AS chiffre_affaires
FROM Base_N V
INNER JOIN Produits P ON V.produit_id = P.produit_id
INNER JOIN Sous_categories SC ON P.sous_categorie_id = SC.sous_categorie_id
INNER JOIN Categories C ON SC.categorie_id = C.categorie_id
GROUP BY C.categorie_produit
ORDER BY chiffre_affaires DESC;
```

9. CA par magasin.

```sql
SELECT M.Nom AS magasin,
       SUM(V.quantite * P.prix_vente_unitaire) AS chiffre_affaires
FROM Base_N V
INNER JOIN Produits P ON V.produit_id = P.produit_id
INNER JOIN Vendeurs VD ON V.vendeur_id = VD.vendeur_id
INNER JOIN Magasins M ON VD.magasin_id = M.magasin_id
GROUP BY M.Nom
ORDER BY chiffre_affaires DESC;
```

10. Nombres de ventes par région (d'origine des clients).

```sql
SELECT G.region,
       COUNT(V.vente_id) AS nb_ventes
FROM Base_N V
INNER JOIN Clients C ON V.client_id = C.client_id
INNER JOIN Geographies G ON C.geo_id = G.geo_id
GROUP BY G.region
ORDER BY nb_ventes DESC;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

11. Age moyen des clients par région.

```sql
SELECT G.region,
       AVG(EXTRACT(YEAR FROM AGE(CURRENT_DATE, C.date_naissance))) AS age_moyen
FROM Clients C
INNER JOIN Geographies G ON C.geo_id = G.geo_id
GROUP BY G.region;
```

12. Les produits actifs dans *Base_N* mais pas dans *Base_N_1*.

```sql
SELECT DISTINCT produit_id FROM Base_N
EXCEPT
SELECT DISTINCT produit_id FROM Base_N_1;
```

13. CA par vendeur et par année.

```sql
SELECT V.vendeur_id,
       VD.nom AS vendeur,
       EXTRACT(YEAR FROM V.date_de_vente) AS année,
       SUM(V.quantite * P.prix_vente_unitaire) AS CA
FROM Base_N V
INNER JOIN Vendeurs VD ON V.vendeur_id = VD.vendeur_id
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY V.vendeur_id, VD.nom, EXTRACT(YEAR FROM V.date_de_vente)

UNION ALL

SELECT V.vendeur_id,
       VD.nom AS vendeur,
       EXTRACT(YEAR FROM V.date_de_vente) AS année,
       SUM(V.quantite * P.prix_vente_unitaire) AS CA
FROM Base_N_1 V
INNER JOIN Vendeurs VD ON V.vendeur_id = VD.vendeur_id
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY V.vendeur_id, VD.nom, EXTRACT(YEAR FROM V.date_de_vente)

UNION ALL

SELECT V.vendeur_id,
       VD.nom AS vendeur,
       EXTRACT(YEAR FROM V.date_de_vente) AS année,
       SUM(V.quantite * P.prix_vente_unitaire) AS CA
FROM Base_N_2 V
INNER JOIN Vendeurs VD ON V.vendeur_id = VD.vendeur_id
INNER JOIN Produits P ON V.produit_id = P.produit_id
GROUP BY V.vendeur_id, VD.nom, EXTRACT(YEAR FROM V.date_de_vente)

ORDER BY vendeur, année;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6. Sous-requêtes et vues

6.1 Sous-requêtes

Les sous-requêtes sont un pilier du SQL avancé. Elles permettent d'utiliser le résultat d'une requête… dans une autre requête.

Une sous-requête peut apparaître dans :

- **WHERE**
- **IN / NOT IN / ANY / ALL**
- **SELECT**
- **FROM (table dérivée)**
- **HAVING**
- **EXISTS / NOT EXISTS**

Elles peuvent retourner :

- **une seule valeur** → sous-requête scalaire
- **une colonne** → utilisée avec IN ou EXISTS
- **une table entière** → dans FROM

6.1.1 Sous-requêtes dans WHERE

1. Sous-requête scalaire (retourne une valeur)

Exemple : produits plus chers que la moyenne

```
SELECT nom, prix_vente_unitaire
FROM Produits
WHERE prix_vente_unitaire > (
    SELECT AVG(prix_vente_unitaire)
    FROM Produits
);
```

- La sous-requête calcule une seule valeur = le prix moyen.
- La requête principale compare chaque produit à cette valeur.

2. Sous-requête avec **IN** (retourne une liste/ colonne complète)

Exemple : vendeurs ayant vendu

```
SELECT *
FROM Vendeurs
WHERE vendeur_id IN (SELECT vendeur_id FROM Base_N);
```

Exemple : vendeurs n'ayant pas vendu

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT *
FROM Vendeurs
WHERE vendeur_id NOT IN (SELECT vendeur_id FROM Base_N);
```

Une alternative à **NOT IN** est **NOT EXISTS** dans le cas où la sous-requête renverrait des NULL.

3. Sous-requêtes avec **EXISTS / NOT EXISTS**

EXISTS : tester l'existence

```sql
SELECT *
FROM Clients C
WHERE EXISTS (
    SELECT 1
    FROM Base_N V
    WHERE V.client_id = C.client_id
);
```

La requête principale est exécutée uniquement si la sous-requête renvoie un résultat.

NOT EXISTS : tester la non-existence

```sql
SELECT *
FROM Vendeurs VD
WHERE NOT EXISTS (
    SELECT 1
    FROM Base_N V
    WHERE V.vendeur_id = VD.vendeur_id
);
```

6.1.2 Sous-requêtes dans FROM (table dérivée)

Exemple : Sélectionner les produits dont le prix de vente unitaire est supérieur à la moyenne des prix de vente unitaire par sous-catégorie.

```sql
SELECT nom, prix_vente_unitaire, prix_moyen
FROM (
    SELECT nom,
           prix_vente_unitaire,
           AVG(prix_vente_unitaire) OVER(PARTITION BY sous_categorie_id) AS prix_moyen
    FROM Produits
) table_derivee
WHERE prix_vente_unitaire > prix_moyen;
```

Attention : Avec FROM il faut donner un nom à la sous-requête comme ici *table_derivee* ou tout simplement *t*.

6.1.3 Sous-requêtes dans SELECT (calculs scalaires)

Exemple : comparer le prix d'un produit au prix moyen

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
SELECT nom,
       prix_vente_unitaire,
       (SELECT AVG(prix_vente_unitaire) FROM Produits) AS prix_moyen
FROM Produits;
```

6.2 Les cas d'utilisation

6.2.1 Utiliser une fonction fenêtre dans un WHERE / HAVING

Exemple : Comparer les prix de vente au prix de vente moyen

```sql
SELECT nom, prix_vente_unitaire, prix_moyen
FROM (
    SELECT nom,
           prix_vente_unitaire,
           AVG(prix_vente_unitaire) OVER(PARTITION BY sous_categorie_id) AS prix_moyen
    FROM Produits
) t
WHERE prix_vente_unitaire > prix_moyen;
```

6.2.2 Faire un TOP N par groupe (TOP ne partitionne pas)

Exemple : TOP 3 des quantités vendues de produits par sous-categorie

```sql
SELECT *
FROM (
    SELECT P.nom,
           P.sous_categorie_id,
           SUM(B.quantite) AS total,
           RANK() OVER(PARTITION BY P.sous_categorie_id ORDER BY SUM(B.quantite) DESC) AS rang
    FROM Base_N B
    JOIN Produits P ON P.produit_id = B.produit_id
    GROUP BY P.nom, P.sous_categorie_id
) t
WHERE rang <= 3;
```

6.2.3 Filtrer sur un cumul, un rang ou un max partitionné

Exemple : Filtrer sur des cumul de quantités > 5000

```sql
SELECT *
FROM (
    SELECT produit_id,
           date_de_vente,
           SUM(quantite) OVER(PARTITION BY produit_id ORDER BY date_de_vente) AS cumul
    FROM Base_N
) t
WHERE cumul > 5000;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6.2.4 Faire une comparaison ligne à ligne

Exemple : Trouver les vendeurs qui dépassent le CA moyen de leur magasin

```
SELECT *
FROM (
    SELECT V.vendeur_id,
           V.magasin_id,
           SUM(B.quantite * P.prix_vente_unitaire) AS ca,
           AVG(SUM(B.quantite * P.prix_vente_unitaire))
               OVER(PARTITION BY V.magasin_id) AS ca_moyen
    FROM Base_N B
    JOIN Vendeurs V ON V.vendeur_id = B.vendeur_id
    JOIN Produits P ON P.produit_id = B.produit_id
    GROUP BY V.vendeur_id, V.magasin_id
) t
WHERE ca > ca_moyen;
```

6.2.5 Sous-requêtes imbriquées vs corrélées

Si une sous requête est indépendante de la requête principale : on dit qu'elle est **imbriquée**.

Dans le cas contraire, elle est **corrélée**.

Les sous-requêtes corrélées sont très pratiques pour des analyses "par catégorisation".

Exemple : produits plus chers que la moyenne de leur sous-catégorie

```
SELECT P.nom, P.sous_categorie_id, P.prix_vente_unitaire
FROM Produits P
WHERE P.prix_vente_unitaire >
(
    SELECT AVG(P2.prix_vente_unitaire)
    FROM Produits P2
    WHERE P2.sous_categorie_id = P.sous_categorie_id
);
```

C'est une alternative à la syntaxe : OVER (partition by…).

6.3 Opérateurs utiles

6.3.1 ANY

Vrai si la comparaison est vraie pour au moins une valeur du sous-ensemble.

Exemple : produits plus chers que au moins un autre produit de leur catégorie.

```
SELECT nom, prix_vente_unitaire
FROM Produits P1
WHERE prix_vente_unitaire > ANY (
    SELECT prix_vente_unitaire
    FROM Produits P2
    WHERE P2.sous_categorie_id = P1.sous_categorie_id
);
```

Ici l'interprétation : prix > le plus petit prix de la catégorie.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6.3.2 ALL

Vrai si la comparaison est vraie pour toutes les valeurs de la sous-requête.

Exemple : produits plus chers que tous les autres produits de leur catégorie.

```
SELECT nom, prix_vente_unitaire
FROM Produits P1
WHERE prix_vente_unitaire > ALL (
    SELECT prix_vente_unitaire
    FROM Produits P2
    WHERE P2.sous_categorie_id = P1.sous_categorie_id
      AND P2.produit_id <> P1.produit_id
);
```

Ici l'interprétation : prix = le prix max de la catégorie.

6.4 Vues

Une vue (VIEW) est un objet SQL qui stocke une requête. Elle ne contient aucune donnée, seulement une définition. Les données sont recalculées à chaque exécution de la vue.

On peut voir une vue comme :

- Une table virtuelle
- Un raccourci vers une requête complexe
- Un moyen de sécuriser l'accès à certaines colonnes
- Un outil pour simplifier un modèle de données

6.4.1 Intérêt des vues

1. Simplifier des requêtes complexes

Au lieu de réécrire une longue requête, on l'enveloppe dans une vue.

2. Sécuriser des données sensibles

On peut exposer uniquement certaines colonnes : ex : cacher emails, salaires, coûts internes…

3. Rendre un modèle plus lisible

On peut fusionner plusieurs tables logiques (JOIN + SELECT).

4. Réutiliser des agrégations ou calculs

Ex : le CA par produit, par vendeur, par magasin…

5. Centraliser la logique métier

Les vues garantissent la cohérence du calcul, même si plusieurs utilisateurs travaillent dessus.

6.4.2 Syntaxe et exemples

La syntaxe de base est la suivante :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
CREATE VIEW nom_vue AS
SELECT ...
FROM ...
```

Exemple : Vue du catalogue produit enrichi

```sql
CREATE VIEW v_produits_complet AS
SELECT P.produit_id,
       P.nom AS produit,
       P.couleur,
       P.poids,
       P.prix_vente_unitaire,
       S.sous_categorie_produit,
       C.categorie_produit
FROM Produits P
JOIN Sous_categories S ON S.sous_categorie_id = P.sous_categorie_id
JOIN Categories C ON C.categorie_id = S.categorie_id;
```

Exemple : Vue du CA par produit (toutes tables Base)

```sql
CREATE VIEW v_ca_produit AS
SELECT produit_id,
       SUM(quantite * prix_vente_unitaire) AS ca
FROM (
    SELECT B.produit_id, B.quantite, P.prix_vente_unitaire
    FROM Base_N B JOIN Produits P ON P.produit_id = B.produit_id
    UNION ALL
    SELECT B1.produit_id, B1.quantite, P.prix_vente_unitaire
    FROM Base_N_1 B1 JOIN Produits P ON P.produit_id = B1.produit_id
    UNION ALL
    SELECT B2.produit_id, B2.quantite, P.prix_vente_unitaire
    FROM Base_N_2 B2 JOIN Produits P ON P.produit_id = B2.produit_id
) t
GROUP BY produit_id;
```

6.5 CTE

Une CTE (Common Table Expression) est une requête temporaire nommée, définie à l'aide du mot-clé WITH, utilisable uniquement par l'instruction SQL qui la suit immédiatement.

Une CTE :

- N'est pas une table
- N'est pas une vue
- N'est pas stockée sur disque
- Améliore la lisibilité et la structuration des requêtes complexes

La syntaxe de base est la suivante :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
WITH nom_cte AS (
    SELECT ...
)
SELECT ...
FROM nom_cte;
```

La requête **doit suivre immédiatement** la CTE. Une **seule instruction** peut consommer la CTE.

6.5.1 CTE comme alternative à une sous-requête

On peut vouloir afficher le total des ventes par produit, puis filtrer les produits ayant vendu plus de 100 unités.

```sql
WITH ventes_par_produit AS (
    SELECT produit_id, SUM(quantité) AS total_ventes
    FROM Base_N
    GROUP BY produit_id
)
SELECT p.nom, v.total_ventes
FROM ventes_par_produit v
JOIN Produits p ON v.produit_id = p.produit_id
WHERE v.total_ventes > 100
ORDER BY v.total_ventes DESC;
```

Pourquoi une CTE est pertinente ici ?

- Elle évite une sous-requête imbriquée
- Elle sépare calcul et filtrage
- Elle facilite la lecture

6.5.2 CTE avec plusieurs niveaux (chaînage)

On peut vouloir calculer le chiffre d'affaires par produit, puis le chiffre d'affaires total par catégorie.

```sql
WITH ca_produit AS (
    SELECT
        b.produit_id,
        SUM(b.quantite * p.prix_vente_unitaire) AS ca
    FROM Base_N b
    JOIN Produits p ON b.produit_id = p.produit_id
    GROUP BY b.produit_id
),
ca_categorie AS (
    SELECT
        sc.categorie_id,
        SUM(cp.ca) AS ca_total
    FROM ca_produit cp
    JOIN Produits p ON cp.produit_id = p.produit_id
    JOIN Sous_categories sc ON p.sous_categorie_id = sc.sous_categorie_id
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
    GROUP BY sc.categorie_id
)
SELECT *
FROM ca_categorie;
```

Avantage clé : découpage logique par étapes métier.

6.5.3 CTE et fonctions analytiques (PARTITION BY)

On peut classer les produits par quantité vendue **à l'intérieur de chaque sous-catégorie**.

```
WITH ventes_produits AS (
    SELECT
        p.nom,
        p.sous_categorie_id,
        SUM(b.quantite) AS total_vendu
    FROM Base_N b
    JOIN Produits p ON b.produit_id = p.produit_id
    GROUP BY p.nom, p.sous_categorie_id
)
SELECT *,
       RANK() OVER (PARTITION BY sous_categorie_id ORDER BY total_vendu DESC) AS rang
FROM ventes_produits;
```

CTE indispensable ici :

- On ne peut pas référencer SUM(quantité) directement dans RANK()
- Séparation claire entre agrégation et analyse

6.5.4 CTE et TOP / OFFSET-FETCH

Top 3 produits les plus vendus par catégorie.

```
WITH ventes AS (
    SELECT
        p.nom,
        sc.categorie_id,
        SUM(b.quantite) AS total_vendu
    FROM Base_N b
    JOIN Produits p ON b.produit_id = p.produit_id
    JOIN Sous_categories sc ON p.sous_categorie_id = sc.sous_categorie_id
    GROUP BY p.nom, sc.categorie_id
),
classement AS (
    SELECT *,
           ROW_NUMBER() OVER (
               PARTITION BY categorie_id
               ORDER BY total_vendu DESC
           ) AS rn
    FROM ventes
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
)
SELECT *
FROM classement
WHERE rn <= 3;
```

6.5.5 Quand utiliser une CTE ?

C'est pertinent pour les usages suivants :

- Calcul en plusieurs étapes
- Fonctions analytiques
- Classements Top-N
- Logique métier lisible

En revanche, à éviter pour :

- Requêtes triviales
- Logique utilisée dans plusieurs requêtes → vue

Les CTE sont un outil de structuration, pas seulement de confort syntaxique. Elles deviennent indispensables dès que :

- La logique métier est complexe
- Les agrégations précèdent les analyses
- Le code doit rester lisible et maintenable

Comparaison avec les sous-requêtes et les vues :

Élément	CTE	Sous-requête	Vue
Lisibilité	Très	Peu	Très
Réutilisable	Non	Non	Oui
Temporaire	Oui	Oui	Non
Récursif	Oui	Non	Non

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6.6 Exercices

1. Afficher la liste des produits dont le prix de vente est supérieur au prix moyen de tous les produits.

nom	prix_vente_unitaire
E_bike1000	7500
E_bike1001	7800
E_bike1002	7700
Road_bike1000	3700
Road_bike1001	3900
Road_bike1002	3900
Road_bike1003	3900
Road_bike1004	3900
VTT1000	2650
VTT1001	2700
VTT1002	2700
VTT1003	2700
VTT1004	2700
VTT1005	2700
VTC1000	950
VTC1001	990
VTC1002	990
Batterie1000	960
Batterie1001	990

2. Trouver les vendeurs ayant vendu au moins une fois l'année N-1.

vendeur_id	nom	prenom
1	Bohr	Niels
2	Einstein	Albert
3	Noether	Emmy
4	Newton	Isaac
5	Curie	Marie
6	Schrodingher	Erwin

3. Trouver les produits qui ont un prix supérieur au prix moyen de leur sous catégorie.

nom	prix_vente_unitaire	sous_categorie_id
E_bike1001	7800	1
E_bike1002	7700	1
Road_bike1001	3900	2
Road_bike1002	3900	2
Road_bike1003	3900	2
Road_bike1004	3900	2
VTT1001	2700	3
VTT1002	2700	3
VTT1003	2700	3
VTT1004	2700	3

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	prix_vente_unitaire	sous_categorie_id
VTT1005	2700	3
VTC1001	990	4
VTC1002	990	4
Batterie1001	990	5
Cables_Ebike	406	6
Freins_Ebike	138	7
Selle_Ebike	150	8
Oesics_run	190	11
Oesics_run	190	11
Oesics_run	190	11
Oesics_run	190	11
Mike_shoes	185	11
Mike_shoes	185	11
Mike_shoes	185	11
Mike_shoes	185	11
SV_Homme	25	16

4. Trouver les vendeurs dont le CA dépasse la moyenne de celui de leur magasin.

vendeur_id	magasin_id	ca	ca_moyen
2	1	4902497	4786905
4	2	4736339	4590829
6	3	4720590	4654676

5. Trouver les produits qui n'ont jamais été vendus l'année N-1.

produit_id	nom
37	Oesics_run
41	Abidas_shoes
45	Mike_shoes
58	Tshirt
64	Short

6. Trouver les produits plus chers que TOUS les autres produits de leur sous-catégorie.

nom	prix_vente_unitaire	sous_categorie_id
E_bike1001	7800	1
Batterie1001	990	5
Cables_Ebike	406	6
Freins_Ebike	138	7
Selle_Ebike	150	8
Combinaison	108	9
Bike_shoes	85	10
Semelle_running	25	13
SV_Homme	25	16

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

7. Afficher les ventes dont le cumul dépasse 500 unités pour chaque produit dans l'ordre chronologique des ventes dans *Base_N*.

vente_id	produit_id	date_de_vente	quantite	cumul
V2025001057	18	2025-05-21	5	510
V2025001359	18	2025-05-21	3	510
V2025001309	18	2025-05-21	1	510
V2025001339	18	2025-05-21	1	510
V2025001014	18	2025-05-22	3	513
V2025001380	18	2025-05-23	2	516
V2025001068	18	2025-05-23	1	516
V2025001114	18	2025-05-24	1	534
V2025000932	18	2025-05-24	1	534
V2025001226	18	2025-05-24	1	534
V2025001407	18	2025-05-24	5	534
V2025001286	18	2025-05-24	5	534
V2025001039	18	2025-05-24	5	534
V2025001110	18	2025-05-25	2	538
V2025001139	18	2025-05-25	2	538
V2025001066	18	2025-05-26	4	545
V2025001320	18	2025-05-26	3	545
V2025001397	18	2025-05-28	5	550
V2025001296	18	2025-05-29	5	558
V2025001167	18	2025-05-29	3	558
V2025000962	18	2025-05-30	1	567
V2025001228	18	2025-05-30	3	567
V2025001041	18	2025-05-30	5	567
V2025001197	18	2025-05-31	1	568
V2025001117	18	2025-06-01	1	573
V2025001022	18	2025-06-01	1	573
V2025001213	18	2025-06-01	3	573
V2025001255	18	2025-06-02	2	582
V2025001276	18	2025-06-02	5	582
V2025001173	18	2025-06-02	2	582

8. Créer une vue affichant le CA par vendeur.

vendeur_id	ca
6	4720590
1	4671312
3	4445318
5	4588762
4	4736339
2	4902497

9. Afficher uniquement les produits dont le total vendu dépasse 7500 unités avec une CTE.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	total_quantite
Oesics_run_socks	7568
Tshirt	7501
SV_Homme	8043
SV_Femme	7547
Maillot_de_bain_Homme	24583
Maillot_de_bain_Homme	25163
Maillot_de_bain_Femme	24945
Maillot_de_bain_Femme	24570
Bonnet_de_bain	25005
Bonnet_de_bain	25955

10. Calculer le CA par catégorie à partir du CA par produit avec des CTE.

categorie_id	ca_total
1	9033660
2	4885583
3	4694115
4	2199149
5	2993784
6	4258527

11. Classer les produits par quantité vendue dans chaque sous-catégorie avec une CTE.

nom	sous_categorie_id	total_vendu	rang_sous_categorie
E_bike1002	1	163	1
E_bike1001	1	160	2
E_bike1000	1	156	3
Road_bike1002	2	155	1
Road_bike1001	2	146	2
Road_bike1000	2	138	3
Road_bike1004	2	136	4
Road_bike1003	2	128	5
VTT1000	3	171	1
VTT1003	3	161	2
VTT1002	3	156	3
VTT1001	3	154	4
VTT1005	3	116	5
VTT1004	3	84	6
VTC1002	4	160	1
VTC1000	4	137	2
VTC1001	4	94	3
Batterie1001	5	1562	1
Batterie1000	5	1406	2
Cables_VTT	6	1758	1
Cables_Road_bike	6	1642	2
Cables_Ebike	6	1549	3
Cables_VTC	6	1455	4
Freins_VTT	7	1590	1

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

nom	sous_categorie_id	total_vendu	rang_sous_categorie
Freins_Road_bike	7	1545	2
Freins_Ebike	7	1487	3
Freins_VTC	7	1429	4
Selle_Ebike	8	1484	1
Selle_VTC	8	1466	2
Selle_VTT	8	1430	3
Selle_Road_bike	8	1368	4
Combinaison	9	1531	1
Bike_shoes	10	2062	1
Abidas_shoes	11	8691	1
Oesics_run	11	8604	2
Mike_shoes	11	8308	3
Oesics_run_socks	12	14571	1
Abidas_run_socks	12	14358	2
Mike_run_socks	12	14054	3
Semelle_running	13	6535	1
Tshirt	14	56840	1
Short	15	28980	1
SV_Homme	16	8043	1
SV_Femme	16	7547	2
Maillot_de_bain_Homme	17	49746	1
Maillot_de_bain_Femme	17	49515	2
Bonnet_de_bain	18	50960	1
JE_watch	19	10673	1

12. Afficher les 3 produits les plus vendus par catégorie avec une CTE.

nom	categorie_id	total_vendu	rn
VTT1000	1	171	1
E_bike1002	1	163	2
VTT1003	1	161	3
Cables_VTT	2	1758	1
Cables_Road_bike	2	1642	2
Freins_VTT	2	1590	3
Abidas_shoes	3	8691	1
Oesics_run	3	8604	2
Mike_shoes	3	8308	3
Tshirt	4	56840	1
Short	4	28980	2
Oesics_run_socks	4	14571	3
Bonnet_de_bain	5	50960	1
Maillot_de_bain_Homme	5	49746	2
Maillot_de_bain_Femme	5	49515	3
JE_watch	6	10673	1

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6.7 Solutions

1. Afficher la liste des produits dont le prix de vente est supérieur au prix moyen de tous les produits.

```
SELECT nom, prix_vente_unitaire
FROM Produits
WHERE prix_vente_unitaire >
      (SELECT AVG(prix_vente_unitaire) FROM Produits);
```

2. Trouver les vendeurs ayant vendu au moins une fois l'année N-1.

```
SELECT vendeur_id, nom, prenom
FROM Vendeurs
WHERE vendeur_id IN (
    SELECT vendeur_id
    FROM Base_N
);
```

3. Trouver les produits qui ont un prix supérieur au prix moyen de leur sous catégorie.

```
SELECT nom, prix_vente_unitaire, sous_categorie_id
FROM Produits P1
WHERE prix_vente_unitaire >
      (SELECT AVG(prix_vente_unitaire)
       FROM Produits P2
       WHERE P2.sous_categorie_id = P1.sous_categorie_id);
```

4. Trouver les vendeurs dont le CA dépasse la moyenne de leur magasin.

```
SELECT *
FROM (
    SELECT V.vendeur_id,
           V.magasin_id,
           SUM(B.quantite * P.prix_vente_unitaire) AS ca,
           AVG(SUM(B.quantite * P.prix_vente_unitaire))
               OVER(PARTITION BY V.magasin_id) AS ca_moyen
    FROM Base_N B
    JOIN Vendeurs V ON V.vendeur_id = B.vendeur_id
    JOIN Produits P ON P.produit_id = B.produit_id
    GROUP BY V.vendeur_id, V.magasin_id
) t
WHERE ca > ca_moyen;
```

5. Trouver les vendeurs qui n'ont jamais vendu l'année N-1.

```
SELECT produit_id, nom
FROM Produits P
WHERE NOT EXISTS (
    SELECT 1
    FROM Base_N_1 B
    WHERE B.produit_id = P.produit_id
);
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

6. Trouver les produits plus chers que TOUS les autres produits de leur sous-catégorie.

```sql
SELECT nom, prix_vente_unitaire, sous_categorie_id
FROM Produits P1
WHERE prix_vente_unitaire > ALL (
    SELECT prix_vente_unitaire
    FROM Produits P2
    WHERE P2.sous_categorie_id = P1.sous_categorie_id
          AND P2.produit_id <> P1.produit_id
);
```

7. Afficher les ventes dont le cumul dépasse 500 unités pour chaque produit dans l'ordre chronologique des ventes dans *Base_N*.

```sql
SELECT *
FROM (
    SELECT vente_id,
      produit_id,
           date_de_vente,
           quantite,
           SUM(quantite) OVER(PARTITION BY produit_id ORDER BY date_de_vente) AS cumul
    FROM Base_N
) t
WHERE cumul > 500;
```

8. Créer une vue affichant le CA par vendeur.

```sql
CREATE VIEW ca_vendeur AS
SELECT vendeur_id,
       SUM(B.quantite * P.prix_vente_unitaire) AS ca
FROM Base_N B
JOIN Produits P ON P.produit_id = B.produit_id
GROUP BY vendeur_id;
```

9. Afficher uniquement les produits dont le total vendu dépasse 7500 unités avec une CTE.

```sql
WITH ventes_par_produit AS (
    SELECT produit_id, SUM(quantite) AS total_quantite
    FROM Base_N
    GROUP BY produit_id
)
SELECT p.nom, v.total_quantite
FROM ventes_par_produit v
JOIN Produits p ON v.produit_id = p.produit_id
WHERE v.total_quantite > 7500;
;
```

10. Calculer le CA par catégorie à partir du CA par produit avec des CTE.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
WITH ca_produit AS (
    SELECT
        b.produit_id,
        SUM(b.quantite * p.prix_vente_unitaire) AS ca
    FROM Base_N b
    JOIN Produits p ON b.produit_id = p.produit_id
    GROUP BY b.produit_id
),
ca_categorie AS (
    SELECT
        sc.categorie_id,
        SUM(cp.ca) AS ca_total
    FROM ca_produit cp
    JOIN Produits p ON cp.produit_id = p.produit_id
    JOIN Sous_categories sc ON p.sous_categorie_id = sc.sous_categorie_id
    GROUP BY sc.categorie_id
)
SELECT *
FROM ca_categorie;
```

11. Classer les produits par quantité vendue dans chaque sous-catégorie avec une CTE.

```sql
WITH ventes_produits AS (
    SELECT
        p.nom,
        p.sous_categorie_id,
        SUM(b.quantite) AS total_vendu
    FROM Base_N b
    JOIN Produits p ON b.produit_id = p.produit_id
    GROUP BY p.nom, p.sous_categorie_id
)
SELECT *,
       RANK() OVER (
           PARTITION BY sous_categorie_id
           ORDER BY total_vendu DESC
       ) AS rang_sous_categorie
FROM ventes_produits;
```

12. Afficher les 3 produits les plus vendus par catégorie avec une CTE.

```sql
WITH ventes AS (
    SELECT
        p.nom,
        sc.categorie_id,
        SUM(b.quantité) AS total_vendu
    FROM Base_N b
    JOIN Produits p ON b.produit_id = p.produit_id
    JOIN Sous_categories sc ON p.sous_categorie_id = sc.sous_categorie_id
    GROUP BY p.nom, sc.categorie_id
),
classement AS (
    SELECT *,
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
            ROW_NUMBER() OVER (
                PARTITION BY categorie_id
                ORDER BY total_vendu DESC
            ) AS rn
    FROM ventes
)
SELECT *
FROM classement
WHERE rn <= 3;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

7. DDL : Création et Structure (CREATE TABLE, ALTER, DROP)

L'objectif est de créer et gérer la structure des objets SQL (bases, tables…)

7.1 Manipulation de base de données

7.1.1 CREATE DATABASE

Les SGBD permettent de créer une base par l'interface graphique.

Il convient néanmoins de connaître la commande SQL à utiliser.

La syntaxe de base est la suivante :

```
CREATE DATABASE nouvelle_base
```

7.1.2 ALTER DATABASE

La commande ALTER DATABASE permet de modifier la structure et les propriétés d'une base de données existante, sans toucher directement aux données qu'elle contient. Contrairement à CREATE DATABASE ou DROP DATABASE, ALTER DATABASE est utilisée pour changer les paramètres, le nom, l'emplacement des fichiers ou le mode de récupération d'une base déjà créée.

La syntaxe de base est la suivante :

```
ALTER DATABASE nouvelle_base
modification
```

On peut par exemple renommer une base de données :

```
ALTER DATABASE nouvelle_base
MODIFY NAME = nouveau_nom_base
```

Ou encore changer le mode d'accès :

```
ALTER DATABASE nouveau_nom_base
SET READ_ONLY; -- Lecture seule

ALTER DATABASE nouveau_nom_base
SET READ_WRITE; -- lecture écriture

ALTER DATABASE nouveau_nom_base
SET SINGLE_USER WITH ROLLBACK IMMEDIATE; -- une seule connexion avec déconnexion
                                         -- des utilisateurs pour appliquer le changement

ALTER DATABASE nouveau_nom_base
SET MULTI_USER; -- plusieurs utilisateurs
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

7.1.3 DROP DATABASE

Cette commande permet de supprimer totalement une base de données et tout ce qu'elle contient…

La syntaxe de base est la suivante :

```
DROP DATABASE nouveau_nom_base
```

Attention : cette commande supprime absolument tout. Ne pas hésiter à faire une sauvegarde avant de supprimer.

7.2 Manipulation de tables

7.2.1 CREATE TABLE

Cette commande, comme son nom l'indique, permet de créer une table.

La syntaxe de base est la suivante :

```
CREATE TABLE nom_table
(
colonne1 type_donnees,
colonne2 type_donnees,
colonne3 type_donnees,
colonne4 type_donnees
)
```

Pour chaque colonne, il faudra ainsi définir le type de données (INT, DATE, VARCHAR…).

Il est également possible de définir des options telles que :

- PRIMARY KEY : Pour indiquer que la colonne est une clé primaire
- FOREIGN KEY REFERENCES table(cle_primaire) : Pour indiquer que la colonne est une clé étrangère et à quelle table(colonne) elle est reliée
- NOT NULL : Pour empêcher une valeur nulle dans la colonne
- UNIQUE : Pour empêcher les doublons
- CHECK : Pour appliquer une règle métier simple
- DEFAULT : Pour attribuer une valeur par défaut lors de l'ajout d'une ligne si rien n'est indiqué à ce moment là

Par exemple, si on voulait créer une table Produit_bis, le code pourrait être :

```
CREATE TABLE Produits_bis
(
    id INT PRIMARY KEY NOT NULL,
    nom VARCHAR(50),
    couleur VARCHAR(50),
    poids SMALLINT,
    sous_categorie_id SMALLINT FOREIGN KEY REFERENCES Sous_categories(sous_categorie_id),
    frais_generaux INT,
    couts_fixes INT,
    cout_unitaire DECIMAL(10,2),
    prix_vente_unitaire DECIMAL(10,2),
    ca_objectif INT
    )
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

- id : c'est la clé primaire de type INT (nombre entier) sans valeur nulle
- nom : nom du produit en VARCHAR(50) (texte de 50 caractères maximum)
- couleur : couleur du produit en VARCHAR(50)
- sous_categorie_id : numéro de sous catégorie en SMALLINT (petit nombre entier) qui est aussi une clé étrangère reliée à la colonne Sous_categorie_id de la table Sous_categories
- frais_generaux : Frais généraux en DECIMAL (nombre décimal)
- couts_fixes : coûts fixes en INT
- cout_unitaire : coût unitaire en DECIMAL (nombre décimal)
- prix_vente_unitaire : prix de vente unitaire en DECIMAL (nombre décimal)
- ca_objectif : Objectif de CA en INT

7.2.2 Liste des principaux types de données sur PostgreSQL

- Nombres entiers

Type de données	Intervalle	Poids
bigint	-2^63 à 2^63-1	8 octets
integer	-2^31 à 2^31-1	4 octets
smallint	-32768 à 32767	2 octets

- Boolean

Utilisé pour stocker des valeurs booléennes. Les valeurs TRUE de chaîne et FALSE peuvent être converties en valeurs de bits : TRUE sont converties en 1, et FALSE sont converties en 0.

- Decimal et numeric

Ces deux types sont interchangeables sur SQL SERVER.

La syntaxe est decimal(p,s) (ou numeric(p,s)).

- p : Nombre total maximal de chiffres décimaux à stocker. Ce nombre **inclut le côté à gauche et le côté à droite** de la virgule. La précision doit être une valeur comprise entre 1 et la précision maximale de 38. La précision par défaut est 18.
- s : Nombre de chiffres décimaux stockés à droite de la virgule. Ce nombre est soustrait de p afin de déterminer le nombre maximal de chiffres à gauche de la virgule décimale. La mise à l'échelle doit être une valeur comprise entre 0 et p et peut uniquement être spécifiée si la précision est indiquée.

Précision	Poids
1-9	5 octets
10-19	9 octets
20-28	13 octets
29-38	17 octets

- Money

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Ce type de données représente une valeur monétaire ou une devise.

Type de données	Intervalle	Poids
money	-922 337 203 685 477,5808 à 922 337 203 685 477,5807	8 octets

- Float et real

Ces types de données approximatives sont à utiliser avec des données numériques à virgule flottante. Les données à virgule flottante sont approximatives ; il n'est donc pas possible de représenter précisément toutes les valeurs de ce type de données.

Les types de données numériques approximatifs ne stockent pas les valeurs exactes spécifiées pour de nombreux nombres ; ils stockent une approximation étroite de la valeur. Pour certaines applications, la petite différence entre la valeur spécifiée et l'approximation stockée n'est pas pertinente. Pour d'autres cependant, la différence est importante. En raison de la nature approximative des types de données float et réels, ne pas utiliser ces types de données lorsque le comportement numérique exact est requis. Les exemples qui nécessitent des valeurs numériques précises sont des données financières ou commerciales, des opérations impliquant des arrondis ou des vérifications d'égalité. Dans ces cas, utilisez les types de données entier, décimal, numérique, monétaire ou smallmoney.

Éviter d'utiliser des colonnes flottantes ou réelles dans les conditions de recherche de clause WHERE, en particulier les = et <> les opérateurs. Il est préférable de limiter les colonnes flottantes et réelles à des comparaisons ou > à < des comparaisons.

Pour le type float, la syntaxe est float(n) où n désigne la précision (nombre de bits pour stocker la mantisse du nombre). Si n est précisé, sa valeur doit être comprise entre 1 et 53. La valeur par défaut de n est 53.

Valeur n	Précision	Poids
1-24	7 chiffres	4 octets
25-53	15 chiffres	8 octets

Type de données	Intervalle	Poids
float	- 1,79E+308 à -2,23E-308, 0 et 2,23E-308 à 1,79E+308	Dépend de n, voir ci-dessus
real	- 3,40E + 38 à -1,18E - 38, 0 et 1,18E - 38 à 3,40E + 38	4 octets

- Date et heure

Type de données	Exemple	commentaire
time	09:52:45.1234567	Affiche l'heure
date	2025-12-25	Affiche seulement la date
timestamp	2025-12-25 09:52:45.1234567	Affiche la date et l'heure
timestamp with time zone	2025-12-25 09:52:45.1234567 +08:00	Affiche la date et l'heure avec une prise en charge du fuseau horaire

- Chaînes de caractères

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Type	Longueur	Unicode (UTF-16)	Stockage	Usage principal
CHAR(n)	Fixe	Non pris en charge	1 octet × n	Codes courts fixes
VARCHAR(n)	Variable	Non pris en charge	1 octet × longueur réelle	Texte standard

Il est possible de consulter la documentation officielle de PostgreSQL sur ce lien.

7.2.3 Tables temporaires

Une table temporaire est une table créée temporairement, stockée dans la base système tempdb, et supprimée automatiquement à la fin de la session.

Elle sert à :

- Stocker des résultats intermédiaires
- Simplifier des requêtes complexes
- Améliorer la lisibilité et parfois les performances
- Visible uniquement dans la session courante
- Supprimée à la fermeture de la connexion

La syntaxe de base est la suivante :

```
CREATE TEMPORARY TABLE nom_table -- TEMP marche aussi
(
colonne1 type_donnees,
colonne2 type_donnees,
colonne3 type_donnees,
colonne4 type_donnees
)
```

On peut créer la table suivante par exemple :

```
CREATE TEMP TABLE Ventes (
    Produit VARCHAR(50),
    Montant DECIMAL(10,2)
);
```

Elle sera supprimée automatiquement à la déconnexion. Sinon il est aussi possible de la supprimer manuellement avec DROP TABLE qui est expliqué plus bas. De manière générale, les commandes qui s'appliquent aux tables classiques s'appliquent aussi aux tables temporaires.

7.2.4 ALTER TABLE

Cette commande permet de modifier une table existante. Il est ainsi possible d'ajouter une colonne, de supprimer une colonne ou de modifier une colonne existante.

La syntaxe de base est la suivante :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
ALTER TABLE nom_table
modification
```

On peut par exemple ajouter une colonne stock dans la table Produits_bis de type smallint :

```
ALTER TABLE Produits_bis
ADD stock SMALLINT
```

On peut aussi supprimer une colonne, par exemple la colonne stock que l'on vient de créer :

```
ALTER TABLE Produits_bis
DROP COLUMN stock
```

On peut également modifier le type d'une colonne :

```
ALTER TABLE Produits_bis
ALTER COLUMN prix_vente_unitaire DECIMAL(10,4)
```

7.2.5 DROP TABLE

Cette commande permet de supprimer définitivement une table d'une base de données.

La syntaxe de base est la suivante :

```
DROP TABLE nom_table
```

On peut par exemple supprimer la table Produits_bis que l'on a créée plus haut :

```
DROP TABLE Produits_bis
```

7.2.6 TRUNCATE TABLE

Cette commande permet de supprimer toutes les données d'une table sans supprimer la table elle-même. Elle purge la table.

La syntaxe de base est la suivante :

```
TRUNCATE TABLE nom_table
```

On peut par exemple effacer toutes les données de la table Base_N_2 :

```
TRUNCATE TABLE Base_N_2
```

7.3 Les contraintes (CONSTRAINT) sur PostgreSQL

Une contrainte est une règle définie au niveau de la base de données qui garantit l'intégrité des données.

Les principaux types de contraintes, nous les avons vus plus haut sont :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Contrainte	Rôle
PRIMARY KEY	Identifie une ligne
FOREIGN KEY	Lien entre tables
UNIQUE	Valeur unique
NOT NULL	Valeur obligatoire
CHECK	Règle métier
DEFAULT	Valeur par défaut

Il y a deux méthodes pour créer une contrainte :

- À la création :

```
CREATE TABLE Exemple (
    id INT CONSTRAINT PK_Exemple PRIMARY KEY, -- on nomme la contrainte PK_Exemple
    -- et c'est une clé primaire
    valeur INT CONSTRAINT CK_Valeur CHECK (valeur > 0) -- on nomme la contrainte CK_Valeur
    --et c'est une contrainte qui impose une valeur positive
);
```

- Après coup :

```
ALTER TABLE Exemple
ADD CONSTRAINT CK_Valeur CHECK (valeur > 0);
```

On peut supprimer une contrainte de la manière suivante :

```
ALTER TABLE Base_N
DROP CONSTRAINT CK_BaseN_Discount;
```

7.4 Manipulation d'index

7.4.1 CREATE INDEX

Un index est une structure de données associée à une table qui permet à SQL Server de retrouver plus rapidement les lignes sans parcourir toute la table.

Comparable à l'index d'un livre :

- Sans index → lecture page par page
- Avec index → accès direct à l'information

Les index permettent de :

- Accélérer les requêtes SELECT
- Améliorer les performances des clauses :
  - WHERE
  - JOIN

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

  - ORDER BY
  - GROUP BY

- Optimiser l'accès aux clés primaires et étrangères

En contrepartie :

- Les index ralentissent les opérations INSERT, UPDATE, DELETE
- Ils occupent de l'espace disque

Pour en savoir plus sur les index sur SQL SERVER, vous pouvez consulter le lien suivant.

Pour créer un index sur une colonne d'une table, La syntaxe de base est la suivante :

```
CREATE INDEX nom_index ON nom_table(nom_colonne)
```

On peut par exemple vouloir indexer la colonne *date_de_vente* de la table *Base_N* souvent interrogée :

```
CREATE INDEX index_exemple ON Base_N(date_de_vente)
```

Il est également possible d'indexer plusieurs colonnes. On peut vouloir indexer également les quantités avec la date :

```
CREATE INDEX index_exemple2 ON Base_N(date_de_vente, quantite)
```

7.4.2 DROP INDEX

On peut également supprimer un index avec la commande DROP.

La syntaxe de base est la suivante :

```
DROP INDEX nom_index ON nom_table
```

C'est ainsi que l'on supprimerait le dernier index créé :

```
DROP INDEX index_exemple2 ON Base_N
```

7.5 Exercices

1. Créer une table *Promotions* avec :

- promotion_id (clé primaire)
- produit_id
- taux_reduction
- date_debut
- date_fin

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

2. Créer une table *Stocks* gérant le stock par produit et par magasin :

- produit_id
- magasin_id ces 2 colonnes constituent la clé primaire composite
- quantite

3. Ajouter une contrainte à la table *Promotions* pour limiter le taux de réduction entre 0 et 70 %.
4. Ajouter une contrainte à la table *Promotions* pour avoir date_fin doit être postérieure à date_debut.
5. Ajouter une colonne date_creation à Clients.
6. Passer discount de *Base_N_2* en DECIMAL(5,2).
7. Supprimer la table *Promotions*.
8. Vider la table *Stocks*.
9. Créer un index sur date_de_vente dans *Base_N*.
10. Créer un index sur produit_id et vendeur_id dans la même table.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

7.6 Solutions

1. Créer une table *Promotions* avec :

- promotion_id (clé primaire)
- produit_id
- taux_reduction
- date_debut
- date_fin

```
CREATE TABLE Promotions (
    promotion_id INT IDENTITY PRIMARY KEY,
    produit_id SMALLINT NOT NULL,
    taux_reduction DECIMAL(5,2) NOT NULL,
    date_debut DATE NOT NULL,
    date_fin DATE NOT NULL
);
```

2. Créer une table *Stocks* gérant le stock par produit et par magasin :

- produit_id
- magasin_id ces 2 colonnes constituent la clé primaire composite
- quantite

```
CREATE TABLE Stocks (
    produit_id SMALLINT NOT NULL,
    magasin_id SMALLINT NOT NULL,
    quantite INT NOT NULL,

    CONSTRAINT PK_Stocks PRIMARY KEY (produit_id, magasin_id),
    CONSTRAINT FK_Stocks_Produit FOREIGN KEY (produit_id)
        REFERENCES Produits(produit_id),
    CONSTRAINT FK_Stocks_Magasin FOREIGN KEY (magasin_id)
        REFERENCES Magasins(magasin_id)
);
```

3. Ajouter une contrainte à la table *Promotions* pour limiter le taux de réduction entre 0 et 70 %.

```
ALTER TABLE Promotions
ADD CONSTRAINT CK_Promotions_Taux
CHECK (taux_reduction BETWEEN 0 AND 0.70);
```

4. Ajouter une contrainte à la table *Promotions* pour avoir date_fin doit être postérieure à date_debut.

```
ALTER TABLE Promotions
ADD CONSTRAINT CK_Promotions_Dates
CHECK (date_fin > date_debut);
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

5. Ajouter une colonne date_creation à Clients.

```
ALTER TABLE Clients
ADD date_creation DATETIME NOT NULL
    CONSTRAINT DF_Clients_DateCreation DEFAULT GETDATE();
```

6. Passer discount de *Base_N_2* en DECIMAL(5,2).

```
ALTER TABLE Clients
ADD date_creation DATETIME NOT NULL
    CONSTRAINT DF_Clients_DateCreation DEFAULT GETDATE();
```

7. Supprimer la table *Promotions*.

```
DROP TABLE Promotions;
```

8. Vider la table *Stocks*.

```
TRUNCATE TABLE Stocks;
```

9. Créer un index sur date_de_vente dans *Base_N*.

```
CREATE INDEX IX_Base_N_DateVente
ON Base_N(date_de_vente);
```

10. Créer un index sur produit_id et vendeur_id dans la même table.

```
CREATE INDEX IX_Base_N_Produit_Vendeur
ON Base_N(produit_id, vendeur_id);
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

8. DML : Ajouter / modifier / supprimer des données (INSERT, UPDATE, DELETE)

8.1 INSERT INTO

L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO.

Il est possible d'insérer des données sans préciser les colonnes concernées, à condition de respecter l'ordre stricte.

La syntaxe de base est la suivante :

```
INSERT INTO nom_table
VALUES ('valeur 1', 'valeur 2',...)
```

Ou alors on peut préciser les colonnes concernées (dans le cas où l'on n'ajouterait pas de données pour chaque colonne).

La syntaxe de base est alors la suivante :

```
INSERT INTO nom_table
(nom_colonne_1, nom_colonne_2, ...)
VALUES ('valeur 1', 'valeur 2',...)
```

On peut par exemple ajouter un nouveau vendeur à notre table de *Vendeurs* :

```
INSERT INTO Vendeurs
(vendeur_id, prenom, nom, magasin_id, vend_objectif)
VALUES
(7,'David', 'Hilbert',3,4000000)
```

On remarquera que la colonne mail contiendra alors la valeur NULL.

Si on affiche maintenant tous les vendeurs, on retrouve notre nouveau vendeur :

```
SELECT * FROM Vendeurs
```

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
1	Niels	Bohr	niels.bohr@jaimeexcel.com	1	3800000
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
3	Emmy	Noether	emmy.noether@jaimeexcel.com	2	4030000
4	Isaac	Newton	isaac.newton@jaimeexcel.com	2	4050000
5	Marie	Curie	marie.curie@jaimeexcel.com	3	3900000
6	Erwin	Schrodingher	erwin.schrodingher@jaimeexcel.com	3	3800000
7	David	Hilbert	NA	3	4000000

On peut évidemment ajouter plusieurs lignes à la fois :

```
INSERT INTO Vendeurs
(vendeur_id, prenom, nom, mail, magasin_id, vend_objectif)
VALUES
(8,'Claude', 'Shannon', 'claude.shannon@jaimeexcel.com',2,4000000),
(9,'Andrei', 'Kolmogorov','andrei.kolmogorov@jaimeexcel.com',1,4000000)
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Et on obtient alors le résultat suivant :

vendeur_id	prenom	nom	mail	magasin_id	vend_objectif
1	Niels	Bohr	niels.bohr@jaimeexcel.com	1	3800000
2	Albert	Einstein	albert.einstein@jaimeexcel.com	1	4200000
3	Emmy	Noether	emmy.noether@jaimeexcel.com	2	4030000
4	Isaac	Newton	isaac.newton@jaimeexcel.com	2	4050000
5	Marie	Curie	marie.curie@jaimeexcel.com	3	3900000
6	Erwin	Schrodingher	erwin.schrodingher@jaimeexcel.com	3	3800000
7	David	Hilbert	NA	3	4000000
8	Claude	Shannon	claude.shannon@jaimeexcel.com	2	4000000
9	Andrei	Kolmogorov	andrei.kolmogorov@jaimeexcel.com	1	4000000

8.2 UPDATE

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications.

La syntaxe de base est la suivante :

```
UPDATE nom_table
SET nom_colonne_1 = 'valeur nouvelle'
WHERE condition
```

En cas de plusieurs modifications :

```
UPDATE nom_table
SET nom_colonne_1 = 'valeur nouvelle', nom_colonne_2 = 'valeur nouvelle2',...
WHERE condition
```

On peut par exemple mettre à jour les objectifs de vente de nos 3 nouveaux vendeurs à 5000 000 :

```
UPDATE Vendeurs
SET vend_objectif = 5000000
WHERE vendeur_id IN (7,8,9)
```

8.3 DELETE

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associé à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

La syntaxe de base est la suivante :

```
DELETE FROM nom_table
WHERE condition
```

Attention : si rien n'es précisé dans WHERE, toutes les lignes sont supprimées.

On peut par exemple supprimer les 3 derniers vendeurs ajoutés :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
DELETE FROM Vendeurs
WHERE vendeur_id IN (7,8,9)
```

8.4 MERGE

La commande MERGE permet de synchroniser deux tables en une seule instruction SQL.

Elle sert à :

- Insérer des lignes manquantes
- Mettre à jour des lignes existantes
- Supprimer des lignes devenues obsolètes

Elle est très utilisée dans :

- L'intégration de fichiers CSV
- Les mises à jour de référentiels
- Les flux ETL
- Les data warehouses

Le principe général est le suivant, MERGE compare : * Une table cible (**TARGET**) * Une source de données (**SOURCE**)

La syntaxe de base est la suivante :

```
MERGE table_cible AS T -- Désigne la table cible, celle qui sera modifiée
USING source AS S -- Désigne la source des données (table, vue, requête).
ON condition_de_jointure -- Condition de correspondance entre la source et la cible (clé métier).

WHEN MATCHED THEN -- Cas où une ligne existe dans la source et dans la cible.
    UPDATE SET ... -- Utilisé pour mettre à jour.

WHEN NOT MATCHED BY TARGET THEN -- Cas où la ligne existe dans la source mais pas dans la cible.
    INSERT (...) -- Utilisé pour insérer.

WHEN NOT MATCHED BY SOURCE THEN -- Cas où la ligne existe dans la cible mais plus dans la source.
    DELETE; -- Utilisé pour supprimer ou désactiver.
```

C'est ainsi que l'on pourrait par exemple mettre à jour les vendeurs depuis un fichier RH :

```
MERGE Vendeurs AS T
USING Vendeurs_import AS S
ON T.vendeur_id = S.vendeur_id

WHEN MATCHED THEN
    UPDATE SET
        T.prenom = S.prenom,
        T.nom = S.nom,
        T.mail = S.mail,
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
        T.magasin_id = S.magasin_id,
        T.vend_objectif = S.vend_objectif

WHEN NOT MATCHED BY TARGET THEN
    INSERT (vendeur_id, prenom, nom, mail, magasin_id, vend_objectif)
    VALUES (S.vendeur_id, S.prenom, S.nom, S.mail, S.magasin_id, S.vend_objectif);
```

8.5 Exercices

1. Insérer une nouvelle vente dans *Base_N*.
2. Insérer plusieurs ventes en une seule requête.
3. Insérer dans *Base_N_2* toutes les ventes de *Base_N*.
4. Augmenter de 5 % le prix de vente de tous les produits rouges.
5. Modifier le discount :

- 10 % si quantité 5
- 5 % sinon.

6. Supprimer les ventes sans client.
7. Convertir les discounts stockés en pourcentage texte (‘10%’) en décimal.
8. Supprimer les ventes avec discount nul et quantité = 1.
9. Décaler toutes les ventes de janvier 2024 d’un jour (*Base_N_1*).
10. Créer une table d’agrégation *ventes_agg* avec 3 colonnes :

- produit_id
- vendeur_id
- quantite (pour la somme des quantités)

11. Mettre à jour la table *ventes_agg* à partir de Base_N.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

8.6 Solutions

1. Insérer une nouvelle vente dans *Base_N*.

```
INSERT INTO Base_N (
    vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente
)
VALUES (
    'V1001', 12, 3, 5, 2, 0.10, '2024-03-15'
);
```

2. Insérer plusieurs ventes en une seule requête.

```
INSERT INTO Base_N (vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente)
VALUES
('V1002', 15, 2, 3, 1, 0.00, '2024-03-16'),
('V1003', 18, 4, 7, 2, 0.05, '2024-03-16');
```

3. Insérer dans *Base_N_2* toutes les ventes de *Base_N*.

```
INSERT INTO Base_N_2
SELECT *
FROM Base_N;
```

4. Augmenter de 5 % le prix de vente de tous les produits rouges.

```
UPDATE Produits
SET prix_vente_unitaire = prix_vente_unitaire * 1.05
WHERE couleur = 'Rouge';
```

5. Modifier le discount :

- 10 % si quantité 5
- 5 % sinon.

```
UPDATE Base_N
SET discount = CASE
    WHEN quantite >= 5 THEN 0.10
    ELSE 0.05
END;
```

6. Supprimer les ventes sans client.

```
DELETE FROM Base_N
WHERE client_id IS NULL;
```

7. Convertir les discounts stockés en pourcentage texte (‘10%’) en décimal.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
UPDATE Base_N
SET discount = CAST(REPLACE(discount, '%', '') AS DECIMAL(5,2)) / 100
WHERE discount LIKE '%[%]';
```

8. Supprimer les ventes avec discount nul et quantité = 1.

```sql
DELETE FROM Base_N
WHERE discount = 0
  AND quantite = 1;
```

9. Décaler toutes les ventes de janvier 2024 d'un jour (*Base_N_1*).

```sql
UPDATE Base_N
SET date_de_vente = DATEADD(DAY, 1, date_de_vente)
WHERE date_de_vente BETWEEN '2024-01-01' AND '2024-01-31';
```

10. Créer une table d'agrégation *ventes_agg* avec 3 colonnes :

- produit_id
- vendeur_id
- quantite (pour la somme des quantités)

```sql
CREATE TABLE ventes_agg (
    produit_id SMALLINT NOT NULL,
    vendeur_id SMALLINT NOT NULL,
    quantite INT NOT NULL,

    CONSTRAINT PK_ventes_agg
        PRIMARY KEY (produit_id, vendeur_id),

    CONSTRAINT FK_ventes_agg_Produit
        FOREIGN KEY (produit_id)
        REFERENCES Produits(produit_id),

    CONSTRAINT FK_ventes_agg_Vendeur
        FOREIGN KEY (vendeur_id)
        REFERENCES Vendeurs(vendeur_id)
);
```

11. Mettre à jour la table *ventes_agg* à partir de Base_N

```sql
MERGE ventes_agg AS cible
USING (
    SELECT produit_id, vendeur_id, SUM(quantite) AS qte
    FROM Base_N
    GROUP BY produit_id, vendeur_id
) AS source
ON cible.produit_id = source.produit_id
   AND cible.vendeur_id = source.vendeur_id
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
WHEN MATCHED THEN
    UPDATE SET quantite = source.qte

WHEN NOT MATCHED THEN
    INSERT (produit_id, vendeur_id, quantite)
    VALUES (source.produit_id, source.vendeur_id, source.qte);
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

9. TCL (Transaction Control Language) : Transactions

Une transaction est un ensemble d'instructions SQL qui doit être traité comme une seule unité logique.

Tout est validé, ou rien ne l'est.

Par exemple , une vente :

- Enregistrer la vente
- Mettre à jour les stocks
- Calculer le CA

Si une seule étape échoue → on annule tout.

Rappel des propriétés ACID, en effet toute transaction respecte 4 propriétés fondamentales :

Propriété	Description
Atomicité	Tout ou rien
Cohérence	La base reste valide
Isolation	Les transactions ne se perturbent pas
Durabilité	Une fois validé, c'est définitif

Les commandes TCL sur PostgreSQL :

Commande	Rôle
BEGIN TRANSACTION	Démarre une transaction
COMMIT	Valide les changements
ROLLBACK	Annule les changements
SAVEPOINT	Point de sauvegarde SQL Server

9.1 TRANSACTION et COMMIT

La syntaxe de base est la suivante :

```sql
BEGIN TRANSACTION;

INSERT INTO Base_N (...);
UPDATE Produits (...);

COMMIT;
```

On contrôle ainsi quand valider.

9.2 ROLLBACK

ROLLBACK annule toutes les modifications depuis le début de la transaction.

```sql
BEGIN TRANSACTION;

DELETE FROM Base_N;

ROLLBACK;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

Aucune ligne supprimée.

Par ailleurs la commande ROLLBACK est obligatoire sur PostgreSQL si une erreur survient dans une transaction.

Aucune commande suivante ne peut s'exécuter tant qu'on ne fait pas ROLLBACK.

Contrairement à SQL Server par exemple, PostgreSQL n'accepte pas de continuer après une erreur sans gestion explicite.

9.3 Gestion des erreurs

En PostgreSQL, **si une commande échoue dans une transaction :**

La transaction **passe immédiatement en état d'erreur.**

Aucune commande suivante ne peut s'exécuter.

Pour continuer malgré l'erreur, il faut utiliser un bloc PL/pgSQL avec EXCEPTION.

La syntaxe de base st la suivante :

```
DO $$
BEGIN
    -- commandes à risque
EXCEPTION
    WHEN condition1 THEN
        -- actions si condition1
    WHEN condition2 THEN
        -- actions si condition2
    WHEN OTHERS THEN
        -- attrape toutes les autres erreurs
END;
$$;
```

Les conditions possibles sont :

- unique_violation → violation d'une contrainte d'unicité
- foreign_key_violation → violation de clé étrangère
- check_violation → violation d'une contrainte CHECK
- division_by_zero, numeric_value_out_of_range, etc.
- OTHERS → attrape toutes les erreurs restantes

Voyons un exemple :

```
BEGIN;

INSERT INTO Base_N(vente_id, "quantité") VALUES ('T006', 5);

DO $$
BEGIN
    INSERT INTO Base_N(vente_id, "quantité") VALUES ('T007', -1);
EXCEPTION
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
    WHEN check_violation THEN
        RAISE NOTICE 'Quantité invalide, insertion ignorée';
    WHEN OTHERS THEN
        RAISE NOTICE 'Autre erreur, transaction continue';
END;
$$;

INSERT INTO Base_N(vente_id, "quantité") VALUES ('T008', 10);

COMMIT;
```

Résultat :

- 'T006' inséré
- 'T007' échoue → message spécifique si violation CHECK, sinon OTHERS
- 'T008' inséré malgré l'erreur

9.4 Exercices

1. Insérer une vente puis valider la transaction.
2. Insérer une vente puis annuler l'opération.
3. Ecrire une transaction qui permet de faire les choses suivantes :

- Insérer une vente 'TCL106' avec quantité 7.
- Tenter d'insérer une vente 'TCL107' avec quantité -3.
- Si une erreur survient, afficher un message et continuer la transaction.
- Insérer ensuite une vente 'TCL108' avec quantité 12.
- Valider la transaction.

4. Ecrire une transaction qui permet de faire les choses suivantes :

- Mettre à jour la quantité de la vente 'TCL101' pour passer de sa valeur actuelle à -3.
- Si une contrainte CHECK est violée (quantité négative), afficher un message et continuer la transaction.
- Ensuite, mettre à jour 'TCL106' pour augmenter la quantité de 5.
- Valider la transaction.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

9.5 Solutions

1. Insérer une vente puis valider la transaction.

```
BEGIN TRANSACTION;

INSERT INTO Base_N (
    vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente
)
VALUES (
    'TCL001', 10, 2, 4, 1, 0.05, CURRENT_DATE
);

COMMIT;
```

2. Insérer une vente puis annuler l'opération.

```
BEGIN TRANSACTION;

INSERT INTO Base_N (vente_id, produit_id, quantite)
VALUES ('TCL002', 3, 2);

ROLLBACK;
```

3. Ecrire une transaction qui permet de faire les choses suivantes :

- Insérer une vente 'TCL106' avec quantité 7.
- Tenter d'insérer une vente 'TCL107' avec quantité -3.
- Si une erreur survient, afficher un message et continuer la transaction.
- Insérer ensuite une vente 'TCL108' avec quantité 12.
- Valider la transaction.

```
BEGIN;

-- vente correcte
INSERT INTO Base_N(vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente)
VALUES ('TCL106', 1, 1, 2, 7, 0.05, CURRENT_DATE);

-- bloc PL/pgSQL pour gérer l'erreur
DO $$
BEGIN
    INSERT INTO Base_N(vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente)
    VALUES ('TCL107', 2, 2, 3, -3, 0.1, CURRENT_DATE);
EXCEPTION
    WHEN check_violation THEN
        RAISE NOTICE 'Quantité invalide pour TCL107, insertion ignorée';
    WHEN OTHERS THEN
        RAISE NOTICE 'Autre erreur pour TCL107, transaction continue';
END;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
$$;

-- vente suivante
INSERT INTO Base_N(vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente)
VALUES ('TCL108', 3, 2, 1, 12, 0.2, CURRENT_DATE);

COMMIT;
```

4. Ecrire une transaction qui permet de faire les choses suivantes :

- Mettre à jour la quantité de la vente 'TCL101' pour passer de sa valeur actuelle à -3.
- Si une contrainte CHECK est violée (quantité négative), afficher un message et continuer la transaction.
- Ensuite, mettre à jour 'TCL106' pour augmenter la quantité de 5.
- Valider la transaction.

```
BEGIN;

DO $$
BEGIN
    UPDATE Base_N
    SET quantite = -3
    WHERE vente_id = 'TCL101';
EXCEPTION
    WHEN check_violation THEN
        RAISE NOTICE 'Erreur : la mise à jour de TCL101 viole la contrainte CHECK, changement ignoré';
    WHEN OTHERS THEN
        RAISE NOTICE 'Autre erreur sur TCL101, transaction continue';
END;
$$;

-- mise à jour correcte
UPDATE Base_N
SET quantite = quantite + 5
WHERE vente_id = 'TCL102';

COMMIT;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

10. Fonctionnalités avancées

10.1 Procédures stockées

Une procédure stockée est un programme SQL enregistré dans la base de données. Elle permet d'exécuter une ou plusieurs instructions SQL de manière :

- Réutilisable
- Sécurisée
- Performante
- Paramétrable

C'est un pilier du SQL professionnel (applications, ETL, reporting).

Il y a plusieurs intérêts à utiliser des procédures stockées :

- Réutilisation : écrire une fois, exécuter partout
- Performance : plan d'exécution compilé
- Sécurité : accès contrôlé aux données
- Maintenance : logique centralisée
- Paramétrage : entrées dynamiques

La syntaxe de base est la suivante :

```
CREATE PROCEDURE nom_procedure(param1 type, param2 type)
LANGUAGE plpgsql
AS $$
BEGIN
    -- instructions SQL
END;
$$;
```

Puis l'exécution de la façon suivante :

```
CALL nom_procedure(valeur1, valeur2)
```

Quelques exemples :

- Une procédure d'ajout de vente :

```
CREATE PROCEDURE ajouter_vente(
    p_vente_id VARCHAR,
    p_client_id SMALLINT,
    p_vendeur_id SMALLINT,
    p_produit_id SMALLINT,
    p_quantite SMALLINT,
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
    p_discount NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Base_N(vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente)
    VALUES (p_vente_id, p_client_id, p_vendeur_id, p_produit_id, p_quantite, p_discount, CURRENT_DATE);
END;
$$;
```

Puis on l'exécute de la façon suivante :

```
CALL ajouter_vente('VPROC001', 1, 2, 3, 5, 0.1);
```

- On peut évidemment ajouter une gestion d'erreur dans une procédure

```
CREATE PROCEDURE ajouter_vente_safe(
    p_vente_id VARCHAR,
    p_client_id INTEGER,
    p_vendeur_id INTEGER,
    p_produit_id INTEGER,
    p_quantite INTEGER,
    p_discount NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Base_N(vente_id, client_id, vendeur_id, produit_id, quantite, discount, date_de_vente)
    VALUES (p_vente_id, p_client_id, p_vendeur_id, p_produit_id, p_quantite, p_discount, CURRENT_DATE);
EXCEPTION
    WHEN foreign_key_violation THEN
        RAISE NOTICE 'Erreur : client, vendeur ou produit invalide';
    WHEN check_violation THEN
        RAISE NOTICE 'Erreur : quantité ou discount invalide';
    WHEN OTHERS THEN
        RAISE NOTICE 'Autre erreur sur la procédure';
END;
$$;

CALL ajouter_vente_safe('V10352', 4, 6, 3, 3, 0.2);
```

Puis on l'exécute de la façon suivante :

```
CALL ajouter_vente_safe('VPROC002', 4, 6, 3, 3, 0.2);
```

Ces exemples montrent clairement que :

- Les procédures ne sont pas faites pour remplacer SELECT
- Elles servent à structurer, sécuriser et industrialiser

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

10.2 Triggers

Un trigger est un objet SQL qui s'exécute automatiquement lorsqu'un événement se produit sur une table ou une vue.

Contrairement aux procédures stockées :

- On ne les appelle pas explicitement
- Ils sont déclenchés par une action DML (INSERT, UPDATE, DELETE)

Les triggers sont utilisés lorsque l'on veut :

- Appliquer une règle métier automatique
- Garantir l'intégrité des données
- Effectuer des contrôles invisibles pour l'utilisateur
- Journaliser des changements (audit)
- Empêcher certaines opérations

Les Triggers les plus courants sont :

Type	Quand s'exécute-t-il ?
AFTER	Après l'opération (par défaut)
BEFORE	Avant l'opération

Sur PostgreSQL, le trigger se crée en 2 étapes :

- Création de la fonction de trigger qui contient la logique
- Puis création du trigger en tant que tel qui appelle la fonction

Pour la fonction de trigger, La syntaxe de base est la suivante :

```
CREATE OR REPLACE FUNCTION nom_fonction_trigger()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    -- Corps de la fonction : instructions SQL
    -- NEW : contient la ligne pour INSERT ou UPDATE
    -- OLD : contient la ligne pour UPDATE ou DELETE

    RETURN NEW; -- pour BEFORE triggers
    -- RETURN OLD; -- pour DELETE triggers si nécessaire
END;
$$;
```

- **RETURNS TRIGGER** : obligatoire, indique que la fonction est destinée à un trigger
- **NEW** : représente la nouvelle ligne (INSERT ou UPDATE)
- **OLD** : représente l'ancienne ligne (UPDATE ou DELETE)

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

- **RETURN NEW / OLD** : permet au trigger BEFORE de valider ou modifier la ligne avant l'opération

Enfin pour le trigger, la syntaxe de base est la suivante :

```
CREATE TRIGGER nom_trigger
    { BEFORE | AFTER } { INSERT | UPDATE | DELETE }
    ON nom_table
    [ FOR EACH ROW | FOR EACH STATEMENT ]
    EXECUTE FUNCTION nom_fonction_trigger();
```

- **BEFORE / AFTER** : moment du déclenchement par rapport à l'événement
- **INSERT / UPDATE / DELETE** : type d'événement
- **FOR EACH ROW** : déclenchement sur chaque ligne
- **FOR EACH STATEMENT** : déclenchement une fois par commande (toutes les lignes concernées)
- **EXECUTE FUNCTION** : le trigger appelle la fonction définie précédemment

Quelques exemples pour illustrer :

- Forcer le discount à 0 si la valeur est NULL :

```
CREATE OR REPLACE FUNCTION normalize_discount()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF NEW.discount IS NULL THEN
        NEW.discount := 0;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_normalize_discount
BEFORE INSERT ON Base_N
FOR EACH ROW
EXECUTE FUNCTION normalize_discount();
```

- Mettre à jour la table ventes_agg après chaque nouvelle vente :

```
CREATE OR REPLACE FUNCTION maj_ventes_agg()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO ventes_agg(produit_id, vendeur_id, quantite)
    VALUES (NEW.produit_id, NEW.vendeur_id, NEW.quantite)
    ON CONFLICT (produit_id, vendeur_id)
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
    DO UPDATE
    SET quantite = ventes_agg.quantite + EXCLUDED.quantite;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_maj_ventes_agg
AFTER INSERT ON Base_N
FOR EACH ROW
EXECUTE FUNCTION maj_ventes_agg();
```

- Interdire la suppression d'une vente si sa quantité est > 10 :

```
CREATE OR REPLACE FUNCTION protect_vente_importante()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.quantite > 10 THEN
        RAISE EXCEPTION
        'Suppression interdite : vente % trop importante', OLD.vente_id;
    END IF;

    RETURN OLD;
END;
$$;

CREATE TRIGGER trg_protect_vente
BEFORE DELETE ON Base_N
FOR EACH ROW
EXECUTE FUNCTION protect_vente_importante();
```

- Logger une seule fois par commande DELETE massive :

On commence par créer une table de log (journal).

```
CREATE TABLE log_commandes (
    action TEXT,
    table_cible TEXT,
    date_action TIMESTAMP
);
```

- Trigger INSTEAD OF pour interdire la suppression de ventes :

```
CREATE TRIGGER trg_no_delete_vente
ON Base_N
INSTEAD OF DELETE
AS
BEGIN
    THROW 50012, 'Suppression des ventes interdite', 1;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
END;

CREATE OR REPLACE FUNCTION log_delete_statement()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO log_commandes
    VALUES ('DELETE', 'Base_N', CURRENT_TIMESTAMP);

    RETURN NULL;
END;
$$;

CREATE TRIGGER trg_log_delete_statement
AFTER DELETE ON Base_N
FOR EACH STATEMENT
EXECUTE FUNCTION log_delete_statement();
```

Les triggers sont des gardiens automatiques des données.

Ils sont pertinents uniquement pour :

- Intégrité
- Audit
- Synchronisation
- Règles critiques

Ils doivent rester simples, courts et sûrs.

10.3 Exercices

1. Créer une procédure AjouterVente qui :

- Insère une vente dans *Base_N*.
- Refuse les quantités 0
- Refuse les discounts hors [0 ; 1]
- Affiche un message clair en cas d'erreur
- N'interrompt pas la session.

2. Créer une procédure qui :

- Augmente la quantité d'une vente donnée
- Refuse une quantité finale négative
- Affiche un message si la vente n'existe pas

3. Créer un trigger qui :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

- Empêche toute insertion avec quantité 0
- Empêche un discount hors [0 ; 1]

4. Tracer toute modification de quantité dans une table ventes_historique avec un Trigger.
5. Interdire la suppression d'une vente dont la quantité est > 3 avec un Trigger.
6. Interdire un discount supérieur à 25 % avec un Trigger.
7. Journaliser les suppressions de ventes avec un Trigger.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

10.4 Solutions

1. Créer une procédure AjouterVente qui :

- Insère une vente dans *Base_N*.
- Refuse les quantités 0
- Refuse les discounts hors [0 ; 1]
- Affiche un message clair en cas d'erreur
- N'interrompt pas la session.

```
CREATE OR REPLACE PROCEDURE ajouter_vente_secure(
    p_vente_id VARCHAR,
    p_client_id INT,
    p_vendeur_id INT,
    p_produit_id INT,
    p_quantite INT,
    p_discount NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF p_quantite <= 0 THEN
        RAISE EXCEPTION 'Quantité invalide : %', p_quantite;
    END IF;

    IF p_discount < 0 OR p_discount > 1 THEN
        RAISE EXCEPTION 'Discount invalide : %', p_discount;
    END IF;

    INSERT INTO Base_N
    VALUES (p_vente_id, p_client_id, p_vendeur_id,
            p_produit_id, p_quantite, p_discount, CURRENT_DATE);

EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Insertion échouée : %', SQLERRM;
END;
$$;
```

2. Créer une procédure qui :

- Augmente la quantité d'une vente donnée
- Refuse une quantité finale négative
- Affiche un message si la vente n'existe pas

```
CREATE OR REPLACE PROCEDURE maj_quantite_vente(
    p_vente_id VARCHAR,
    p_delta INT
)
LANGUAGE plpgsql
AS $$
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
DECLARE
    v_qte INT;
BEGIN
    SELECT quantite
    INTO v_qte
    FROM Base_N
    WHERE vente_id = p_vente_id;

    IF NOT FOUND THEN
        RAISE NOTICE 'Vente % inexistante', p_vente_id;
        RETURN;
    END IF;

    IF v_qte + p_delta < 0 THEN
        RAISE EXCEPTION 'Quantité finale négative interdite';
    END IF;

    UPDATE Base_N
    SET quantite = quantite + p_delta
    WHERE vente_id = p_vente_id;

END;
$$;
```

3. Créer un trigger qui :

- Empêche toute insertion avec quantité 0
- Empêche un discount hors [0 ; 1]

```
CREATE OR REPLACE FUNCTION check_vente()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF NEW.quantite <= 0 THEN
        RAISE EXCEPTION 'Quantité invalide';
    END IF;

    IF NEW.discount < 0 OR NEW.discount > 1 THEN
        RAISE EXCEPTION 'Discount invalide';
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_check_vente
BEFORE INSERT ON Base_N
FOR EACH ROW
EXECUTE FUNCTION check_vente();
```

4. Tracer toute modification de quantité dans une table ventes_historique..

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
CREATE TABLE ventes_historique (
    vente_id TEXT,
    ancienne_quantite INT,
    nouvelle_quantite INT,
    date_modif TIMESTAMP
);

CREATE OR REPLACE FUNCTION log_update_qte()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.quantite <> NEW.quantite THEN
        INSERT INTO ventes_historique
        VALUES (
            OLD.vente_id,
            OLD.quantite,
            NEW.quantite,
            CURRENT_TIMESTAMP
        );
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_log_update_qte
AFTER UPDATE ON Base_N
FOR EACH ROW
EXECUTE FUNCTION log_update_qte();
```

5. 5. Interdire la suppression d'une vente dont la quantité est > 3 avec un Trigger.

```sql
CREATE OR REPLACE FUNCTION protect_vente()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.quantite > 10 THEN
        RAISE EXCEPTION
        'Suppression interdite pour vente %', OLD.vente_id;
    END IF;

    RETURN OLD;
END;
$$;

CREATE TRIGGER trg_protect_delete
BEFORE DELETE ON Base_N
FOR EACH ROW
EXECUTE FUNCTION protect_vente();
```

6. Interdire un discount supérieur à 25 % avec un Trigger.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
CREATE OR REPLACE FUNCTION check_discount_25()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF NEW.discount > 0.25 THEN
        RAISE EXCEPTION
        'Discount interdit : %. Maximum autorisé = 25 %%', NEW.discount;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_check_discount_25
BEFORE INSERT OR UPDATE ON Base_N
FOR EACH ROW
EXECUTE FUNCTION check_discount_25();
```

7. Journaliser les suppressions de ventes avec un Trigger.

```
CREATE TABLE journal_suppressions_ventes (
    vente_id TEXT,
    vendeur_id SMALLINT,
    quantite_supprimee INT,
    date_suppression TIMESTAMP
);

CREATE OR REPLACE FUNCTION log_delete_vente()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO journal_suppressions_ventes (
        vente_id,
        vendeur_id,
        quantite_supprimee,
        date_suppression
    )
    VALUES (
        OLD.vente_id,
        OLD.vendeur_id,
        OLD."quantité",
        CURRENT_TIMESTAMP
    );

    RETURN OLD;
END;
$$;

CREATE TRIGGER trg_log_delete_vente
AFTER DELETE ON Base_N
FOR EACH ROW
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
EXECUTE FUNCTION log_delete_vente();
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

11. DCL (Data Control Language) : Gestion des droits (GRANT, REVOKE)

Le DCL regroupe les commandes SQL qui permettent de gérer la sécurité d'une base de données :

- Qui peut voir les données
- Qui peut les modifier
- Qui peut exécuter des objets (procédures, vues…)

Sur PostgreSQL, il n'y a pas de concept de login ou de user. Tout est un **ROLE** :

- Utilisateur
- Groupe
- Rôle technique
- Rôle applicatif

Un rôle peut :

- Se connecter (LOGIN)
- Posséder des objets
- Hériter de droits

11.1 Gestion des rôles

11.1.1 Création d'un rôle

La syntaxe de base est la suivante :

```
CREATE ROLE analyste;
```

Pour créer un rôle avec un droit de connexion :

```
CREATE ROLE analyste LOGIN PASSWORD 'mot_de_passe';
```

11.1.2 Modifier un rôle

La syntaxe de base est la suivante :

```
ALTER ROLE analyste PASSWORD 'nouveau_mdp';
```

11.1.3 Supprimer un rôle

La syntaxe de base est la suivante :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
DROP ROLE analyste;
```

Attention, Un rôle ne peut être supprimé s'il possède des objets ou des droits. Il faut d'abord les lui retirer.

11.2 Droits et privilèges

Les privilèges contrôlent l'accès aux objets de la base.

Objet	Privilèges possibles
Base de données	CONNECT, CREATE
Schéma	USAGE, CREATE
Table / Vue	SELECT, INSERT, UPDATE, DELETE
Fonction / Procédure	EXECUTE
Séquence	USAGE, SELECT, UPDATE

11.3 GRANT

Cette commande permet de donner un droit.

La syntaxe de base est la suivante :

```
GRANT permission ON objet TO rôle;
```

On peut accorder des droits sur une table :

```
GRANT SELECT, INSERT ON Base_N TO analyste;
```

Accorder des droits sur un schéma (indispensable) :

```
GRANT USAGE ON SCHEMA public TO analyste;
```

Sans ce droit, le rôle ne peut pas accéder aux tables du schéma.

On peut également accorder des droits sur toutes les tables d'un schéma :

```
GRANT SELECT ON ALL TABLES IN SCHEMA public TO analyste;
```

On peut aussi accorder un rôle à partir d'un autre rôle (héritage)

```
CREATE ROLE lecture_seule;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO lecture_seule;

GRANT lecture_seule TO analyste;
```

11.4 REVOKE

La commande permet de supprimer un droit accordé.

La syntaxe de base est la suivante :

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```
REVOKE permission
ON objet
FROM utilisateur_ou_rôle;
```

On peut révoquer un droit spécifique :

```
REVOKE INSERT ON Base_N FROM analyste;
```

Mais aussi révoquer tous les droits :

```
REVOKE ALL ON Base_N FROM analyste;
```

Il est également possible d'effectuer une révocation en cascade. C'est à dire révoquer tous les droits accordés par ce rôle à d'autres.

```
REVOKE ALL ON Base_N FROM analyste CASCADE;
```

11.5 Exercices

1. Créer un rôle lecteur capable de se connecter à la base, sans autres privilèges.
2. Donner au rôle lecteur le droit de lire toutes les tables du schéma public.
3. Faire en sorte que toutes les futures tables du schéma public soient automatiquement lisibles par lecteur.
4. Créer un rôle app_vente pouvant :

- Lire
- Insérer
- Modifier les données de la table Base_N.

5. Retirer au rôle app_vente le droit de modifier (UPDATE) la table Base_N.
6. Créer :

- Un rôle lecture_seule
- Un rôle analyste héritant des droits de lecture_seule

7. Autoriser analyste à exécuter une fonction ajouter_vente_secure() sans lui donner accès direct à la table Base_N.
8. Retirer tous les droits du rôle lecteur sur la table Produits.
9. Supprimer complètement le rôle app_vente.

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

11.6 Solutions

1. Créer un rôle lecteur capable de se connecter à la base, sans autres privilèges.

```
CREATE ROLE lecteur
LOGIN
PASSWORD 'lecteur_pwd';
```

2. Donner au rôle lecteur le droit de lire toutes les tables du schéma public.

```
GRANT USAGE ON SCHEMA public TO lecteur;

GRANT SELECT
ON ALL TABLES IN SCHEMA public
TO lecteur;
```

3. Faire en sorte que toutes les futures tables du schéma public soient automatiquement lisibles par lecteur.

```
ALTER DEFAULT PRIVILEGES
IN SCHEMA public
GRANT SELECT ON TABLES TO lecteur;
```

4. Créer un rôle app_vente pouvant :

- Lire
- Insérer
- Modifier les données de la table Base_N.

```
CREATE ROLE app_vente LOGIN PASSWORD 'app_pwd';

GRANT USAGE ON SCHEMA public TO app_vente;

GRANT SELECT, INSERT, UPDATE
ON Base_N
TO app_vente;
```

5. Retirer au rôle app_vente le droit de modifier (UPDATE) la table Base_N.

```
REVOKE UPDATE
ON Base_N
FROM app_vente;
```

6. Créer :

- Un rôle lecture_seule
- Un rôle analyste héritant des droits de lecture_seule

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.

```sql
CREATE ROLE lecture_seule;

GRANT SELECT
ON ALL TABLES IN SCHEMA public
TO lecture_seule;

CREATE ROLE analyste LOGIN PASSWORD 'analyste_pwd';

GRANT lecture_seule TO analyste;
```

7. Autoriser analyste à exécuter une fonction ajouter_vente_secure() sans lui donner accès direct à la table Base_N.

```sql
REVOKE ALL ON Base_N FROM analyste;

GRANT EXECUTE
ON FUNCTION ajouter_vente_secure(...)
TO analyste;
```

La table reste protégée, la logique est encapsulée.

8. Retirer tous les droits du rôle lecteur sur la table Produits.

```sql
REVOKE ALL
ON Produits
FROM lecteur;
```

9. Supprimer complètement le rôle app_vente.

```sql
REVOKE ALL
ON ALL TABLES IN SCHEMA public
FROM app_vente;

DROP ROLE app_vente;
```

© 2026 – Samy AMRI, JaimeExcel – Toute diffusion, revente ou usage non autorisé est interdit.