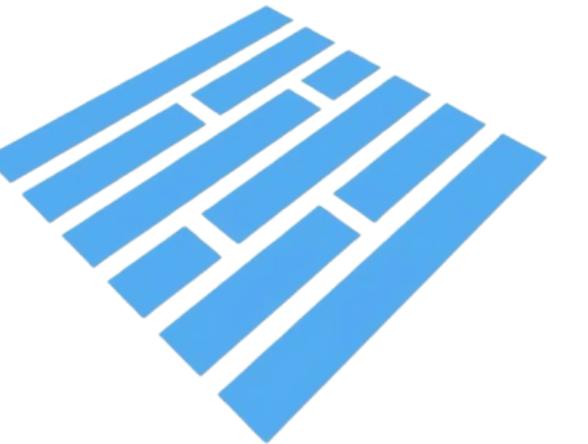




Apache Parquet



ENCADRÉ PAR:

- Pr. Lotfi Najdi

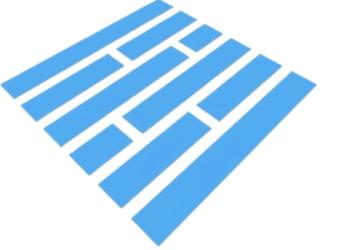
PRÉSENTÉ PAR:

- Roukmi Anas
- Rekbi Abdelkrim

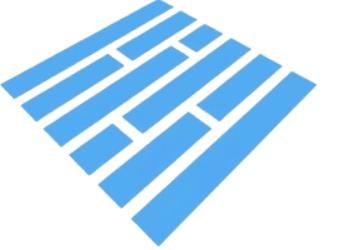
PLAN



- **Introduction**
- **Format Apache Parquet**
- **Importance de Parquet en BI**
- **Comparaison avec d'autres formats**
- **Cas d'usage**
- **Conclusion**



Introduction

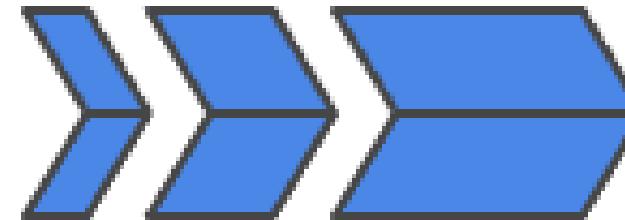


Dans un contexte où la Business Intelligence dépend fortement de la rapidité et de l'efficacité du traitement des données, le choix du format de stockage est déterminant. Apache Parquet, un format en colonnes, s'impose comme un standard moderne en permettant de réduire jusqu'à 70 % les coûts de stockage cloud et d'accélérer les requêtes analytiques par rapport aux formats traditionnels comme CSV ou JSON. Sa conception technique répond parfaitement aux besoins de la BI à grande échelle.

Adoption de Parquet dans la BI d'entreprise (2018-2024)



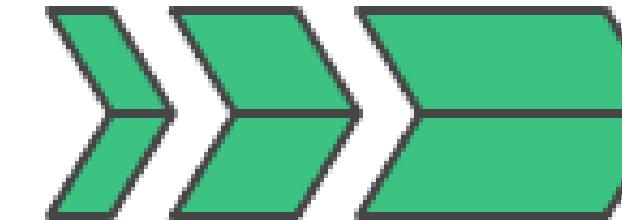
2018



Adoption précoce

Parquet commence à être utilisé par les entreprises pionnières en BI

2020



Croissance rapide

Parquet gagne en popularité et est adopté par de nombreuses entreprises

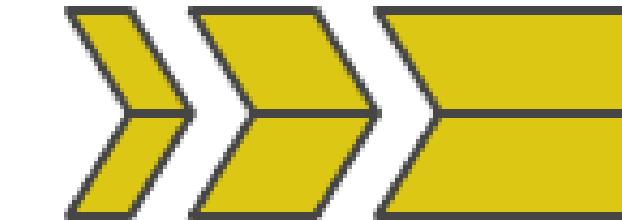
2022



Adoption généralisée

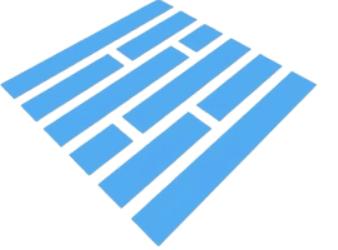
Parquet devient un format standard dans la plupart des pipelines BI

2024



Maturité

Parquet est largement intégré et considéré comme une technologie établie en BI

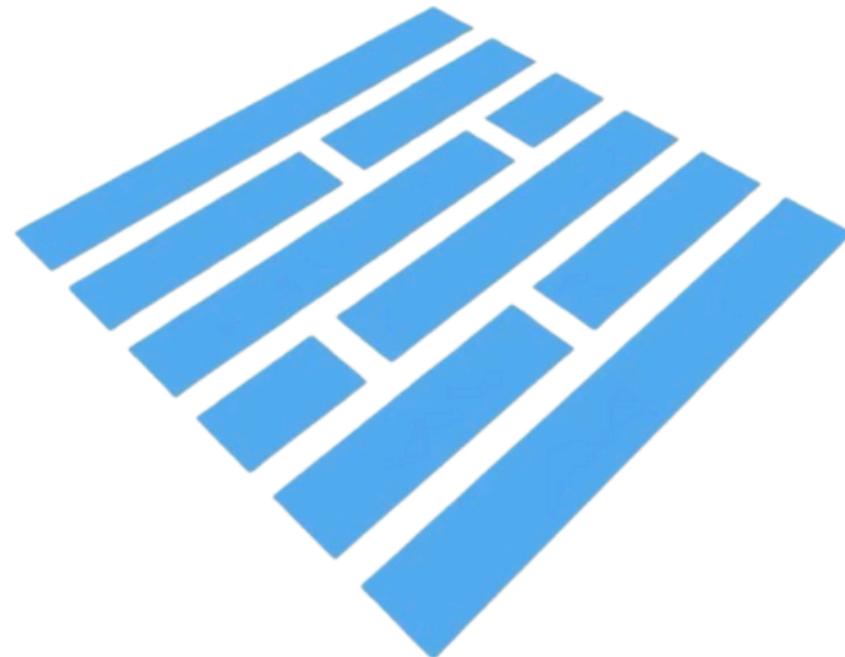


Format Apache Parquet

Qu'est-ce qu'Apache Parquet ?



Apache Parquet est un format open-source en colonnes, fait pour le Big Data. Il permet de lire uniquement les colonnes utiles, ce qui rend les requêtes plus rapides et économise des ressources.



Stockage en colonnes



Contrairement aux formats basés sur des lignes comme CSV, Parquet organise les données en colonnes. Ainsi, lors de l'exécution d'une requête, seules les colonnes nécessaires sont extraites au lieu de tout charger. Cela améliore les performances et réduit l'utilisation des E/S.

std_id	std_name	age	scores
101	Alex	20	65
102	James	21	45
103	Mike	24	78

Row-based

101	Alex	20	65
102	James	21	45
103	Mike	24	78

Column-based

101	102	103
Alex	James	Mike
20	21	24
65	43	78

Structure interne du fichier parquet

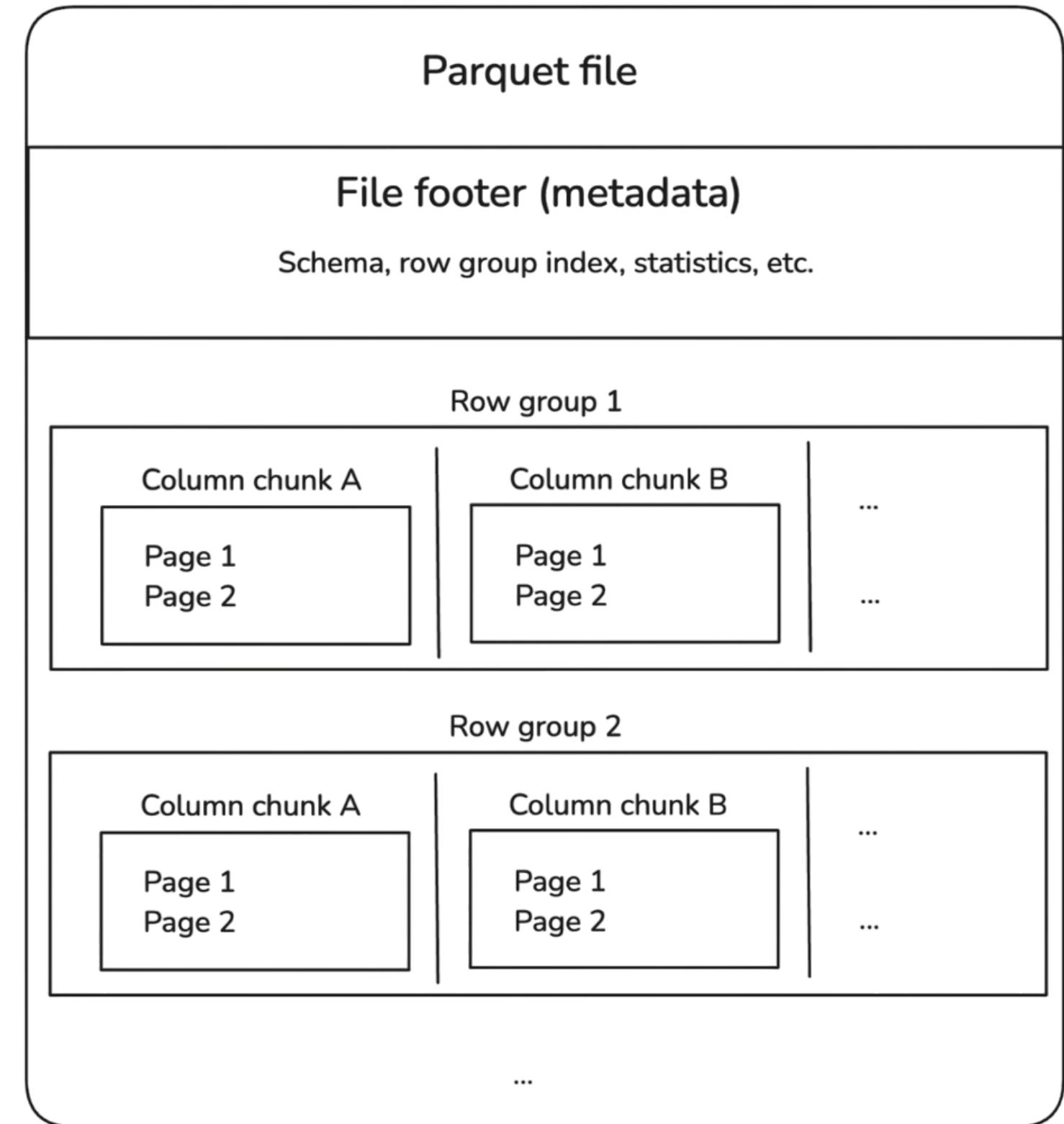


Groupes de lignes : Parquet divise les données en groupes, chacun contenant plusieurs lignes, mais stockées par colonnes pour une lecture plus rapide.

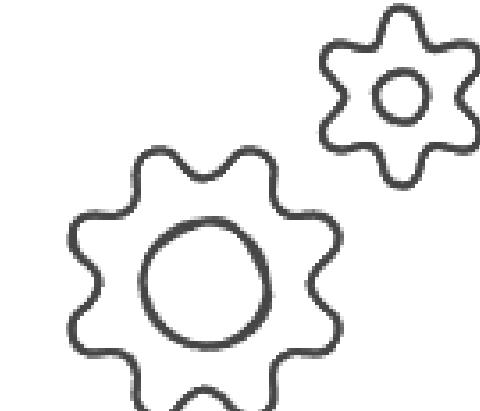
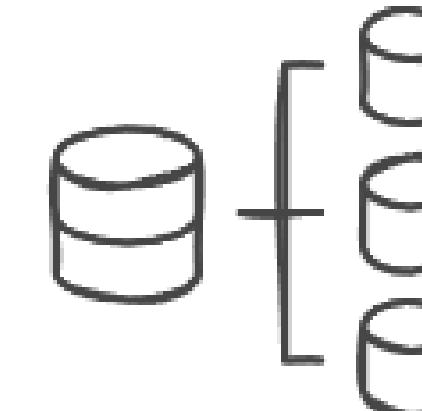
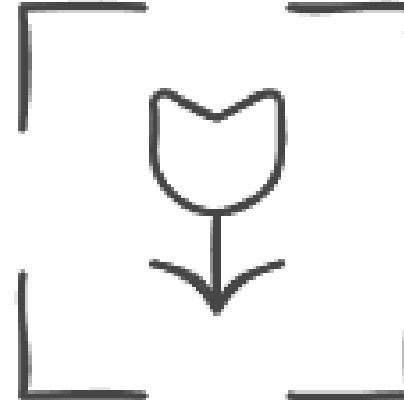
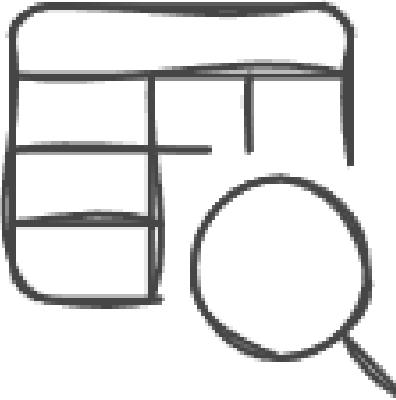
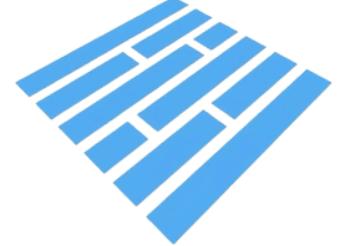
Blocs de colonnes : Dans chaque groupe, les données sont séparées par colonnes. Cela permet de lire seulement les colonnes nécessaires.

Pages : Chaque bloc est découpé en pages pour économiser la mémoire. Ces pages sont souvent compressées.

Pied de page (footer) : À la fin du fichier, on trouve des métadonnées (schéma, positions des groupes, statistiques) utiles pour accélérer les recherches et filtrer les données rapidement.



Aspects techniques



Statistiques au niveau de la page

Ignore les pages non pertinentes en utilisant des valeurs min/max.

1

Filtres Bloom

Permet un filtrage rapide sur des colonnes spécifiques.

2

Données imbriquées

Prend en charge efficacement des objets JSON complexes.

3

Évolution du schéma

Permet d'ajouter ou de supprimer des colonnes sans casser les requêtes.

4

Exemple statistiques au niveau de la page



Requête
Ventes après 2023-03-01



Base de données - Pages

Page A

2023-01-01 à 2023-01-10

Données janvier

FILTRÉ

Page B

2023-02-15 à 2023-02-28

Données février

FILTRÉ

Page C

2023-03-05 à 2023-03-20

Données mars

ACTIF

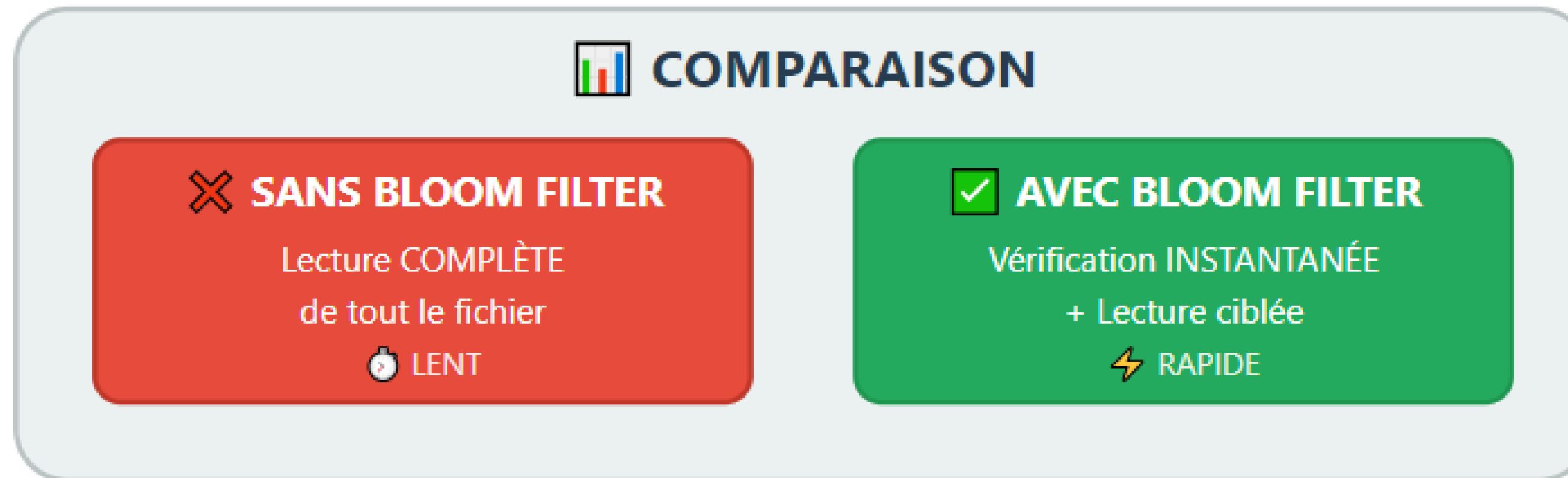
Page D

2023-04-01 à 2023-04-15

Données avril

ACTIF

Exemple filtres de Bloom



Exemple données imbriquées



✗ AVANT : JSON dans une seule colonne

ID	DATA (JSON)
1	{"client": {"nom": "Ali", "ville": "Fès"}, "commande": 1200}
2	{"client": {"nom": "Sara", "ville": "Rabat"}, "commande": 800}

⚠ Problème : Doit parser tout le JSON pour accéder à "ville"

↓ TRANSFORMATION

✓ APRÈS : Colonnes séparées (Columnar)

ID	client.nom	client.ville	commande
1	Ali	Fès	1200
2	Sara	Rabat	800

REQUÊTE

clientville = "Fès"

Accès direct à la colonne !

Exemple évolution du schéma

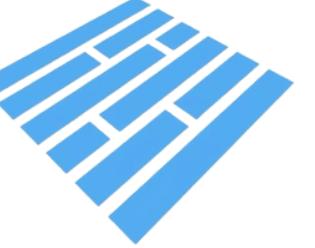


Id	name	age
1	Alex	20
2	James	21
3	Jacob	23

Id	name	salary
4	Mike	\$ 60,000
5	Bob	\$ 55,000



Id	name	age	salary
1	Alex	20	null
2	James	21	null
3	Jacob	23	null
4	Mike	null	\$ 60,000
5	Bob	null	\$ 55,000



Importance de Parquet en BI

Performance



Requêtes jusqu'à 10 fois plus rapides grâce au stockage colonnaire

- **Principe :**
 - **Column pruning** : Réduit le volume de données lues en ne chargeant que les colonnes nécessaires.
 - **Partitionnement** : Accélère les analyses sur de grands datasets en segmentant les données.

Exemple : Une agrégation sur 1 To prend des secondes avec Parquet.



Parquet

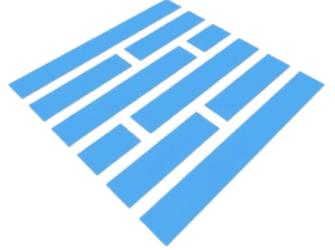


CSV

Plus rapide

Plus lent

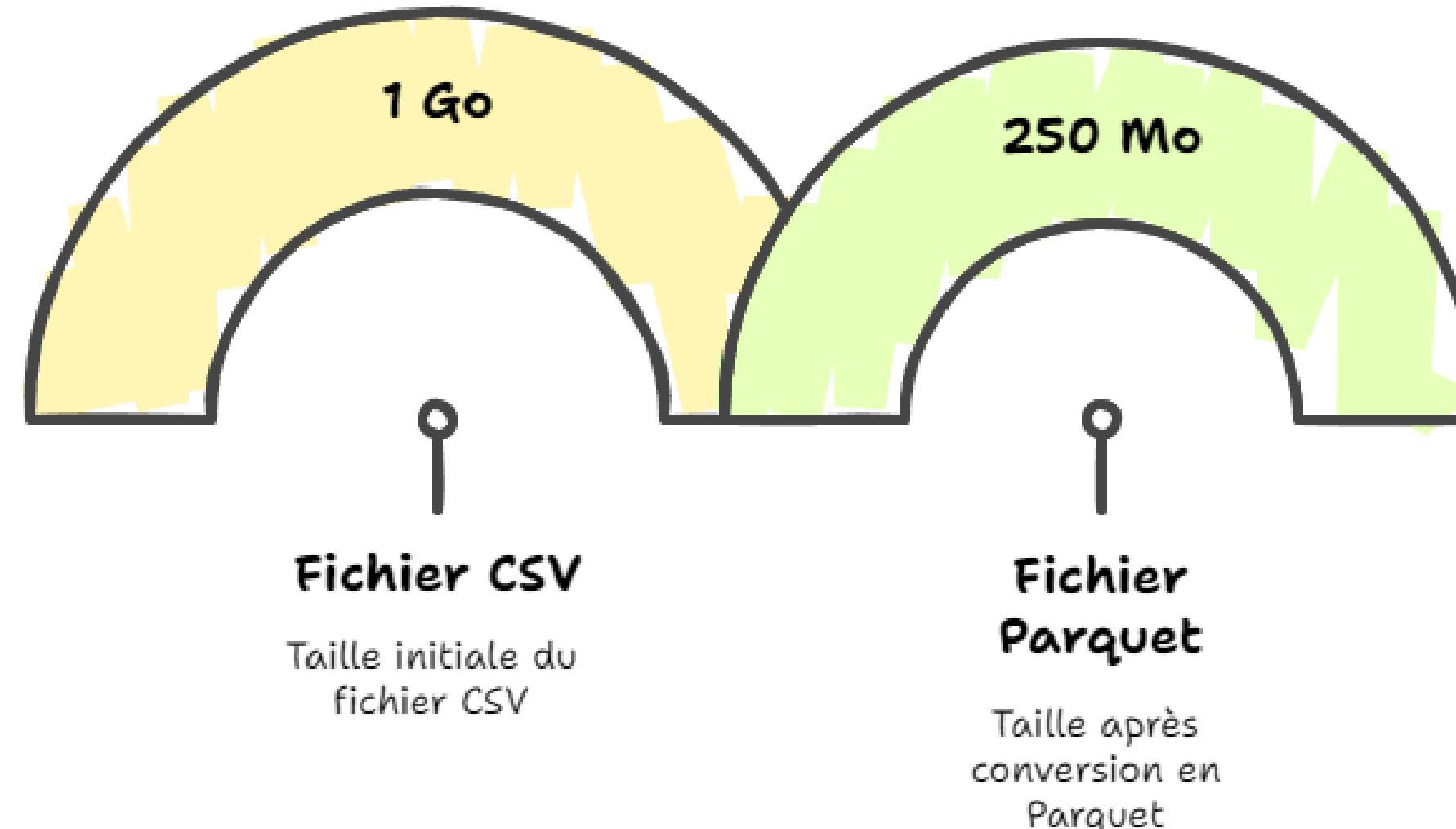
Compression



Réduction de la taille des fichiers (jusqu'à 75 % par rapport à CSV)

Avantages :

- Moins de stockage, entraînant une réduction des coûts dans le cloud.
- Transferts de données plus rapides grâce à la taille réduite.



Bénéfices quantifiés de Parquet en BI



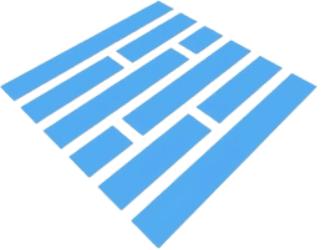
Réduction I/O (jusqu'à 90 %)

- **Principe** : on ne lit que les colonnes et blocs de données nécessaires, pas tout le fichier.
- **Impact** : sur un dataset de 100 Go, on ne lit souvent que 10 Go au lieu de 100 Go.

Réduction des transferts réseau (de 3x à 5x)

- **Principe** : Fichiers compressés et lecture partielle des données.
- **Impact** : Un fichier CSV de 50 Go est réduit à 10 Go en Parquet, diminuant ainsi la bande passante utilisée jusqu'à 5 fois.

Bénéfices quantifiés de Parquet en BI



Réduction de l'empreinte mémoire (de 50 % à 80 %)

- **Principe** : compression par colonne et traitement par blocs .
- **Impact** : un job Spark utilisera la moitié (voire moins) de la RAM nécessaire avec un format non-colonne.

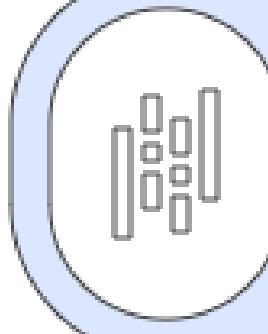
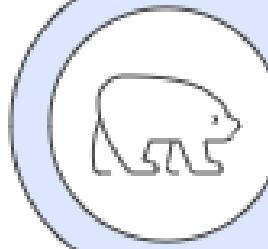
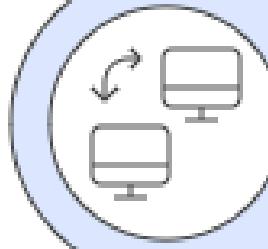
Économies sur le stockage cloud (de 70 % à 80 % des coûts)

- **Principe** : taille disque divisée par 4 à 5 grâce à la compression Parquet.
- **Impact** : **1 To** de données CSV, coûtant **23 \$/mois** à stocker, passe à **200 Go** une fois compressé, soit un coût de **4,6 \$/mois**.

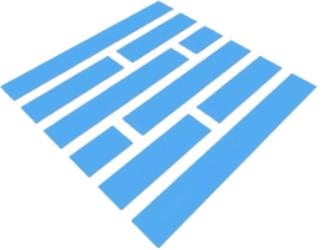
Intégration facile avec les outils BI modernes



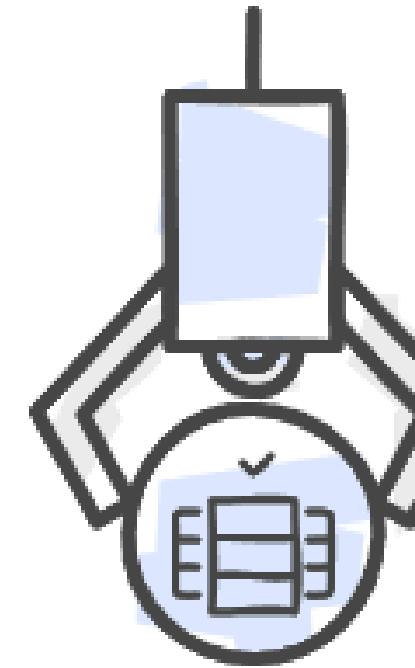
- Écosystème Jupyter / Python

- Pandas**
Lecture et écriture faciles de fichiers Parquet.
Utilise les fonctions `read_parquet` et `to_parquet`.
- Polars**
Traitement extrêmement rapide pour les gros fichiers.
- Dask**
Traitement parallèle de très grands ensembles de données.

Intégration facile avec les outils BI modernes

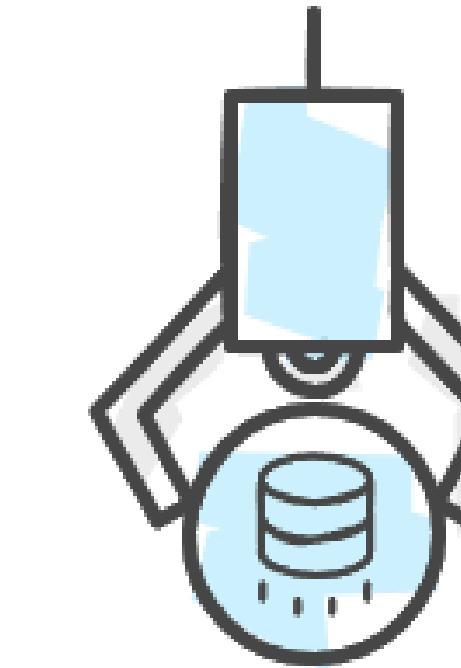


- Outils cloud BI (cloud-native)



AWS Glue

Lire et transformer
des fichiers Parquet



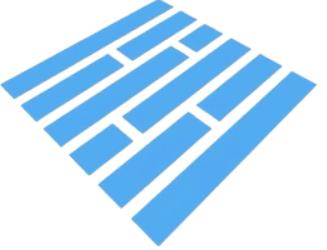
**Azure Data
Factory**

Déplacer des
données Parquet
entre différentes
sources

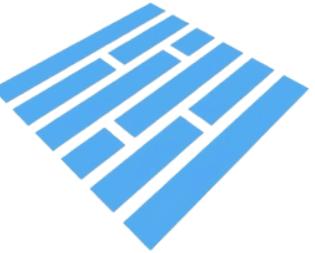


**Google Cloud
Dataflow**

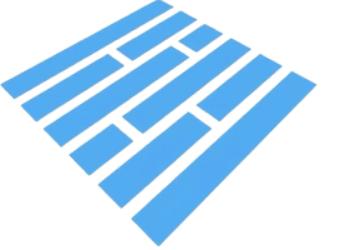
Traitements
distribué de
fichiers Parquet



Comparaison avec d'autres formats

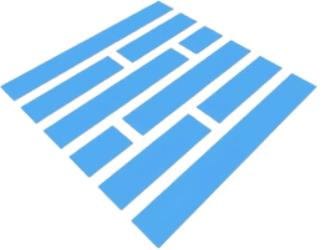


Critère	Parquet	CSV	JSON
Type de stockage	Colonnes	Lignes	Lignes (hiérarchique)
Compression	Excellente (Snappy, Gzip, ect.)	Faible (fichiers bruts)	Moyenne
Vitesse lecture BI	Très rapide (colonnes utiles seulement)	Lente (lit tout)	Lente
Taille des fichiers	Petite	Très grande	Grande
Lecture selective	Oui (colonnes + filtres)	Non	Très limitée
Utilisation en cloud	Optimisé (Athena, BigQuery...)	Supporté mais coûteux	Supporté mais moins optimisé
Évolutivité (schema)	Supporte l'évolution	Non	Non



Cas d'usage

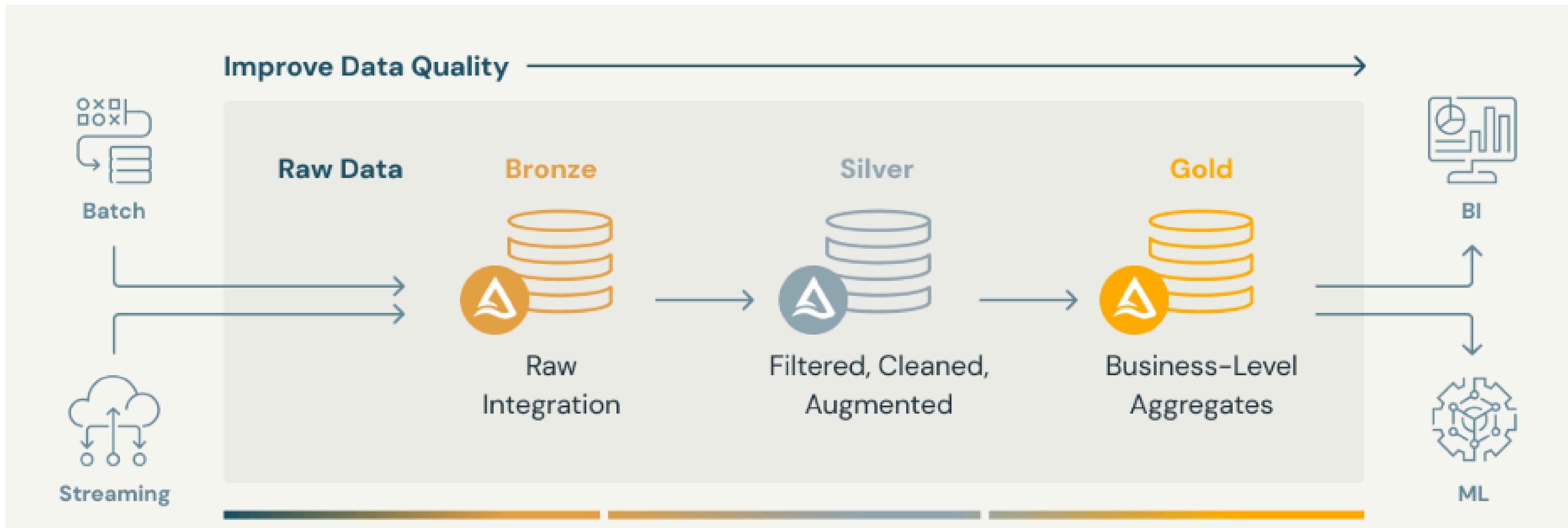
Architecture Medallion (Bronze / Silver / Gold)



Concept : C'est une approche en couches pour organiser les données dans un data lake ou data lakehouse.

- **Bronze** : Données brutes et non transformées, souvent conservées dans leur format original (JSON, CSV).
- **Silver** : Données nettoyées, filtrées et partiellement transformées, généralement stockées au format Parquet avec partitionnement.
- **Gold** : Données prêtes pour la consommation BI, agrégées et optimisées, toujours au format Parquet.

Architecture Medallion (Bronze / Silver / Gold)



Chargement incrémental



Problème : Les données arrivent tous les jours ou toutes les heures. Recharger toute la table serait trop long et coûteux.

Solution avec Parquet :

- Les fichiers sont partitionnés par date (par exemple : year=2025/month=05/day=29/).
- Lors des mises à jour, seuls les nouveaux fichiers sont ajoutés ou modifiés.
- Cela permet des opérations efficaces comme l'ajout (append), la mise à jour (update) ou la fusion (merge) sans toucher aux anciens fichiers.

BI multi-tenant



Défi : Plusieurs clients ou départements utilisent le même tableau de bord, mais chacun doit voir uniquement ses propres données.

Solution avec Parquet :

- Les fichiers sont partitionnés par client (ex. : **client_id=123/**).
- Le moteur BI lit seulement les partitions utiles (**partition pruning**).
- En combinant cela avec des règles de sécurité (Row-Level Security), l’isolation des données est garantie.

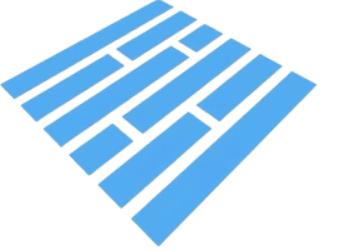
Streaming analytics



Contexte : Dans le traitement de données en quasi temps réel, les données arrivent en continu et sont traitées par petits lots (micro-batches), par exemple toutes les 5 minutes.

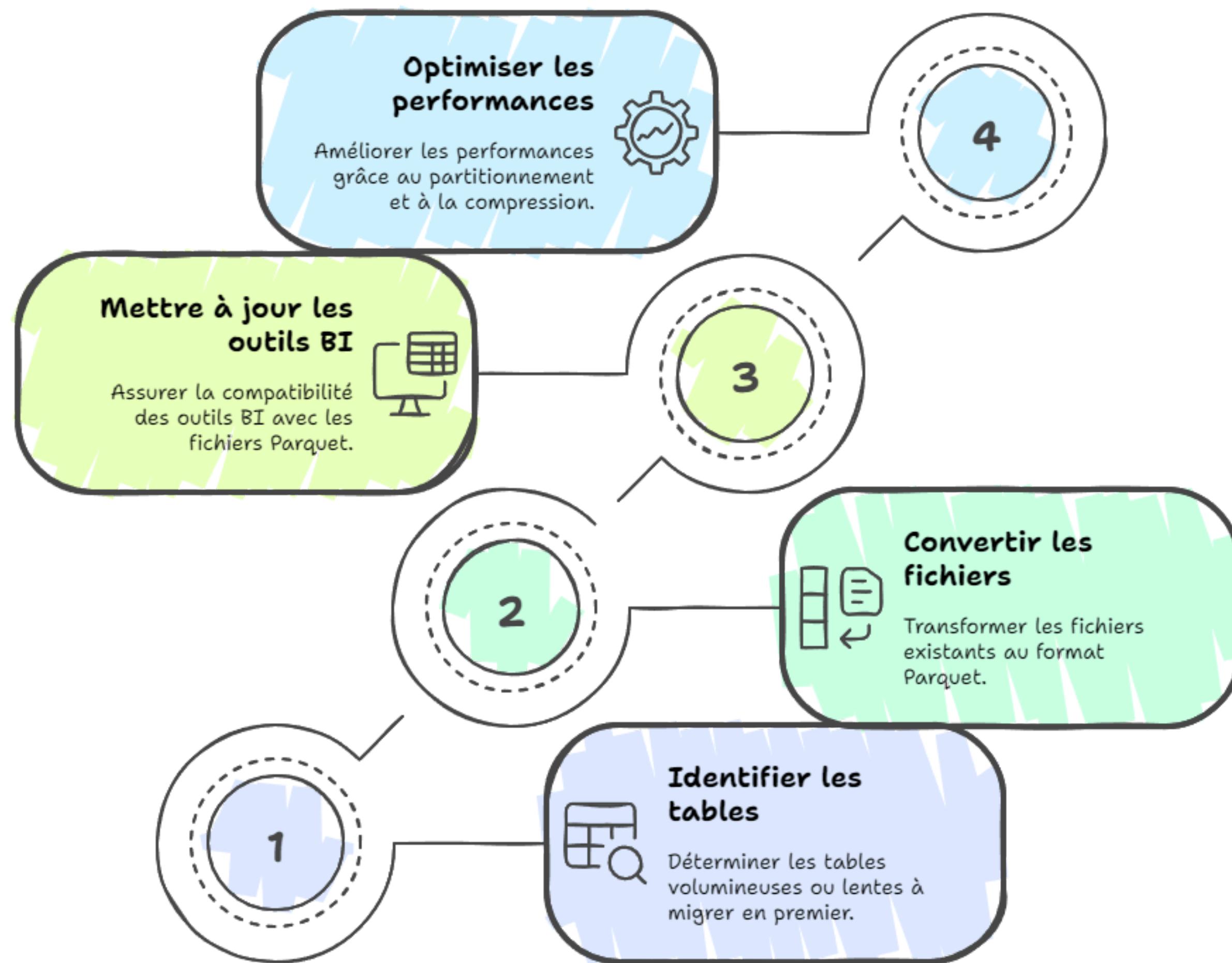
Solution avec Parquet :

- Chaque **micro-batch** est enregistré sous forme de fichier Parquet, partitionné par date et heure.
- Parquet conserve des statistiques et métadonnées, ce qui accélère les analyses.
- Des moteurs comme **Spark Streaming** ou **Apache Flink** exploitent Parquet pour lire et écrire efficacement ces flux de données.

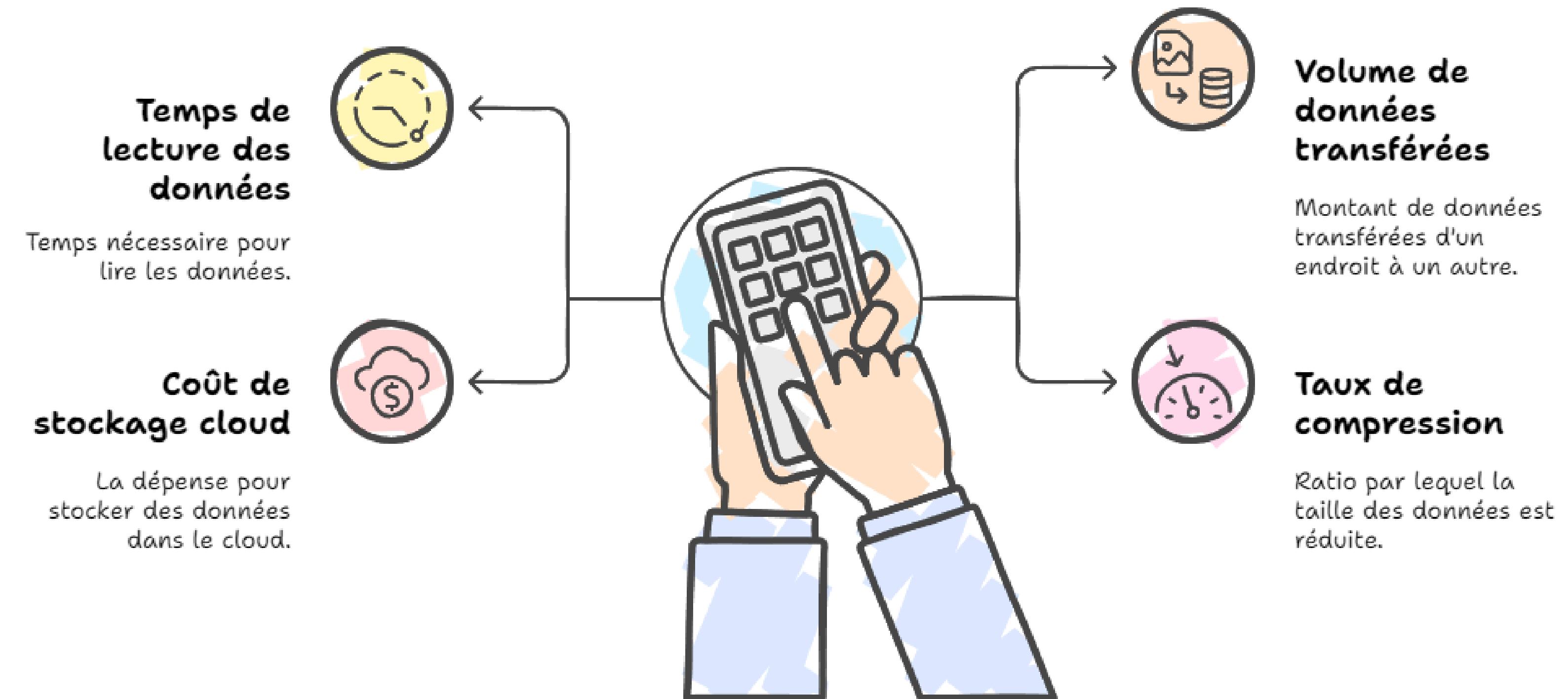


Conclusion

Stratégie de migration vers Parquet



Indicateurs clés à surveiller



Anti-modèles de Data Lake

Trop de petits fichiers

Avoir trop de petits fichiers ralentit les performances. Préférez regrouper les fichiers.



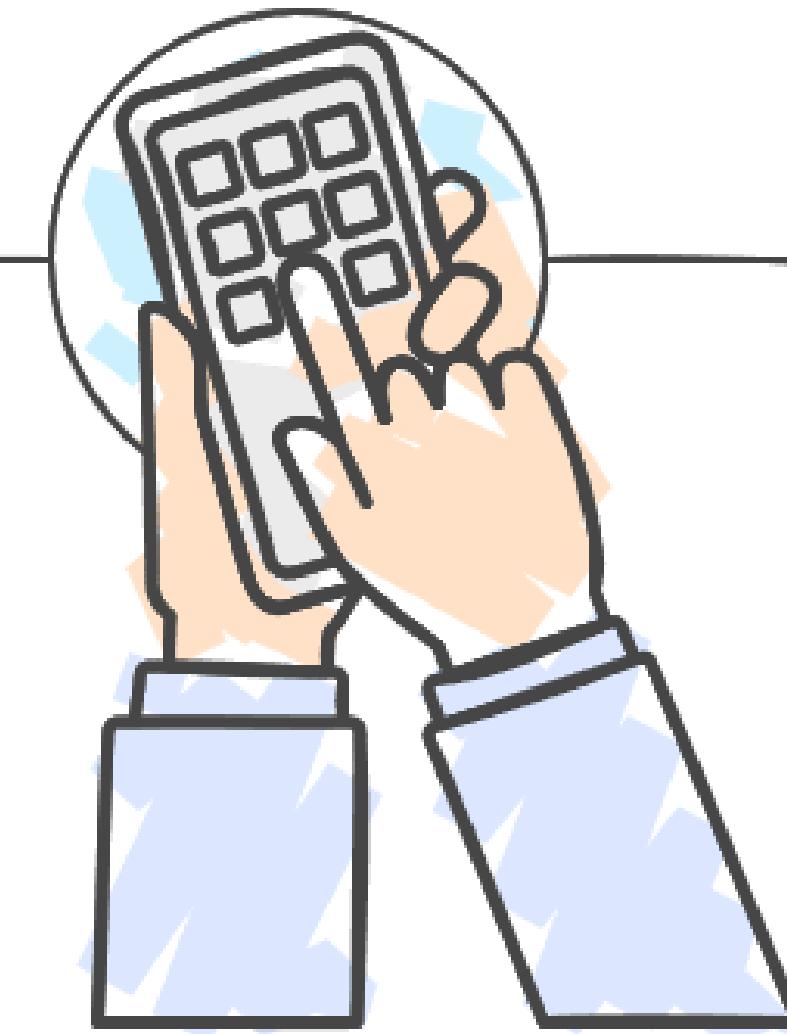
Fichiers trop gros

Des fichiers excessivement grands causent une lenteur lors des opérations de lecture.



Mauvais partitionnement

Un mauvais partitionnement empêche le partition pruning de se produire efficacement.



Merci De Votre Attention

