TD : SQL avec Elastisearch

⚠️ Basé sur l'index **kibana_sample_data_ecommerce**

Récapitulatif des 15 TD

TD 1-5 : Fondamentaux

- SELECT, WHERE, IN, Comparaisons, ORDER BY

TD 6-8 : Agrégations

- COUNT/SUM/AVG/MIN/MAX, GROUP BY, HAVING

TD 9-11 : Transformations

- Fonctions de chaînes, dates, CASE WHEN

TD 12-14 : Techniques Avancées

- Sous-requêtes, DISTINCT, LIKE

TD 15 : Analyses Complexes

- Combinaison de toutes les techniques pour des cas business réels

---

Optimisation des Performances

Règles d'Or :

- Toujours utiliser LIMIT en phase de test
- Filtrer avec WHERE avant de grouper (moins de données à traiter)
- Créer des index sur les champs fréquemment utilisés dans WHERE/GROUP BY
- Éviter SELECT * et ne sélectionner que les colonnes nécessaires
- Utiliser EXPLAIN pour analyser les plans d'exécution
- Préférer les agrégations natives aux calculs en post-traitement

Méthodologie de Résolution

Processus en 6 Étapes :

1. **Comprendre** la question business
2. **Identifier** les tables/champs nécessaires
3. **Écrire** la requête en plusieurs étapes
4. **Tester** chaque partie indépendamment
5. **Valider** les résultats avec des cas connus
6. **Optimiser** si nécessaire

Checklist Qualité d'une Requête

- [ ] La requête répond-elle à la question posée ?

- [ ] Utilise-t-elle des alias clairs et explicites ?
- [ ] Est-elle formatée de manière lisible ?
- [ ] Les filtres sont-ils optimaux (WHERE vs HAVING) ?
- [ ] Y a-t-il un LIMIT approprié ?
- [ ] Les agrégations sont-elles correctes ?
- [ ] Les types de données sont-ils cohérents ?

Ressources Supplémentaires

Documentation Officielle :

- Elasticsearch SQL : https://www.elastic.co/guide/en/elasticsearch/reference/current/xpack-sql.html
- Kibana Dev Tools pour tester les requêtes
- Visualisation des résultats dans Kibana

Outils Pratiques :

- Export des données en CSV/JSON pour analyses complémentaires
- Utilisation de variables pour les requêtes paramétrées
- Création de vues pour les requêtes fréquentes

Formats d'Exécution

Via API REST :

```
POST /_sql?format=txt
{
  "query": "SELECT * FROM kibana_sample_data_ecommerce LIMIT 5"
}
```

Formats disponibles :

- `format=txt` : affichage texte lisible
- `format=json` : format JSON structuré
- `format=csv` : export CSV
- `format=yaml` : format YAML

Commandes Utiles

Voir la structure d'un index :

```
DESCRIBE kibana_sample_data_ecommerce;
```

Lister les tables/index :

```
SHOW TABLES;
```

Voir les fonctions disponibles :

```
SHOW FUNCTIONS;
```

TD 1 : Requêtes SELECT Simples

Objectif

Comprendre la structure de base d'une requête SQL Elasticsearch et récupérer des données.

Requête SQL de Base

```
SELECT * FROM kibana_sample_data_ecommerce LIMIT 10;
```

Explication

1. **SELECT** : sélectionne toutes les colonnes
2. **FROM** : spécifie l'index Elasticsearch (équivalent d'une table)
3. **LIMIT** : limite le nombre de résultats retournés

Sélection de Colonnes Spécifiques

```
SELECT order_id, customer_full_name, taxful_total_price
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Exécution via API

```
POST /_sql?format=json
{
  "query": "SELECT customer_full_name, total_unique_products FROM
kibana_sample_data_ecommerce LIMIT 10"
}
```

Bonnes Pratiques

- Toujours utiliser LIMIT pour éviter de surcharger le cluster
- Sélectionner uniquement les colonnes nécessaires (performances)
- Utiliser `format=txt` pour une lecture facile, `format=json` pour traitement programmatique

- Les noms d'index sont sensibles à la casse

Erreurs à Éviter

- Oublier LIMIT sur de gros volumes de données
- Utiliser SELECT * systématiquement (charge réseau inutile)
- Ne pas vérifier l'existence de l'index avant la requête

Exercices Supplémentaires TD 1

Exercice 1.1 : Récupérer uniquement order_id, customer_id et total_quantity des 5 premières commandes

Corrigé :

```
SELECT order_id, customer_id, total_quantity
FROM kibana_sample_data_ecommerce
LIMIT 5;
```

Exercice 1.2 : Récupérer toutes les informations géographiques (geoip) des 10 premières commandes

Corrigé :

```
SELECT order_id, geoip.city_name, geoip.country_iso_code, geoip.continent_name
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Exercice 1.3 : Lister les emails et numéros de téléphone des clients

Corrigé :

```
SELECT customer_full_name, email, customer_phone
FROM kibana_sample_data_ecommerce
LIMIT 20;
```

TD 2 : Clause WHERE - Filtres Simples

Objectif

Filtrer les données avec des conditions d'égalité.

Requête SQL avec WHERE

```
SELECT order_id, customer_full_name, customer_gender
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
LIMIT 10;
```

Explication

1. **WHERE** : clause de filtrage
2. **=** : opérateur d'égalité stricte
3. Les chaînes de caractères doivent être entre guillemets simples
4. Respecter la casse pour les valeurs (FEMALE vs female)

Filtres Multiples avec AND

```
SELECT order_id, customer_gender, currency
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
  AND currency = 'EUR'
LIMIT 10;
```

Filtres Multiples avec OR

```
SELECT order_id, currency
FROM kibana_sample_data_ecommerce
WHERE currency = 'EUR'
   OR currency = 'USD'
LIMIT 10;
```

Bonnes Pratiques

- Filtrer au maximum côté base de données (WHERE) plutôt qu'en post-traitement
- Utiliser des index sur les champs filtrés fréquemment
- Combiner AND et OR avec des parenthèses pour éviter les ambiguïtés
- Privilégier les champs keyword pour les filtres d'égalité

Erreurs à Éviter

- Oublier les guillemets autour des chaînes de caractères
- Mauvaise casse dans les valeurs (case-sensitive)
- Utiliser WHERE sur des champs text sans précaution

Exercices Supplémentaires TD 2

Exercice 2.1 : Trouver toutes les commandes du jour "Monday"

Corrigé :

```
SELECT order_id, customer_full_name, day_of_week
FROM kibana_sample_data_ecommerce
WHERE day_of_week = 'Monday'
LIMIT 20;
```

Exercice 2.2 : Trouver les commandes de type "order" en devise EUR

Corrigé :

```
SELECT order_id, type, currency, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE type = 'order'
  AND currency = 'EUR'
LIMIT 15;
```

Exercice 2.3 : Trouver les commandes des clients masculins d'Europe ou d'Asie

Corrigé :

```
SELECT order_id, customer_gender, geoip.continent_name
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'MALE'
  AND (geoip.continent_name = 'Europe' OR geoip.continent_name = 'Asia')
LIMIT 20;
```

Exercice 2.4 : Trouver les commandes où le customer_id est "10" ou "20"

Corrigé :

```
SELECT order_id, customer_id, customer_full_name
FROM kibana_sample_data_ecommerce
WHERE customer_id = '10'
   OR customer_id = '20';
```

TD 3 : Opérateur IN et NOT IN

Objectif

Simplifier les filtres avec plusieurs valeurs possibles.

Requête SQL avec IN

```
SELECT order_id, currency, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE currency IN ('EUR', 'USD', 'GBP')
LIMIT 10;
```

Explication

1. **IN** : vérifie si la valeur est dans une liste
2. Équivalent à plusieurs OR : `currency = 'EUR' OR currency = 'USD' OR currency = 'GBP'`
3. Plus lisible et performant que des OR multiples
4. La liste de valeurs est entre parenthèses, séparées par des virgules

Requête avec NOT IN

```
SELECT order_id, day_of_week
FROM kibana_sample_data_ecommerce
WHERE day_of_week NOT IN ('Saturday', 'Sunday')
LIMIT 10;
```

Comparaison OR vs IN

```
-- Moins lisible
WHERE currency = 'EUR' OR currency = 'USD' OR currency = 'GBP'

-- Plus lisible
WHERE currency IN ('EUR', 'USD', 'GBP')
```

Bonnes Pratiques

- Utiliser IN pour plus de 2 valeurs à comparer
- NOT IN est utile pour les exclusions
- Limiter la liste IN à un nombre raisonnable de valeurs (< 100)
- IN fonctionne bien avec les champs keyword

Erreurs à Éviter

- Oublier les parenthèses autour de la liste de valeurs
- Mélanger types de données dans la liste IN
- Utiliser IN avec trop de valeurs (impact performance)

- Oublier les guillemets pour les chaînes

Exercices Supplémentaires TD 3

Exercice 3.1 : Trouver les commandes passées lundi, mercredi ou vendredi

Corrigé :

```
SELECT order_id, day_of_week, customer_full_name
FROM kibana_sample_data_ecommerce
WHERE day_of_week IN ('Monday', 'Wednesday', 'Friday')
LIMIT 15;
```

Exercice 3.2 : Trouver les commandes des continents Europe, Amérique du Nord et Asie

Corrigé :

```
SELECT order_id, geoip.continent_name, geoip.country_iso_code
FROM kibana_sample_data_ecommerce
WHERE geoip.continent_name IN ('Europe', 'North America', 'Asia')
LIMIT 20;
```

Exercice 3.3 : Exclure les commandes en EUR et USD (toutes les autres devises)

Corrigé :

```
SELECT order_id, currency, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE currency NOT IN ('EUR', 'USD')
LIMIT 10;
```

Exercice 3.4 : Combiner IN et AND - Commandes féminines en EUR ou GBP

Corrigé :

```
SELECT order_id, customer_gender, currency, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
  AND currency IN ('EUR', 'GBP')
LIMIT 15;
```

TD 4 : Opérateurs de Comparaison Numérique

Objectif

Filtrer sur des valeurs numériques avec les opérateurs <, >, <=, >=, !=.

Requête SQL - Supérieur à

```sql
SELECT order_id, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price > 100
LIMIT 10;
```

Explication

1. **>** : strictement supérieur
2. **>=** : supérieur ou égal
3. **<** : strictement inférieur
4. **<=** : inférieur ou égal
5. **!=** ou **<>** : différent de

Requête avec Intervalle (BETWEEN)

```sql
SELECT order_id, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price >= 50
  AND taxful_total_price <= 150
LIMIT 10;
```

Version avec BETWEEN

```sql
SELECT order_id, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price BETWEEN 50 AND 150
LIMIT 10;
```

Explication BETWEEN

1. **BETWEEN** est inclusif (inclut les bornes)
2. Équivalent à : `>= valeur_min AND <= valeur_max`
3. Plus lisible pour les intervalles

Comparaisons sur Dates

```sql
SELECT order_id, order_date
FROM kibana_sample_data_ecommerce
WHERE order_date >= '2024-01-01'
LIMIT 10;
```

Bonnes Pratiques

- Utiliser BETWEEN pour les intervalles (plus lisible)
- Indexer les champs numériques fréquemment filtrés
- Attention aux types : ne pas comparer des strings avec des nombres
- Pour les dates, utiliser le format ISO 8601 : 'YYYY-MM-DD'

Erreurs à Éviter

- Inverser les bornes dans BETWEEN (min doit être < max)
- Oublier que BETWEEN est inclusif
- Comparer des types incompatibles
- Utiliser != sur des champs pouvant être NULL

Exercices Supplémentaires TD 4

Exercice 4.1 : Trouver les commandes avec un montant total supérieur à 200

Corrigé :

```
SELECT order_id, customer_full_name, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price > 200
LIMIT 10;
```

Exercice 4.2 : Trouver les commandes avec une quantité totale entre 3 et 5 articles

Corrigé :

```
SELECT order_id, total_quantity, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE total_quantity BETWEEN 3 AND 5
LIMIT 15;
```

Exercice 4.3 : Trouver les commandes de moins de 30 euros

Corrigé :

```
SELECT order_id, taxful_total_price, currency
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price < 30
LIMIT 10;
```

Exercice 4.4 : Trouver les clients ayant passé commande avec 2 produits uniques ou plus, montant > 80

Corrigé :

```
SELECT order_id, total_unique_products, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE total_unique_products >= 2
  AND taxful_total_price > 80
LIMIT 20;
```

Exercice 4.5 : Trouver les commandes passées en 2024 (après le 1er janvier 2024)

Corrigé :

```
SELECT order_id, order_date, customer_full_name
FROM kibana_sample_data_ecommerce
WHERE order_date >= '2024-01-01'
LIMIT 10;
```

TD 5 : Tri avec ORDER BY

Objectif

Trier les résultats par ordre croissant ou décroissant.

Requête SQL - Tri Croissant

```
SELECT order_id, taxful_total_price
FROM kibana_sample_data_ecommerce
ORDER BY taxful_total_price ASC
LIMIT 10;
```

Explication

1. **ORDER BY** : clause de tri
2. **ASC** : ordre croissant (ascending) - valeurs les plus petites d'abord
3. **DESC** : ordre décroissant (descending) - valeurs les plus grandes d'abord
4. ASC est la valeur par défaut (peut être omis)

Requête - Tri Décroissant

```
SELECT order_id, taxful_total_price
FROM kibana_sample_data_ecommerce
ORDER BY taxful_total_price DESC
LIMIT 10;
```

Tri Multi-Colonnes

```
SELECT order_id, currency, taxful_total_price
FROM kibana_sample_data_ecommerce
ORDER BY currency ASC, taxful_total_price DESC
LIMIT 20;
```

Explication Tri Multi-Colonnes

1. Trie d'abord par currency (ordre alphabétique)
2. Pour chaque devise, trie par taxful_total_price décroissant
3. L'ordre des colonnes dans ORDER BY est important

Tri avec Filtre

```
SELECT order_id, customer_gender, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
ORDER BY taxful_total_price DESC
LIMIT 5;
```

Bonnes Pratiques

- Toujours spécifier ASC ou DESC explicitement (clarté du code)
- Utiliser ORDER BY avec LIMIT pour éviter de trier toutes les données
- Trier sur des champs indexés pour de meilleures performances
- Le tri peut être coûteux sur de gros volumes

Erreurs à Éviter

- Oublier LIMIT avec ORDER BY sur de gros datasets
- Trier sur des champs text non keyword (performances)
- Inverser l'ordre ASC/DESC selon le besoin
- Ordre des colonnes dans ORDER BY incorrect pour tri multi-niveaux

---

Exercices Supplémentaires TD 5

Exercice 5.1 : Trouver les 10 commandes les moins chères

Corrigé :

```
SELECT order_id, customer_full_name, taxful_total_price
FROM kibana_sample_data_ecommerce
```

```
ORDER BY taxful_total_price ASC
LIMIT 10;
```

Exercice 5.2 : Trouver les 5 commandes les plus récentes

Corrigé :

```
SELECT order_id, order_date, customer_full_name
FROM kibana_sample_data_ecommerce
ORDER BY order_date DESC
LIMIT 5;
```

Exercice 5.3 : Lister les commandes par genre (alphabétique) puis par montant décroissant

Corrigé :

```
SELECT order_id, customer_gender, taxful_total_price
FROM kibana_sample_data_ecommerce
ORDER BY customer_gender ASC, taxful_total_price DESC
LIMIT 20;
```

Exercice 5.4 : Trouver les 10 plus grosses commandes en EUR uniquement

Corrigé :

```
SELECT order_id, currency, taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE currency = 'EUR'
ORDER BY taxful_total_price DESC
LIMIT 10;
```

Exercice 5.5 : Lister les commandes par continent, puis par pays, puis par montant

Corrigé :

```
SELECT order_id, geoip.continent_name, geoip.country_iso_code, taxful_total_price
FROM kibana_sample_data_ecommerce
ORDER BY geoip.continent_name ASC, geoip.country_iso_code ASC, taxful_total_price
DESC
LIMIT 30;
```

TD 6 : Fonctions d'Agrégation (COUNT, SUM, AVG, MIN, MAX)

Objectif

Calculer des statistiques sur l'ensemble des données.

COUNT - Compter les Enregistrements

```
SELECT COUNT(*) AS nombre_commandes
FROM kibana_sample_data_ecommerce;
```

Explication

1. **COUNT(*)** : compte toutes les lignes
2. **AS** : alias pour renommer la colonne résultante
3. Pas de LIMIT nécessaire (une seule ligne de résultat)
4. COUNT(*) compte aussi les valeurs NULL

SUM - Somme des Valeurs

```
SELECT SUM(taxful_total_price) AS chiffre_affaires_total
FROM kibana_sample_data_ecommerce;
```

AVG - Moyenne

```
SELECT AVG(taxful_total_price) AS panier_moyen
FROM kibana_sample_data_ecommerce;
```

MIN et MAX

```
SELECT
  MIN(taxful_total_price) AS montant_minimum,
  MAX(taxful_total_price) AS montant_maximum
FROM kibana_sample_data_ecommerce;
```

Plusieurs Agrégations Simultanées

```
SELECT
  COUNT(*) AS nombre_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  MIN(taxful_total_price) AS min_commande,
  MAX(taxful_total_price) AS max_commande
FROM kibana_sample_data_ecommerce;
```

Agrégation avec Filtre

```
SELECT
  COUNT(*) AS commandes_femmes,
  AVG(taxful_total_price) AS panier_moyen_femmes
FROM kibana_sample_data_ecommerce
```

```
WHERE customer_gender = 'FEMALE';
```

Bonnes Pratiques

- Toujours utiliser des alias (AS) pour nommer les colonnes agrégées
- COUNT(*) vs COUNT(colonne) : COUNT(colonne) ignore les NULL
- Vérifier les types de données avant d'utiliser SUM ou AVG
- Les agrégations sont calculées sur l'ensemble des données filtrées

Erreurs à Éviter

- Oublier les alias (résultats difficiles à interpréter)
- Utiliser SUM sur des champs non numériques
- Mélanger colonnes simples et agrégées sans GROUP BY
- Utiliser LIMIT avec des agrégations (inutile)

Exercices Supplémentaires TD 6

Exercice 6.1 : Compter le nombre total de commandes en EUR

Corrigé :

```
SELECT COUNT(*) AS commandes_eur
FROM kibana_sample_data_ecommerce
WHERE currency = 'EUR';
```

Exercice 6.2 : Calculer le chiffre d'affaires total généré le lundi

Corrigé :

```
SELECT SUM(taxful_total_price) AS ca_lundi
FROM kibana_sample_data_ecommerce
WHERE day_of_week = 'Monday';
```

Exercice 6.3 : Trouver le panier moyen des clients masculins

Corrigé :

```
SELECT AVG(taxful_total_price) AS panier_moyen_hommes
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'MALE';
```

Exercice 6.4 : Statistiques complètes sur les commandes d'Europe

Corrigé :

```
SELECT
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  MIN(taxful_total_price) AS min_commande,
  MAX(taxful_total_price) AS max_commande
FROM kibana_sample_data_ecommerce
WHERE geoip.continent_name = 'Europe';
```

Exercice 6.5 : Trouver la quantité totale d'articles vendus

Corrigé :

```
SELECT SUM(total_quantity) AS articles_vendus_total
FROM kibana_sample_data_ecommerce;
```

TD 7 : GROUP BY - Regroupements

Objectif

Regrouper les données et calculer des agrégations par groupe.

Requête GROUP BY Simple

```
SELECT
  customer_gender,
  COUNT(*) AS nombre_commandes
FROM kibana_sample_data_ecommerce
GROUP BY customer_gender;
```

Explication

1. **GROUP BY** : regroupe les lignes ayant la même valeur
2. Crée un groupe par valeur unique de la colonne
3. Les agrégations sont calculées pour chaque groupe
4. Toute colonne non agrégée doit être dans GROUP BY

GROUP BY avec Plusieurs Agrégations

```
SELECT
  currency,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen
```

```
FROM kibana_sample_data_ecommerce
GROUP BY currency;
```

GROUP BY Multi-Colonnes

```
SELECT
  customer_gender,
  currency,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS panier_moyen
FROM kibana_sample_data_ecommerce
GROUP BY customer_gender, currency;
```

Explication GROUP BY Multi-Colonnes

1. Crée un groupe pour chaque combinaison unique de valeurs
2. Exemple : (MALE, EUR), (MALE, USD), (FEMALE, EUR), etc.
3. Les agrégations sont calculées pour chaque combinaison

GROUP BY avec ORDER BY

```
SELECT
  day_of_week,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_jour
FROM kibana_sample_data_ecommerce
GROUP BY day_of_week
ORDER BY ca_jour DESC;
```

GROUP BY avec WHERE

```
SELECT
  geoip.continent_name,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS panier_moyen
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
GROUP BY geoip.continent_name
ORDER BY nb_commandes DESC;
```

Ordre d'Exécution SQL

1. FROM - sélection de la table
2. WHERE - filtrage des lignes
3. GROUP BY - regroupement

4. Agrégations (COUNT, SUM, etc.)
5. ORDER BY - tri des résultats
6. LIMIT - limitation des résultats

Bonnes Pratiques

- Toute colonne non agrégée doit être dans GROUP BY
- Utiliser des alias pour les colonnes agrégées
- Combiner WHERE (filtre avant regroupement) et HAVING (filtre après)
- Trier par colonnes agrégées pour voir les top/bottom performers

Erreurs à Éviter

- Oublier une colonne SELECT dans GROUP BY
- Utiliser HAVING sans GROUP BY
- Confondre WHERE (filtre lignes) et HAVING (filtre groupes)
- Ne pas utiliser d'alias pour les colonnes calculées

Exercices Supplémentaires TD 7

Exercice 7.1 : Nombre de commandes par jour de la semaine

Corrigé :

```
SELECT
  day_of_week,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY day_of_week
ORDER BY nb_commandes DESC;
```

Exercice 7.2 : Chiffre d'affaires par continent

Corrigé :

```
SELECT
  geoip.continent_name,
  SUM(taxful_total_price) AS ca_continent,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY geoip.continent_name
ORDER BY ca_continent DESC;
```

Exercice 7.3 : Panier moyen par devise et par genre

Corrigé :

```
SELECT
  currency,
  customer_gender,
  AVG(taxful_total_price) AS panier_moyen,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY currency, customer_gender
ORDER BY currency, customer_gender;
```

Exercice 7.4 : Top 5 des pays générant le plus de CA

Corrigé :

```
SELECT
  geoip.country_iso_code,
  SUM(taxful_total_price) AS ca_pays,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY geoip.country_iso_code
ORDER BY ca_pays DESC
LIMIT 5;
```

Exercice 7.5 : Statistiques des commandes féminines par jour de semaine

Corrigé :

```
SELECT
  day_of_week,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS panier_moyen,
  SUM(taxful_total_price) AS ca_total
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
GROUP BY day_of_week
ORDER BY ca_total DESC;
```

TD 8 : HAVING - Filtrer les Groupes

Objectif

Filtrer les résultats après le regroupement (contrairement à WHERE qui filtre avant).

Requête avec HAVING

```
SELECT
  currency,
```

```
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total
FROM kibana_sample_data_ecommerce
GROUP BY currency
HAVING COUNT(*) > 1000;
```

Explication

1. **HAVING** : filtre appliqué APRÈS le GROUP BY
2. Utilise les fonctions d'agrégation (COUNT, SUM, AVG, etc.)
3. WHERE filtre les lignes AVANT regroupement
4. HAVING filtre les groupes APRÈS regroupement

Différence WHERE vs HAVING

```
-- WHERE : filtre les lignes individuelles
SELECT
  currency,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price > 50
GROUP BY currency;

-- HAVING : filtre les groupes
SELECT
  currency,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY currency
HAVING COUNT(*) > 1000;
```

WHERE et HAVING Combinés

```
SELECT
  geoip.continent_name,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS panier_moyen
FROM kibana_sample_data_ecommerce
WHERE customer_gender = 'FEMALE'
GROUP BY geoip.continent_name
HAVING COUNT(*) > 500
ORDER BY panier_moyen DESC;
```

HAVING avec Plusieurs Conditions

```
SELECT
  day_of_week,
```

```
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_jour
FROM kibana_sample_data_ecommerce
GROUP BY day_of_week
HAVING COUNT(*) > 500
  AND SUM(taxful_total_price) > 50000
ORDER BY ca_jour DESC;
```

Ordre d'Exécution Complet

1. FROM
2. WHERE (filtre lignes)
3. GROUP BY
4. Agrégations
5. HAVING (filtre groupes)
6. ORDER BY
7. LIMIT

Bonnes Pratiques

- Utiliser WHERE pour filtrer les données brutes (plus performant)
- Utiliser HAVING pour filtrer sur des résultats d'agrégation
- Combiner WHERE et HAVING pour des requêtes optimisées
- Utiliser des alias dans SELECT mais répéter l'agrégation dans HAVING

Erreurs à Éviter

- Utiliser HAVING sans GROUP BY
- Mettre des filtres de lignes dans HAVING au lieu de WHERE
- Utiliser des colonnes non agrégées dans HAVING
- Oublier que HAVING s'applique après GROUP BY

Exercices Supplémentaires TD 8

Exercice 8.1 : Trouver les jours de la semaine avec plus de 600 commandes

Corrigé :

```
SELECT
  day_of_week,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
```

```
GROUP BY day_of_week
HAVING COUNT(*) > 600
ORDER BY nb_commandes DESC;
```

Exercice 8.2 : Trouver les devises avec un panier moyen supérieur à 80

Corrigé :

```
SELECT
  currency,
  AVG(taxful_total_price) AS panier_moyen,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY currency
HAVING AVG(taxful_total_price) > 80;
```

Exercice 8.3 : Trouver les continents avec au moins 1000 commandes et un CA > 100000

Corrigé :

```
SELECT
  geoip.continent_name,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total
FROM kibana_sample_data_ecommerce
GROUP BY geoip.continent_name
HAVING COUNT(*) >= 1000
  AND SUM(taxful_total_price) > 100000
ORDER BY ca_total DESC;
```

Exercice 8.4 : Pays avec moins de 100 commandes (petits marchés)

Corrigé :

```
SELECT
  geoip.country_iso_code,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total
FROM kibana_sample_data_ecommerce
GROUP BY geoip.country_iso_code
HAVING COUNT(*) < 100
ORDER BY nb_commandes ASC;
```

Exercice 8.5 : Combinaison WHERE + HAVING - Commandes féminines, continents avec panier moyen > 75

Corrigé :

```
SELECT
  geoip.continent_name,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS panier_moyen
FROM kibana_sample_data_ecommerce
```

```
WHERE customer_gender = 'FEMALE'
GROUP BY geoip.continent_name
HAVING AVG(taxful_total_price) > 75
ORDER BY panier_moyen DESC;
```

TD 9 : Fonctions de Chaînes de Caractères

Objectif

Manipuler et transformer les données textuelles.

CONCAT - Concaténation

```
SELECT
  order_id,
  CONCAT(customer_first_name, ' ', customer_last_name) AS nom_complet
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Explication

1. **CONCAT** : fusionne plusieurs chaînes de caractères
2. Accepte un nombre variable d'arguments
3. Utile pour créer des identifiants ou des labels

UPPER et LOWER - Conversion de Casse

```
SELECT
  order_id,
  UPPER(customer_full_name) AS nom_majuscule,
  LOWER(customer_full_name) AS nom_minuscule
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

LENGTH - Longueur d'une Chaîne

```
SELECT
  customer_full_name,
  LENGTH(customer_full_name) AS longueur_nom
FROM kibana_sample_data_ecommerce
WHERE LENGTH(customer_full_name) > 20
LIMIT 10;
```

SUBSTRING - Extraction de Sous-chaînes

```
SELECT
  order_id,
  SUBSTRING(order_id, 1, 3) AS prefixe_ordre
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

TRIM, LTRIM, RTRIM - Suppression d'Espaces

```
SELECT
  TRIM(customer_full_name) AS nom_sans_espaces
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Bonnes Pratiques

- Utiliser les fonctions de chaînes pour normaliser les données
- UPPER/LOWER pour les comparaisons insensibles à la casse
- LENGTH pour valider la longueur des données
- CONCAT pour créer des clés composites

Erreurs à Éviter

- Utiliser CONCAT avec des valeurs NULL (retourne NULL)
- Oublier les index lors de SUBSTRING (commence à 1 en SQL)
- Appliquer ces fonctions sur des champs non textuels
- Performance : éviter les fonctions sur de gros volumes dans WHERE

Exercices Supplémentaires TD 9

Exercice 9.1 : Créer un identifiant client avec préfixe "CUST-"

Corrigé :

```
SELECT
  CONCAT('CUST-', customer_id) AS identifiant_client,
  customer_full_name
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Exercice 9.2 : Afficher les emails en majuscules

Corrigé :

```
SELECT
  customer_full_name,
  UPPER(email) AS email_majuscule
FROM kibana_sample_data_ecommerce
LIMIT 15;
```

Exercice 9.3 : Trouver les clients avec des noms de plus de 15 caractères

Corrigé :

```
SELECT
  customer_full_name,
  LENGTH(customer_full_name) AS longueur
FROM kibana_sample_data_ecommerce
WHERE LENGTH(customer_full_name) > 15
LIMIT 20;
```

Exercice 9.4 : Extraire les 2 premiers caractères du code pays

Corrigé :

```
SELECT
  order_id,
  geoip.country_iso_code,
  SUBSTRING(geoip.country_iso_code, 1, 2) AS code_court
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Exercice 9.5 : Créer un libellé de commande complet

Corrigé :

```
SELECT
  CONCAT('Commande #', order_id, ' - ', customer_full_name, ' (', currency, ')') AS
libelle_commande
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

TD 10 : Fonctions de Date et Temps

Objectif

Manipuler et extraire des informations des dates.

YEAR, MONTH, DAY - Extraction de Composants

```
SELECT
  order_id,
```

```
  order_date,
  YEAR(order_date) AS annee,
  MONTH(order_date) AS mois,
  DAY(order_date) AS jour
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Explication

1. **YEAR** : extrait l'année (format numérique)
2. **MONTH** : extrait le mois (1-12)
3. **DAY** : extrait le jour du mois (1-31)
4. **DAY_OF_WEEK** : jour de la semaine (1=dimanche, 7=samedi)

DAY_OF_WEEK et WEEK

```
SELECT
  order_id,
  order_date,
  DAY_OF_WEEK(order_date) AS jour_semaine_num,
  WEEK(order_date) AS numero_semaine
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Filtres sur Dates Extraites

```
SELECT
  order_id,
  order_date,
  taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE YEAR(order_date) = 2024
  AND MONTH(order_date) = 6
LIMIT 20;
```

NOW() et CURRENT_TIMESTAMP

```
SELECT
  order_id,
  order_date,
  CURRENT_TIMESTAMP AS date_actuelle
FROM kibana_sample_data_ecommerce
LIMIT 5;
```

Agrégations par Période

```
SELECT
  YEAR(order_date) AS annee,
  MONTH(order_date) AS mois,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_mensuel
FROM kibana_sample_data_ecommerce
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY annee DESC, mois DESC;
```

Bonnes Pratiques

- Utiliser les fonctions de date pour les analyses temporelles
- Indexer les champs date pour de meilleures performances
- GROUP BY sur YEAR/MONTH pour des tendances mensuelles
- Utiliser BETWEEN pour les intervalles de dates

Erreurs à Éviter

- Appliquer des fonctions de date sur des champs non date
- Oublier de trier chronologiquement les résultats temporels
- Comparer des dates avec des chaînes sans conversion
- Utiliser des fonctions de date dans WHERE (impact performance)

Exercices Supplémentaires TD 10

Exercice 10.1 : Compter les commandes par année

Corrigé :

```
SELECT
  YEAR(order_date) AS annee,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY YEAR(order_date)
ORDER BY annee DESC;
```

Exercice 10.2 : CA par mois pour l'année 2024

Corrigé :

```
SELECT
  MONTH(order_date) AS mois,
  SUM(taxful_total_price) AS ca_mensuel,
  COUNT(*) AS nb_commandes
```

```
FROM kibana_sample_data_ecommerce
WHERE YEAR(order_date) = 2024
GROUP BY MONTH(order_date)
ORDER BY mois ASC;
```

Exercice 10.3 : Trouver les commandes du premier trimestre (janvier-mars)

Corrigé :

```
SELECT
  order_id,
  order_date,
  taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE MONTH(order_date) IN (1, 2, 3)
LIMIT 20;
```

Exercice 10.4 : Analyser les commandes par jour de la semaine (numérique)

Corrigé :

```
SELECT
  DAY_OF_WEEK(order_date) AS jour_num,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS panier_moyen
FROM kibana_sample_data_ecommerce
GROUP BY DAY_OF_WEEK(order_date)
ORDER BY jour_num;
```

Exercice 10.5 : CA hebdomadaire (par numéro de semaine)

Corrigé :

```
SELECT
  YEAR(order_date) AS annee,
  WEEK(order_date) AS semaine,
  SUM(taxful_total_price) AS ca_hebdo,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY YEAR(order_date), WEEK(order_date)
ORDER BY annee DESC, semaine DESC
LIMIT 10;
```

TD 11 : CASE WHEN - Logique Conditionnelle

Objectif

Créer des colonnes calculées avec des conditions logiques.

Structure de Base CASE WHEN

```
SELECT
  order_id,
  taxful_total_price,
  CASE
    WHEN taxful_total_price < 50 THEN 'Petit panier'
    WHEN taxful_total_price >= 50 AND taxful_total_price < 100 THEN 'Panier moyen'
    ELSE 'Gros panier'
  END AS categorie_panier
FROM kibana_sample_data_ecommerce
LIMIT 20;
```

Explication

1. **CASE** : début du bloc conditionnel
2. **WHEN condition THEN valeur** : chaque condition et sa valeur
3. **ELSE** : valeur par défaut si aucune condition n'est vraie
4. **END** : fin du bloc (obligatoire)
5. **AS** : alias pour nommer la nouvelle colonne

CASE avec Égalités

```
SELECT
  order_id,
  customer_gender,
  CASE customer_gender
    WHEN 'MALE' THEN 'Homme'
    WHEN 'FEMALE' THEN 'Femme'
    ELSE 'Non spécifié'
  END AS genre_fr
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Agrégations avec CASE

```
SELECT
  COUNT(*) AS total_commandes,
  SUM(CASE WHEN customer_gender = 'MALE' THEN 1 ELSE 0 END) AS commandes_hommes,
  SUM(CASE WHEN customer_gender = 'FEMALE' THEN 1 ELSE 0 END) AS commandes_femmes
FROM kibana_sample_data_ecommerce;
```

CASE dans GROUP BY

```
SELECT
  CASE
```

```
    WHEN taxful_total_price < 50 THEN 'Petit'
    WHEN taxful_total_price < 100 THEN 'Moyen'
    ELSE 'Gros'
  END AS segment_panier,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS montant_moyen
FROM kibana_sample_data_ecommerce
GROUP BY
  CASE
    WHEN taxful_total_price < 50 THEN 'Petit'
    WHEN taxful_total_price < 100 THEN 'Moyen'
    ELSE 'Gros'
  END;
```

CASE Imbriqués

```
SELECT
  order_id,
  customer_gender,
  taxful_total_price,
  CASE
    WHEN customer_gender = 'FEMALE' THEN
      CASE
        WHEN taxful_total_price > 100 THEN 'Femme - Premium'
        ELSE 'Femme - Standard'
      END
    WHEN customer_gender = 'MALE' THEN
      CASE
        WHEN taxful_total_price > 100 THEN 'Homme - Premium'
        ELSE 'Homme - Standard'
      END
    ELSE 'Autre'
  END AS segment_client
FROM kibana_sample_data_ecommerce
LIMIT 20;
```

Bonnes Pratiques

- Toujours inclure un ELSE pour gérer les cas non prévus
- Utiliser des alias explicites pour les colonnes CASE
- Ordonner les conditions de la plus spécifique à la plus générale
- Pour GROUP BY, répéter l'expression CASE complète

Erreurs à Éviter

- Oublier le END à la fin du CASE
- Ne pas prévoir de ELSE (retourne NULL par défaut)
- Conditions qui se chevauchent (seule la première est évaluée)

- Types de retour incohérents entre les branches

Exercices Supplémentaires TD 11

Exercice 11.1 : Catégoriser les commandes par tranche de prix (0-30, 30-80, 80-150, 150+)

Corrigé :

```
SELECT
  order_id,
  taxful_total_price,
  CASE
    WHEN taxful_total_price < 30 THEN '0-30'
    WHEN taxful_total_price < 80 THEN '30-80'
    WHEN taxful_total_price < 150 THEN '80-150'
    ELSE '150+'
  END AS tranche_prix
FROM kibana_sample_data_ecommerce
LIMIT 25;
```

Exercice 11.2 : Identifier les jours ouvrables vs weekend

Corrigé :

```
SELECT
  order_id,
  day_of_week,
  CASE
    WHEN day_of_week IN ('Saturday', 'Sunday') THEN 'Weekend'
    ELSE 'Semaine'
  END AS type_jour
FROM kibana_sample_data_ecommerce
LIMIT 20;
```

Exercice 11.3 : Compter les commandes par segment de panier

Corrigé :

```
SELECT
  CASE
    WHEN taxful_total_price < 50 THEN 'Petit panier'
    WHEN taxful_total_price < 100 THEN 'Panier moyen'
    ELSE 'Gros panier'
  END AS segment,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS montant_moyen
FROM kibana_sample_data_ecommerce
GROUP BY
  CASE
    WHEN taxful_total_price < 50 THEN 'Petit panier'
    WHEN taxful_total_price < 100 THEN 'Panier moyen'
```

```
    ELSE 'Gros panier'
  END
ORDER BY montant_moyen DESC;
```

Exercice 11.4 : Traduire les noms de continents en français

Corrigé :

```
SELECT
  order_id,
  geoip.continent_name,
  CASE geoip.continent_name
    WHEN 'Europe' THEN 'Europe'
    WHEN 'Asia' THEN 'Asie'
    WHEN 'North America' THEN 'Amérique du Nord'
    WHEN 'South America' THEN 'Amérique du Sud'
    WHEN 'Africa' THEN 'Afrique'
    WHEN 'Oceania' THEN 'Océanie'
    ELSE 'Autre'
  END AS continent_fr
FROM kibana_sample_data_ecommerce
LIMIT 15;
```

Exercice 11.5 : Créer un score de valeur client (genre + montant)

Corrigé :

```
SELECT
  order_id,
  customer_gender,
  taxful_total_price,
  CASE
    WHEN customer_gender = 'FEMALE' AND taxful_total_price > 150 THEN 'A - Femme
Premium'
    WHEN customer_gender = 'FEMALE' AND taxful_total_price > 80 THEN 'B - Femme
Standard'
    WHEN customer_gender = 'MALE' AND taxful_total_price > 150 THEN 'A - Homme
Premium'
    WHEN customer_gender = 'MALE' AND taxful_total_price > 80 THEN 'B - Homme
Standard'
    ELSE 'C - Basique'
  END AS segment_valeur
FROM kibana_sample_data_ecommerce
LIMIT 20;
```

TD 12 : Sous-Requêtes (Subqueries)

Objectif

Utiliser les résultats d'une requête dans une autre requête.

Sous-Requête dans WHERE

```
SELECT
  order_id,
  taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE taxful_total_price > (
  SELECT AVG(taxful_total_price)
  FROM kibana_sample_data_ecommerce
)
LIMIT 10;
```

Explication

1. La sous-requête (entre parenthèses) est exécutée en premier
2. Elle retourne une valeur unique (ici la moyenne)
3. Cette valeur est utilisée dans la requête principale
4. Les commandes supérieures à la moyenne sont retournées

Sous-Requête avec IN

```
SELECT
  order_id,
  customer_id,
  taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE customer_id IN (
  SELECT customer_id
  FROM kibana_sample_data_ecommerce
  WHERE taxful_total_price > 200
)
LIMIT 20;
```

Sous-Requête dans SELECT

```
SELECT
  order_id,
  taxful_total_price,
  (SELECT AVG(taxful_total_price) FROM kibana_sample_data_ecommerce) AS
moyenne_generale,
```

```
  taxful_total_price - (SELECT AVG(taxful_total_price) FROM
kibana_sample_data_ecommerce) AS ecart_moyenne
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

Sous-Requête dans FROM (Table Dérivée)

```
SELECT
  segment,
  AVG(nb_commandes) AS moyenne_commandes_segment
FROM (
  SELECT
    customer_id,
    CASE
      WHEN COUNT(*) < 3 THEN 'Occasionnel'
      ELSE 'Régulier'
    END AS segment,
    COUNT(*) AS nb_commandes
  FROM kibana_sample_data_ecommerce
  GROUP BY customer_id
) AS segments_clients
GROUP BY segment;
```

Sous-Requête Corrélée

```
SELECT
  e1.order_id,
  e1.customer_id,
  e1.taxful_total_price
FROM kibana_sample_data_ecommerce e1
WHERE e1.taxful_total_price > (
  SELECT AVG(e2.taxful_total_price)
  FROM kibana_sample_data_ecommerce e2
  WHERE e2.customer_gender = e1.customer_gender
)
LIMIT 15;
```

Bonnes Pratiques

- Utiliser des sous-requêtes pour des calculs complexes
- Préférer les JOIN aux sous-requêtes corrélées (performances)
- Toujours donner des alias aux tables dérivées
- Limiter la profondeur d'imbrication (max 2-3 niveaux)

Erreurs à Éviter

- Sous-requête retournant plusieurs valeurs avec opérateur d'égalité

- Oublier les alias pour les tables dérivées
- Sous-requêtes corrélées sur gros volumes (très lent)
- Ne pas tester la sous-requête indépendamment

Exercices Supplémentaires TD 12

Exercice 12.1 : Trouver les commandes avec un montant supérieur au panier moyen en EUR

Corrigé :

```
SELECT
  order_id,
  currency,
  taxful_total_price
FROM kibana_sample_data_ecommerce
WHERE currency = 'EUR'
  AND taxful_total_price > (
    SELECT AVG(taxful_total_price)
    FROM kibana_sample_data_ecommerce
    WHERE currency = 'EUR'
  )
LIMIT 15;
```

Exercice 12.2 : Lister les clients ayant passé une commande > 150

Corrigé :

```
SELECT DISTINCT
  customer_id,
  customer_full_name
FROM kibana_sample_data_ecommerce
WHERE customer_id IN (
  SELECT customer_id
  FROM kibana_sample_data_ecommerce
  WHERE taxful_total_price > 150
)
LIMIT 20;
```

Exercice 12.3 : Afficher l'écart de chaque commande par rapport à la moyenne de son continent

Corrigé :

```
SELECT
  e1.order_id,
  e1.geoip.continent_name,
  e1.taxful_total_price,
  (SELECT AVG(e2.taxful_total_price)
   FROM kibana_sample_data_ecommerce e2
   WHERE e2.geoip.continent_name = e1.geoip.continent_name) AS moyenne_continent,
  e1.taxful_total_price -
```

```
  (SELECT AVG(e2.taxful_total_price)
   FROM kibana_sample_data_ecommerce e2
   WHERE e2.geoip.continent_name = e1.geoip.continent_name) AS ecart
FROM kibana_sample_data_ecommerce e1
LIMIT 10;
```

Exercice 12.4 : Trouver les pays avec un CA supérieur à la moyenne mondiale

Corrigé :

```
SELECT
  geoip.country_iso_code,
  SUM(taxful_total_price) AS ca_pays
FROM kibana_sample_data_ecommerce
GROUP BY geoip.country_iso_code
HAVING SUM(taxful_total_price) > (
  SELECT AVG(ca_total)
  FROM (
    SELECT SUM(taxful_total_price) AS ca_total
    FROM kibana_sample_data_ecommerce
    GROUP BY geoip.country_iso_code
  ) AS ca_par_pays
);
```

Exercice 12.5 : Calculer le percentile de chaque commande par rapport au total

Corrigé :

```
SELECT
  order_id,
  taxful_total_price,
  (SELECT AVG(taxful_total_price) FROM kibana_sample_data_ecommerce) AS moyenne,
  (SELECT MAX(taxful_total_price) FROM kibana_sample_data_ecommerce) AS maximum,
  ROUND(taxful_total_price * 100.0 / (SELECT MAX(taxful_total_price) FROM
kibana_sample_data_ecommerce), 2) AS pourcentage_max
FROM kibana_sample_data_ecommerce
LIMIT 10;
```

TD 13 : DISTINCT et Dédoublonnage

Objectif

Éliminer les doublons et compter les valeurs uniques.

DISTINCT Simple

```
SELECT DISTINCT customer_gender
```

```
FROM kibana_sample_data_ecommerce;
```

Explication

1. **DISTINCT** : élimine les doublons dans les résultats
2. Retourne uniquement les valeurs uniques
3. S'applique à l'ensemble des colonnes SELECT
4. Peut impacter les performances sur gros volumes

DISTINCT sur Plusieurs Colonnes

```
SELECT DISTINCT
  customer_gender,
  currency
FROM kibana_sample_data_ecommerce;
```

COUNT DISTINCT - Compter les Valeurs Uniques

```
SELECT
  COUNT(DISTINCT customer_id) AS nb_clients_uniques,
  COUNT(*) AS nb_commandes_total
FROM kibana_sample_data_ecommerce;
```

DISTINCT avec GROUP BY

```
SELECT
  day_of_week,
  COUNT(DISTINCT customer_id) AS nb_clients_uniques,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY day_of_week
ORDER BY nb_clients_uniques DESC;
```

Plusieurs COUNT DISTINCT

```
SELECT
  COUNT(DISTINCT customer_id) AS clients_uniques,
  COUNT(DISTINCT geoip.country_iso_code) AS pays_uniques,
  COUNT(DISTINCT currency) AS devises_uniques
FROM kibana_sample_data_ecommerce;
```

DISTINCT avec Filtres

```
SELECT DISTINCT
  geoip.continent_name,
```

```
  geoip.country_iso_code
FROM kibana_sample_data_ecommerce
WHERE geoip.continent_name = 'Europe'
ORDER BY geoip.country_iso_code;
```

Bonnes Pratiques

- Utiliser DISTINCT uniquement quand nécessaire (coût performance)
- Préférer GROUP BY pour des agrégations en même temps
- COUNT(DISTINCT) pour mesurer la cardinalité
- Combiner avec ORDER BY pour un résultat lisible

Erreurs à Éviter

- Utiliser DISTINCT sans comprendre pourquoi il y a des doublons
- DISTINCT sur trop de colonnes (peu de dédoublonnage)
- Oublier que DISTINCT s'applique à toutes les colonnes SELECT
- Sur-utilisation sur de gros volumes (préférer GROUP BY)

Exercices Supplémentaires TD 13

Exercice 13.1 : Lister tous les pays présents dans les données

Corrigé :

```
SELECT DISTINCT geoip.country_iso_code
FROM kibana_sample_data_ecommerce
ORDER BY geoip.country_iso_code;
```

Exercice 13.2 : Compter le nombre de clients uniques par continent

Corrigé :

```
SELECT
  geoip.continent_name,
  COUNT(DISTINCT customer_id) AS clients_uniques
FROM kibana_sample_data_ecommerce
GROUP BY geoip.continent_name
ORDER BY clients_uniques DESC;
```

Exercice 13.3 : Trouver toutes les combinaisons genre/devise

Corrigé :

```
SELECT DISTINCT
```

```
  customer_gender,
  currency
FROM kibana_sample_data_ecommerce
ORDER BY customer_gender, currency;
```

Exercice 13.4 : Statistiques de diversité par jour de semaine

Corrigé :

```
SELECT
  day_of_week,
  COUNT(DISTINCT customer_id) AS clients_uniques,
  COUNT(DISTINCT geoip.country_iso_code) AS pays_uniques,
  COUNT(*) AS total_commandes
FROM kibana_sample_data_ecommerce
GROUP BY day_of_week
ORDER BY clients_uniques DESC;
```

Exercice 13.5 : Nombre de clients uniques par tranche de panier

Corrigé :

```
SELECT
  CASE
    WHEN taxful_total_price < 50 THEN 'Petit'
    WHEN taxful_total_price < 100 THEN 'Moyen'
    ELSE 'Gros'
  END AS segment_panier,
  COUNT(DISTINCT customer_id) AS clients_uniques,
  COUNT(*) AS nb_commandes
FROM kibana_sample_data_ecommerce
GROUP BY
  CASE
    WHEN taxful_total_price < 50 THEN 'Petit'
    WHEN taxful_total_price < 100 THEN 'Moyen'
    ELSE 'Gros'
  END;
```

TD 14 : LIKE et Recherche de Patterns

Objectif

Rechercher des motifs dans les chaînes de caractères.

LIKE avec Wildcards

```
SELECT
  order_id,
```

```
  customer_full_name
FROM kibana_sample_data_ecommerce
WHERE customer_full_name LIKE '%Smith%'
LIMIT 10;
```

Explication

1. **LIKE** : opérateur de recherche de motifs
2. **%** : représente zéro ou plusieurs caractères
3. **_** : représente exactement un caractère
4. Sensible à la casse selon la configuration

Patterns Courants

```
-- Commence par
WHERE customer_full_name LIKE 'John%'

-- Se termine par
WHERE customer_full_name LIKE '%son'

-- Contient
WHERE customer_full_name LIKE '%man%'

-- Deuxième lettre est 'a'
WHERE customer_full_name LIKE '_a%'
```

LIKE avec Plusieurs Conditions

```
SELECT
  order_id,
  customer_full_name,
  email
FROM kibana_sample_data_ecommerce
WHERE customer_full_name LIKE '%Smith%'
   OR customer_full_name LIKE '%Johnson%'
LIMIT 15;
```

NOT LIKE - Exclusion de Patterns

```
SELECT
  order_id,
  email
FROM kibana_sample_data_ecommerce
WHERE email NOT LIKE '%gmail.com%'
LIMIT 10;
```

LIKE avec Agrégations

```
SELECT
  CASE
    WHEN email LIKE '%gmail.com%' THEN 'Gmail'
    WHEN email LIKE '%yahoo.com%' THEN 'Yahoo'
    ELSE 'Autre'
  END AS fournisseur_email,
  COUNT(*) AS nb_clients
FROM kibana_sample_data_ecommerce
GROUP BY
  CASE
    WHEN email LIKE '%gmail.com%' THEN 'Gmail'
    WHEN email LIKE '%yahoo.com%' THEN 'Yahoo'
    ELSE 'Autre'
  END;
```

Bonnes Pratiques

- Utiliser LIKE pour des recherches flexibles sur du texte
- Éviter % au début de la chaîne (impact performance - pas d'index)
- Combiner avec UPPER/LOWER pour recherche insensible à la casse
- Préférer les champs keyword pour LIKE

Erreurs à Éviter

- Utiliser LIKE sur des champs non textuels
- Oublier les % pour les recherches partielles
- Pattern trop large (%%) qui retourne tout
- Performance : LIKE '%term%' est plus lent que 'term%'

Exercices Supplémentaires TD 14

Exercice 14.1 : Trouver les clients dont le nom commence par 'M'

Corrigé :

```
SELECT
  order_id,
  customer_full_name
FROM kibana_sample_data_ecommerce
WHERE customer_full_name LIKE 'M%'
LIMIT 20;
```

Exercice 14.2 : Trouver les emails se terminant par '.com'

Corrigé :

```
SELECT DISTINCT
  email
FROM kibana_sample_data_ecommerce
WHERE email LIKE '%.com'
LIMIT 15;
```

Exercice 14.3 : Compter les clients par domaine email principal

Corrigé :

```
SELECT
  CASE
    WHEN email LIKE '%gmail%' THEN 'Gmail'
    WHEN email LIKE '%yahoo%' THEN 'Yahoo'
    WHEN email LIKE '%hotmail%' THEN 'Hotmail'
    ELSE 'Autre'
  END AS domaine,
  COUNT(*) AS nb_clients
FROM kibana_sample_data_ecommerce
GROUP BY
  CASE
    WHEN email LIKE '%gmail%' THEN 'Gmail'
    WHEN email LIKE '%yahoo%' THEN 'Yahoo'
    WHEN email LIKE '%hotmail%' THEN 'Hotmail'
    ELSE 'Autre'
  END
ORDER BY nb_clients DESC;
```

Exercice 14.4 : Trouver les noms contenant exactement 5 caractères avant un espace

Corrigé :

```
SELECT
  customer_full_name
FROM kibana_sample_data_ecommerce
WHERE customer_full_name LIKE '_____ %'
LIMIT 10;
```

Exercice 14.5 : Exclure les emails des domaines .net et .org

Corrigé :

```
SELECT
  customer_full_name,
  email
FROM kibana_sample_data_ecommerce
WHERE email NOT LIKE '%.net'
  AND email NOT LIKE '%.org'
LIMIT 20;
```

TD 15 : Requêtes Avancées - Analyse Complète

Objectif

Combiner toutes les techniques apprises pour des analyses business complexes.

Analyse RFM (Récence, Fréquence, Montant)

```
SELECT
  customer_id,
  customer_full_name,
  COUNT(*) AS frequence,
  SUM(taxful_total_price) AS montant_total,
  AVG(taxful_total_price) AS panier_moyen,
  MAX(order_date) AS derniere_commande
FROM kibana_sample_data_ecommerce
GROUP BY customer_id, customer_full_name
HAVING COUNT(*) > 1
ORDER BY montant_total DESC
LIMIT 20;
```

Explication

1. Regroupe par client pour analyser son comportement
2. Calcule plusieurs métriques : fréquence, montant total, panier moyen
3. Filtre les clients avec plusieurs commandes (HAVING)
4. Trie par valeur client décroissante

Analyse de Cohorte par Mois

```
SELECT
  YEAR(order_date) AS annee,
  MONTH(order_date) AS mois,
  COUNT(DISTINCT customer_id) AS clients_actifs,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  SUM(total_quantity) AS articles_vendus
FROM kibana_sample_data_ecommerce
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY annee DESC, mois DESC;
```

Segmentation Client Multi-Critères

```
SELECT
  CASE
    WHEN nb_commandes >= 3 AND panier_moyen > 100 THEN 'VIP'
    WHEN nb_commandes >= 3 THEN 'Fidèle'
    WHEN panier_moyen > 100 THEN 'Premium'
    ELSE 'Standard'
  END AS segment_client,
  COUNT(*) AS nb_clients,
  AVG(panier_moyen) AS panier_moyen_segment,
  SUM(ca_total) AS ca_segment
FROM (
  SELECT
    customer_id,
    COUNT(*) AS nb_commandes,
    AVG(taxful_total_price) AS panier_moyen,
    SUM(taxful_total_price) AS ca_total
  FROM kibana_sample_data_ecommerce
  GROUP BY customer_id
) AS stats_clients
GROUP BY
  CASE
    WHEN nb_commandes >= 3 AND panier_moyen > 100 THEN 'VIP'
    WHEN nb_commandes >= 3 THEN 'Fidèle'
    WHEN panier_moyen > 100 THEN 'Premium'
    ELSE 'Standard'
  END
ORDER BY ca_segment DESC;
```

Analyse Géographique Complète

```
SELECT
  geoip.continent_name,
  geoip.country_iso_code,
  COUNT(DISTINCT customer_id) AS clients_uniques,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  SUM(total_quantity) AS articles_vendus,
  ROUND(SUM(taxful_total_price) * 100.0 / (SELECT SUM(taxful_total_price) FROM
kibana_sample_data_ecommerce), 2) AS part_ca_pct
FROM kibana_sample_data_ecommerce
GROUP BY geoip.continent_name, geoip.country_iso_code
HAVING COUNT(*) > 50
ORDER BY ca_total DESC
LIMIT 15;
```

Performance par Jour et Devise

```
SELECT
```

```
  day_of_week,
  currency,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  MIN(taxful_total_price) AS min_commande,
  MAX(taxful_total_price) AS max_commande
FROM kibana_sample_data_ecommerce
GROUP BY day_of_week, currency
HAVING COUNT(*) > 100
ORDER BY day_of_week, ca_total DESC;
```

Top Clients par Continent

```
SELECT
  geoip.continent_name,
  customer_id,
  customer_full_name,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_client
FROM kibana_sample_data_ecommerce
WHERE geoip.continent_name IN ('Europe', 'North America', 'Asia')
GROUP BY geoip.continent_name, customer_id, customer_full_name
HAVING COUNT(*) >= 2
ORDER BY geoip.continent_name, ca_client DESC;
```

Analyse de Panier - Distribution

```
SELECT
  CASE
    WHEN total_quantity = 1 THEN '1 article'
    WHEN total_quantity = 2 THEN '2 articles'
    WHEN total_quantity BETWEEN 3 AND 5 THEN '3-5 articles'
    ELSE '6+ articles'
  END AS taille_panier,
  COUNT(*) AS nb_commandes,
  AVG(taxful_total_price) AS montant_moyen,
  SUM(taxful_total_price) AS ca_total,
  ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM kibana_sample_data_ecommerce), 2) AS
pct_commandes
FROM kibana_sample_data_ecommerce
GROUP BY
  CASE
    WHEN total_quantity = 1 THEN '1 article'
    WHEN total_quantity = 2 THEN '2 articles'
    WHEN total_quantity BETWEEN 3 AND 5 THEN '3-5 articles'
    ELSE '6+ articles'
  END
ORDER BY nb_commandes DESC;
```

Bonnes Pratiques - Synthèse

- Commencer par une question business claire
- Construire la requête étape par étape
- Tester chaque partie séparément
- Utiliser des sous-requêtes pour des calculs complexes
- Optimiser avec des index sur les champs filtrés et groupés
- Documenter les requêtes complexes avec des commentaires
- Utiliser des vues pour les requêtes fréquentes

Erreurs à Éviter - Synthèse

- Requêtes trop complexes (diviser en étapes)
- Pas de LIMIT sur les tests (charge le cluster)
- Oublier les index sur les champs WHERE/GROUP BY
- Ne pas vérifier les NULL dans les agrégations
- Sous-requêtes corrélées sur gros volumes
- Pas de validation des résultats

Exercices Finaux Avancés TD 15

Exercice 15.1 : Créer un tableau de bord des KPIs globaux

Corrigé :

```
SELECT
  COUNT(*) AS total_commandes,
  COUNT(DISTINCT customer_id) AS clients_uniques,
  COUNT(DISTINCT geoip.country_iso_code) AS pays_couverts,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  SUM(total_quantity) AS articles_vendus,
  AVG(total_quantity) AS articles_par_commande,
  MIN(order_date) AS premiere_commande,
  MAX(order_date) AS derniere_commande
FROM kibana_sample_data_ecommerce;
```

Exercice 15.2 : Identifier les meilleurs jours par devise

Corrigé :

```
SELECT
  currency,
  day_of_week,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_jour
```

```
FROM kibana_sample_data_ecommerce
GROUP BY currency, day_of_week
HAVING COUNT(*) > 50
ORDER BY currency, ca_jour DESC;
```

Exercice 15.3 : Analyse de la répartition géographique du CA par genre

Corrigé :

```
SELECT
  geoip.continent_name,
  customer_gender,
  COUNT(*) AS nb_commandes,
  SUM(taxful_total_price) AS ca_total,
  AVG(taxful_total_price) AS panier_moyen,
  COUNT(DISTINCT customer_id) AS clients_uniques
FROM kibana_sample_data_ecommerce
GROUP BY geoip.continent_name, customer_gender
HAVING COUNT(*) > 100
ORDER BY geoip.continent_name, ca_total DESC;
```

Exercice 15.4 : Classer les clients en déciles de dépenses

Corrigé :

```
SELECT
  CASE
    WHEN ca_client < (SELECT AVG(ca) * 0.5 FROM (SELECT SUM(taxful_total_price) AS ca
FROM kibana_sample_data_ecommerce GROUP BY customer_id) AS t) THEN 'D1-D5 Bas'
    WHEN ca_client < (SELECT AVG(ca) * 1.5 FROM (SELECT SUM(taxful_total_price) AS ca
FROM kibana_sample_data_ecommerce GROUP BY customer_id) AS t) THEN 'D6-D8 Moyen'
    ELSE 'D9-D10 Haut'
  END AS decile,
  COUNT(*) AS nb_clients,
  AVG(ca_client) AS ca_moyen,
  SUM(ca_client) AS ca_total_segment
FROM (
  SELECT
    customer_id,
    SUM(taxful_total_price) AS ca_client
  FROM kibana_sample_data_ecommerce
  GROUP BY customer_id
) AS clients_ca
GROUP BY
  CASE
    WHEN ca_client < (SELECT AVG(ca) * 0.5 FROM (SELECT SUM(taxful_total_price) AS ca
FROM kibana_sample_data_ecommerce GROUP BY customer_id) AS t) THEN 'D1-D5 Bas'
    WHEN ca_client < (SELECT AVG(ca) * 1.5 FROM (SELECT SUM(taxful_total_price) AS ca
FROM kibana_sample_data_ecommerce GROUP BY customer_id) AS t) THEN 'D6-D8 Moyen'
    ELSE 'D9-D10 Haut'
  END
ORDER BY ca_total_segment DESC;
```

Exercice 15.5 : Rapport mensuel complet avec évolution

Corrigé :

```
SELECT
  YEAR(order_date) AS annee,
  MONTH(order_date) AS mois,
  COUNT(*) AS nb_commandes,
  COUNT(DISTINCT customer_id) AS clients_actifs,
  SUM(taxful_total_price) AS ca_mensuel,
  AVG(taxful_total_price) AS panier_moyen,
  SUM(total_quantity) AS articles_vendus,
  COUNT(DISTINCT geoip.country_iso_code) AS pays_actifs,
  ROUND(AVG(taxful_total_price) / AVG(total_quantity), 2) AS prix_moyen_article
FROM kibana_sample_data_ecommerce
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY annee DESC, mois DESC;
```

Conclusion et Bonnes Pratiques Générales

Ordre d'Exécution SQL (Rappel Important)

```
1. FROM         - Sélection de la table/index
2. WHERE        - Filtrage des lignes
3. GROUP BY     - Regroupement
4. Agrégations - COUNT, SUM, AVG, etc.
5. HAVING       - Filtrage des groupes
6. SELECT       - Projection des colonnes
7. DISTINCT     - Élimination des doublons
8. ORDER BY     - Tri
9. LIMIT        - Limitation des résultats
```