Architecture médaillon

Exemple concret et notions de base

Parquet

kafka

Massives, répétitives, produites par des systèmes

Données métiers irrégulières, à fiabiliser avant usage

Couche Bronze

Objectif : stocker la donnée brute telle qu'ingérée

build.bronze.potatoes_raw

build.bronze.strawberry_brut

Ajout d’une colonne de métadonnée date_stockage

build.bronze.crm_topic1_raw

Une table bronze par topic, sans traitement des events dans les topics

build.bronze.api_customers_raw

kafka

build.bronze.potatoes_raw

build.bronze.strawberry_raw

build.bronze.crm_topic1_raw

build.bronze.api_customers_raw

Ajout d'une colonne de métadonnée date_stockage

Une table bronze par topic, sans traitement des events dans les topics

Couche Silver

Objectif : nettoyer, standardiser et structurer

build.bronze.potatoes_raw

Trier les patates abîmées →
Supprimer les valeurs nulles / incohérentes

Eplucher, laver →
Standardiser les formats /renommer selon les conventions de nommage

build.silver.patatoes_clean

kafka

build.bronze.crm_topic1_raw

Extraire les events du topic →
extraire et normaliser les événements métier contenus dans le topic

Désérialiser →
rendre les messages encodés lisibles et exploitables

build.silver. topic1_event1_raw

build.silver. topic1_event2_raw

Couche Gold

Objectif : préparer des données métiers de valeur prêtes à consommer

build.silver.patatoes_clean

Dans la Gold, la donnée est prête à être servie à l'utilisateur final

Besoin 1 : *Suivre les ventes journalières de produits transformés*

Action : Agréger les quantités issues de la production

build.gold.french_fries_agg

Besoin 2 : Comparer le score de satisfaction moyen par ferme d'origine

Action : Enrichir les données avec jointures

build.gold.potato_farm_satisfaction

Récap

Objectif : Synthétiser les points clés de la section et fixer les messages à retenir

architecture médaillon

Bronze

Objectif : stocker la donnée brute telle qu'ingérée

- Données ingérées telles quelles (brutes, semi-structurées)
- Provenance diverse : Kafka, API, fichiers, ERP…
- Pas ou peu de traitement (possible ajout de métadonnées)
- Utilisée pour la traçabilité, l'archivage, ou reprocessing

Silver

Objectif : nettoyer, standardiser et structurer

- Traitement des valeurs nulles, doublons, données corrompues
- Typage, renommage, cast, nettoyage des formats (dates, strings)
- Conformité aux modèles métier
- Servira de base à plusieurs cas d'usage Gold

Gold

Objectif : préparer des objets métiers prêts à consommer

- Agrégation, jointures complexes, KPI calculés
- Tables analytiques optimisées pour la BI / reporting / ML
- Couches alignées avec les besoins finaux des métiers (ex : finance, logistique, marketing)
- Utilisées dans Power BI, Tableau, MLflow, export Excel…

Certification Databricks data engineer associate

0. Fondamentaux

Chaque table Databricks vit dans une arborescence : catalog.schema.table

- Il n’y a pas d’obligation stricte, mais des recommandations de nommage
  Exemple :
  - retail_build.gold.fact_sales (**snake_case** pour la lisibilité)
  - retail_prod.gold.fact_sales (Le nom indique **immédiatement** qu’il s’agit d’un environnement de production)
- Un utilisateur de la plateforme doit pouvoir comprendre, au premier coup d'œil, ce que contient une table rien qu’en lisant son **nom**.
- N.B En Community Edition, pas de catalogue visible (schema.table)

Kafka, c'est un tuyau à événements : on capte, on stocke, on consomme

- **Topic** : Une file logique nommée
- **Partition** : Découpe physique d'un topic → permet de **paralléliser** l'écriture/lecture
- **Offset** : Chaque message a une position ordonnée (par partition)
- **Event** : Un message produit par une app (ex : clic, login, commande)
- **Producteurs / Consommateurs** : Les apps envoient et lisent les événements

App (producer) → Kafka (topic) → Databricks (consumer)

Types de compute Databricks

🎯Objectif :

Comprendre les différents environnements d'exécution dans Databricks
(selon le type de tâche : batch, notebook, dashboard, ML), et comment bien choisir.

Type de compute	Cas d'usage typique	Persistant ?	Possible Serverless
SQL Warehouse	Dashboards, Analystes	Oui	Oui
Job Cluster	Pipelines batch	Non (ephem.)	Oui (via workflows)
All-purpose Cluster	Notebooks interactifs	Oui	Oui
ML Runtime	Training ML / Model Serving	Oui	Non

https://docs.databricks.com/aws/en/delta

Serverless Compute Databricks

🎯Objectif :
Comprendre comment fonctionne le mode Serverless dans Databricks,
ses avantages, ses limites, et les cas d'usage adaptés.

Databricks provisionne, auto-scale et **éteint** le compute à la demande, de manière totalement automatisée.

Pièges / Tips :

- ✅ Très efficace pour les dashboards partagés → performance + économie
- ✅ Aucune gestion de machine → focus sur le code
- ❌ Ne supporte pas les workloads ML Runtime / GPU
- ℹ️ Le temps de shutdown minimum est réglable via l'API (par défaut : 10 min, min 1 min en REST)

https://docs.databricks.com/aws/en/admin/sql/serverless

Photon – Moteur vectorisé ultra-performant

🎯Objectif :

Comprendre ce qu'est Photon, le moteur vectorisé en C++ développé par Databricks,
et dans quels cas il peut améliorer drastiquement les performances SQL.

- **Photon** est un moteur d'exécution écrit en C++
- **Optimisé** pour les architectures modernes (AVX, CPU vectorisés)
- **Accélère** les requêtes SQL, DML (MERGE, UPDATE, DELETE), et les jointures
- Compatible avec Delta Lake
- Utilisé automatiquement dans les SQL Warehouses et Jobs Serverless récents

Pièges / Tips :

✔ Offre un boost notable sur les requêtes en lecture et en écriture (Merge)

ℹ Peut réduire le temps de traitement de x2 à x10 sur les workloads BI lourds

https://www.databricks.com/fr/product/photon

Jobs & Workflows – Orchestration dans Databricks

🎯 Objectif :

Comprendre comment orchestrer des traitements data sur Databricks via les Jobs et les Workflows.
Permet de planifier, chaîner, monitorer et automatiser les tâches Spark, SQL, DLT ou notebooks.

Un Job = une tâche planifiée qui exécute un notebook, un script Python ou un SQL
Un Workflow = un ensemble de tâches dépendantes (ex. : Silver → Gold)
Possibilité de :
- ajouter des conditions (ex : `only if previous success`)
- définir des alertes (email, webhook)
- utiliser des paramètres (widgets, arguments)

Supporte les déclenchements :
- Manuels
- Plannings (CRON)
- API (webhook / GitHub Actions / Airflow)

Pièges / Tips :

✅ Toujours préférer Job Cluster pour les tâches planifiées

✅ Les tâches SQL peuvent être intégrées directement depuis un notebook ou une requête

❌ Attention aux dépendances circulaires dans les Workflows complexes

https://www.databricks.com/fr/product/photon

Delta Live Tables (DLT)

🎯 Objectif :

Créer des pipelines Bronze → Silver → Gold, avec scheduling, lineage et qualité intégrée.

Élément	Rôle
@dlt.table	Déclare une table persistée
@dlt.expect	Règle de validation (qualité)
Modes	Triggered (batch) ou Continuous (stream)

Exemple :

```
@dlt.table
@dlt.expect("montant_positif", "montant > 0")
def ventes_silver():
    return dlt.read("ventes_bronze")
```

Pièges / Tips :

- ✔ Pas besoin de write() → déclaratif
- ✔ Monitoring, alertes et rollback → automatiques
- ℹ Actif uniquement sur tables Unity Catalog
- ✘ Débogage complexe a utiliser sur des orchestrations simple

https://www.databricks.com/product/data-engineering/lakeflow-declarative-pipelines

Certification Databricks data engineer associate

1. Lakehouse Platform (24%)

Fondations – Lakehouse Platform

🎯 Objectif :

Comprendre comment Delta Lake assure la fiabilité des données avec des garanties ACID (Fiabilité des tables Delta), un journal de transactions, et la gestion du schéma.

Exemple :

```sql
CREATE TABLE ventes (
  id INT,
  produit STRING,
  montant DOUBLE
) USING DELTA;
-- Insertion incorrecte : types non conformes
INSERT INTO ventes VALUES ("abc", "téléphone", "non_num");
```

Propriété	Signification
A	Atomicité : chaque opération est tout ou rien
C	Cohérence : les règles de schéma sont respectées
I	Isolation : pas de conflits entre jobs concurrents
D	Durabilité : les données validées sont persistées

Pièges / Tips :

- ❌ Données insérées sans respecter le schéma → bloquées (enforcement).
- ❌ Fichiers ajoutés directement en Parquet → casse l'intégrité Delta.
- ✅ DESCRIBE DETAIL ma_table = rapide pour vérifier type Delta, version, chemin.
- ℹ️ Évolution du schéma possible avec MERGE ou Auto Loader (si activée).

https://docs.databricks.com/aws/en/delta

Optimisations Delta

🎯 Objectif :

Savoir optimiser les performances de lecture/écriture des tables Delta grâce à OPTIMIZE, Z-ORDER et la commande VACUUM (pour le stockage)

Exemple :

```
-- Compactage des petits fichiers (meilleure lecture)
OPTIMIZE ventes;
-- Optimisation pour les requêtes filtrées (clustering
logique)
OPTIMIZE ventes ZORDER BY (produit);
-- Nettoyage des versions obsolètes (au-delà de 7 jours
par défaut)
VACUUM ventes RETAIN 168 HOURS;
```

Pièges / Tips :

- ❌ VACUUM < 7 jours nécessitent spark.databricks.delta.retentionDurationCheck.enabled = false
- ❌ Pas de ZORDER sur toutes les colonnes — choisir celles souvent filtrées
- ℹ Suivre la taille et fragmentation avec: DESCRIBE DETAIL ma_table

https://docs.databricks.com/aws/en/delta

Unity Catalog & DBFS

🎯 Objectif :

Comprendre la gestion centralisée des données et métadonnées via Unity Catalog, et savoir naviguer le stockage DBFS.)

Terme	Définition
Catalog	Conteneur principal (ex : main, catalog_name)
Schema	Regroupe les tables dans un catalog (ex : silver, marketing
Table	Données structurées, format Delta ou autre (ex : client)

Exemple :

```
-- Accès à une table en notation 3 niveaux
SELECT * FROM main.silver.clients;
```

DBFS (Databricks File System) :

- Système de fichiers virtualisé sur le cloud (stockage objet sous-jacent)
- Accès possible via /dbfs/, dbfs:/, ou interface notebooks
- Idéal pour fichiers temporaires, logs, checkpoints...

Pièges / Tips :

- ❌ Tables UC ≠ tables legacy : les tables UC sont gérées dans un metastore centralisé
- ✅ Préférer UC pour sécurité, audit, gouvernance
- ❌ Le DBFS n'est pas un stockage de prod ,utiliser des tables Delta
- ℹ SHOW CATALOGS, SHOW SCHEMAS IN catalog, SHOW TABLES pour explorer

https://docs.databricks.com/aws/en/data-governance/unity-catalog

Certification Databricks data engineer associate

DELTA LAKE

2. ELT avec Spark SQL & Delta (29%)

Syntaxe SQL vs PySpark 1/2

🎯 Objectif :

Savoir écrire des requêtes équivalentes en SQL et en PySpark pour les Opérations classiques : SELECT / JOIN / GROUP BY

Action	SQL (catalog.schema.table)	PySpark (DataFrame API)	Tip ciblé
Lecture + affichage	SELECT * FROM prod.sales.clients;	df = spark.table("prod.sales.clients") display(df)	Rappel spark : display() (ou show()) déclenche le plan → sinon tout reste lazy
Sélection de colonnes	SELECT id, montant FROM …	df.select("id", "montant")	Sélectionner tôt pour réduire l'I/O
Filtre	WHERE montant > 100	df.filter(col("montant") > 100)	Préférer col()/expr() pour éviter les erreurs de typos
Jointure (inner)	… JOIN b ON a.id = b.id	a.join(b, "id")	Broadcast des petites tables pour gain en perf
Agrégation	GROUP BY region	df.groupBy("region").agg(sum("montant").alias("total"))	.agg() combine plusieurs mesures en une passe

ℹ df est réutiliser dans les lignes suivantes

https://docs.databricks.com/aws/en/pyspark/
https://docs.databricks.com/aws/en/sql/language-manual/

Syntaxe SQL vs PySpark 2/2

🎯 Objectif :

Savoir écrire des requêtes équivalentes en SQL et en PySpark pour les Opérations courante SQL : Window,LAG,Rank

Action	SQL (catalog.schema.table)	PySpark (DataFrame API)	Tip ciblé
ROW_NUMBER()	`ROW_NUMBER() OVER (PARTITION BY region ORDER BY montant DESC)`	`w = Window.partitionBy("region").orderBy(desc("montant"))` `df.withColumn("rang", row_number().over(w))`	Fenêtres ↔ Window
LAG() (valeur précédente)	`LAG(montant,1) OVER (ORDER BY date)`	`w = Window.orderBy("date")` `df.withColumn("prev", lag("montant", 1).over(w))`	❌ Lag() fonctionne mais n'est **pas recommandé à grande échelle** dans Spark (coûts : shuffle, mémoire).
Dense Rank	`DENSE_RANK() OVER (PARTITION BY region ORDER BY total DESC)`	`w = Window.partitionBy("region").orderBy(desc("total"))` `df.withColumn("rk", dense_rank().over(w))`	dense_rank sans “trous”

https://docs.databricks.com/aws/en/pyspark/
https://docs.databricks.com/aws/en/sql/language-manual/

Créer & requêter des Delta Tables

🎯 Objectif :

Créer des tables Delta (gérées ou externes) et les interroger via SQL ou PySpark en respectant les bonnes pratiques du catalog.

Exemple :

```
-- Table Delta gérée (managed)
CREATE TABLE prod.ventes.clients (
  id INT,
  nom STRING,
  pays STRING
) USING DELTA;
-- Table Delta externe
CREATE TABLE ext.clients
USING DELTA
LOCATION
'abfss://datalake@storage.dfs.core.windows.net/delta/
clients';
```

Pièges / Tips :

- ⚠ LOCATION obligatoire pour les tables externes, interdit pour les managed
- ✓ Préférer catalog.schema.table (notation à 3 niveaux) avec Unity Catalog
- ✓ df.write.format("delta").saveAsTable("catalog.schema.nom") Crée une table managed
- ⚠ Pour une table externe en pyspark .save(« fullpath") + en SQL CREATE TABLE … LOCATION ‘fullpath’

https://docs.databricks.com/aws/en/delta

MERGE INTO – Delta Lake

🎯 Objectif :

Utiliser MERGE INTO pour effectuer des opérations UPSERT (insert/update/delete) dans une table Delta.

Exemple :

```
MERGE INTO target_table AS target
USING source_table AS source
ON target.id = source.id
WHEN MATCHED THEN UPDATE SET target.val =
source.val
WHEN NOT MATCHED THEN INSERT (id, val) VALUES
(source.id, source.val)
```

Pour activer l'évolution de schéma (runtime 15.2+)
MERGE WITH SCHEMA EVOLUTION

Pièges / Tips :

- ✗ Plusieurs lignes source pour une même ligne target → erreur (sauf si désambigüisées).
- ✓ Plusieurs lignes target (exemple SCD2) → autorisé si modèle prévu. https://docs.databricks.com/aws/en/sql/language-manual/delta-merge-into#examples

https://docs.databricks.com/sql/language-manual/delta-merge-into.html

Time Travel – Delta Lake

🎯 Objectif :

Interroger les versions passées d'une table Delta (Time Travel) à l'aide de VERSION ou TIMESTAMP AS OF. (Audit & Récupération)

Exemple :

```
-- Explorer les versions de la table
DESCRIBE HISTORY table_name;
-- Voir la table dans une version
SELECT * FROM sales VERSION AS OF 42;
SELECT * FROM sales TIMESTAMP AS OF '2024-06-
01T00:00:00Z';
```

Pièges / Tips :

ℹ Après VACUUM, les anciennes versions deviennent inaccessibles

Par défaut :
delta.logRetentionDuration = 30 days
delta.deletedFileRetentionDuration = 7 days

https://docs.databricks.com/delta/history.html

Change Data Feed – Delta Lake

🎯 Objectif :

Suivre les changements ligne à ligne dans une table Delta entre deux versions.

Exemple :

```
CREATE TABLE my_table (
  id INT, name STRING
)
USING DELTA
TBLPROPERTIES ('delta.enableChangeDataFeed' =
'true');
-- Lecture des changements entre deux versions:
SELECT * FROM table_changes('my_table', 100, 110);
```

Pièges / Tips :

- ℹ Le Change Data Feed est désactivé par défaut.
- ℹ Il faut l'activer explicitement à la création ou via ALTER TABLE
- ✓ Ajuster la durée via : delta.deletedFileRetentionDuration
- ℹ Utilisez le mot-clé VALIDATE pour tester les fichiers sans déclencher l'ingestion

https://docs.databricks.com/delta/delta-change-data-feed.html

COPY INTO & Contraintes

🎯 Objectif :

Charger efficacement des fichiers vers une table Delta avec COPY INTO et appliquer des contraintes de qualité de données (NOT NULL, CHECK).

Exemple :

```
-- Ingestion de fichiers CSV dans une table Delta
COPY INTO prod.bronze.clients
FROM '/mnt/landing/clients/'
FILEFORMAT = CSV
FORMAT_OPTIONS ('header' = 'true’);

-- Table avec contraintes
CREATE TABLE prod.silver.clients (
  id INT NOT NULL,
  email STRING,
  age INT CHECK (age >= 0)
) USING DELTA;
```

Pièges / Tips :

⚠ COPY INTO ne détecte pas les schémas → définir le schéma cible dans la table

✓ auto Loader à privilégier pour ingestion continue (cf. slide auto loader)

✗ Erreurs levées si une contrainte est violée → bon pour la qualité des données , toutes les lignes sont rejetée (ACID)

✗ si on ALTER TABLE pour ajouter une contrainte toutes les lignes doivent répondre a la contrainte sinon erreur

https://docs.databricks.com/aws/en/sql/language-manual/delta-copy-into
https://docs.databricks.com/aws/en/tables/constraints

Auto Loader – Ingestion évolutive

🎯 Objectif :

Ingestion automatique et scalable de fichiers entrants vers une table Delta avec détection de schéma et gestion incrémentale.

Exemple :

```
df = (
  spark.readStream.format("cloudFiles")
  .option("cloudFiles.format", "csv")
  .option("cloudFiles.inferColumnTypes", "true")
  .option("cloudFiles.schemaLocation",
"/mnt/schemas/clients/")
  .load("/mnt/landing/clients/"))

df.writeStream \
  .format("delta") \
  .option("checkpointLocation",
"/mnt/checkpoints/clients/") \
  .outputMode("append") \
  .table("bronze.clients")
```

Pièges / Tips :

- ⚠ Le dossier de schéma (schemaLocation) est obligatoire
- ✓ Détection incrémentale des nouveaux fichiers (≠ COPY INTO)
- ℹ Compatible avec trigger = once ou availableNow pour ingestion contrôlée
- ✗ Ne pas utiliser overwrite en mode streaming

https://docs.databricks.com/aws/en/ingestion/cloud-object-storage/auto-loader

Certification Databricks data engineer associate

3. Ingestion & Streaming (22%)

ReadStream & WriteStream

🎯 Objectif :

Lire et écrire des flux de données avec Structured Streaming dans Databricks, en utilisant les bons formats et options essentielles.

Exemple :

```
df = (
  spark.readStream
  .format("delta") # ou "cloudFiles","kafka",etc.
  .load("/mnt/bronze/ventes_stream/")
)
df.writeStream \
  .format("delta") \
  .option("checkpointLocation",
"/mnt/checkpoints/ventes/") \
  .outputMode("append") \
  .table(« main.silver.ventes")
```

Pièges / Tips :

- ⚠ checkpointLocation est obligatoire → suivi de l'état du flux
- ℹ display(df) ne montre que le flux en mémoire – pas l'écriture réelle
- ✔ Compatible avec trigger = once ou availableNow pour ingestion contrôlée
- ✘ Ne pas utiliser overwrite en mode streaming

https://docs.databricks.com/aws/en/structured-streaming/concepts

Watermark – Gestion du retard (lateness)

🎯 Objectif :

Définir un seuil de retard toléré pour des événements en retard, afin d'éviter que l'état ne croisse à l'infini.

En streaming, certains événements arrivent en **retard** (latence réseau, buffers).Sans limite, Spark n'évacue jamais l'état d'une fenêtre, ce qui consomme de plus en plus de mémoire.

Exemple :

```
df = spark.readStream.format("delta").load("/mnt/stream/ventes/")
df_clean = df.withWatermark("event_time", "10 minutes")
agg = (
  df_clean
  .groupBy(window(col("event_time"), "5 minutes"))
  .count()
)
# Spark purgera les états des fenêtres dont la fin
# est passée depuis > 10 min par rapport au max
# event_time vu.
```

Pièges / Tips :

- ✔ Définir un délai ≈ 2 × la latence max observée.
- ⚠ Compatible uniquement avec les requêtes stateful (window, aggregation avec groupBy, etc.)
- ✘ Pas de purging = fuite mémoire garantie avec des windows ou stateful aggregations

https://docs.databricks.com/aws/en/structured-streaming/watermarks

Triggers – Contrôler le déclenchement

🎯 Objectif :

Définir quand Spark traite les données en streaming : en continu, à intervalle fixe, ou à la demande.

Trigger	Comportement	Cas d'usage typique
(par défaut)	Auto micro-batch	Streaming continu classique
.trigger(processingTime='5 min')	Batch toutes les 5 minutes	Agrégation périodique (latence maîtrisée)
.trigger(once=True)	Exécute une seule fois le flux disponible	Ingestion de fichiers ponctuelle
.trigger(availableNow=True)	Traite tout le backlog → s'arrête	Micro-batch piloté type batch, scalable

Pièges / Tips :

- ✅ availableNow = excellent compromis entre batch et streaming
- ⚠️ trigger(once) ne fonctionne qu'une fois → pas pour pipelines continus
- ✅ Toujours configurer checkpointLocation → reprise exacte en cas d'interruption
- ⚠️ Pas compatible avec tous les formats (ex : certains connecteurs Kafka)

https://docs.databricks.com/aws/en/structured-streaming/triggers

Stateful Streaming & CDC mix

🎯 Objectif :

Effectuer des agrégations avec état en streaming (stateful) et combiner cela avec le suivi des changements (Change Data Feed).

Pré-requis : activer CDF sur la table source

Exemple :

```
df_clean = (
  df.withWatermark("event_time", "10 minutes")
    .groupBy(window(col("event_time"), "5 minutes"),
col("region"))
    .agg({"montant": "sum"}))
-- À la création
CREATE TABLE main.silver.clients (
  id INT, nom STRING
) USING DELTA
TBLPROPERTIES ('delta.enableChangeDataFeed' = 'true’);
-- Lecture des changements entre deux versions
SELECT * FROM table_changes(‘main.silver.clients', 25, 30);
```

Pièges / Tips :

- ⚠ withWatermark obligatoire pour que l’état soit nettoyé
- ✗ Trop de clés uniques = état très lourd (OOM possible)
- ⚠ Pas de CDF sans activation explicite (TBLPROPERTIES)
- ✓ Combinaison idéale : CDF → Silver → agrégation stateful → Gold

https://docs.databricks.com/aws/en/delta/delta-change-data-feed
https://docs.databricks.com/aws/en/structured-streaming/concepts

Certification Databricks data engineer associate

4. Gestion et maintenance de Delta Lake (15%)

(Bien couvert dans les autres slides)

Delta Sharing

🎯 Objectif :

Partager des données Delta Lake entre comptes, régions ou plateformes sans copie ni export.

Concepts clés :

- **Producer** : publie une ou plusieurs tables Delta (en read-only).
- **Consumer** : accède à la table partagée via un endpoint Delta Sharing (REST / SQL / client open source).
- **Protocol** open-source compatible avec non-Databricks (Snowflake, Power BI…).

Share → Recipient → Provider Token

Exemple :

```
-- Créer un SHARE et ajouter des tables à exposer
CREATE SHARE ventes_share;
ALTER SHARE ventes_share ADD TABLE main.gold.ventes;
-- Déclarer le destinataire et Accorder l'accès en lecture
CREATE RECIPIENT external_consumer
USING SHARED ACCESS TOKEN '<token_généré_ou_rotation>';
GRANT SELECT ON SHARE ventes_share TO RECIPIENT external_consumer;
-- Côté Consumer (tenant externe)
SELECT * FROM delta.`https://sharing-endpoint/ventes`;
```

Pièges / Tips :

⚠ Pas de gestion d'identité interne → accès par lien sécurisé + contrôle externe

✗ Read-only uniquement (pas d'écriture possible pour le consumer).

✓ Supporte la mise à jour dynamique des données par le producteur.

⚠ Delta Sharing ne fonctionne que sur des tables Unity Catalog

https://docs.databricks.com/data-sharing

Delta Cache

🎯 Objectif :

Accélérer les lectures Delta Lake sans duplication mémoire (caching disque local côté cluster).

Activation :

```
spark.conf.set("spark.databricks.io.cache.enabled", "true")
```

	Spark df.cache()	**Delta Cache**
Portée	Dataset (DataFrame)	Table Delta (automatique)
Stockage	Mémoire (RAM)	Disque local (SSD cluster)
Volatilité	Perte à chaque exécution	Persistant sur cluster attaché
Type	Manuel	Automatique si activé

Pièges / Tips :

ℹ À utiliser sur les tables Silver/Gold consultées souvent, pas sur les tables brutes.

⚠ Non disponible sur drivers/workers sans disque local (ephemeral / spot sans SSD).

https://docs.databricks.com/performance/databricks-caching.html

Certification Databricks data engineer associate

5. Gouvernance avec Unity Catalog (10%)

(En partie couvert dans les autres slides)

Sécurité Unity Catalog : GRANT / REVOKE & hiérarchie des privilèges

🎯 Objectif :

Comprendre qui peut faire quoi dans Unity Catalog via la chaîne catalog → schema → table / view.

Exemple :

```
-- Révoquer quand un collaborateur quitte l'équipe
REVOKE ALL PRIVILEGES ON SCHEMA main.finance FROM USER
byebyebob@example.com;
```

Niveau	Exemples de privilèges	Syntaxe clé
CATALOG	USE CATALOG, CREATE SCHEMA, OWNERSHIP	GRANT USE CATALOG ON CATALOG main TO ROLE analyst;
SCHEMA	CREATE TABLE, USAGE, OWNERSHIP	GRANT CREATE TABLE ON SCHEMA main.finance TO ROLE etl_job;
TABLE / VIEW	SELECT, INSERT, UPDATE, DELETE, REFERENCES	GRANT SELECT, INSERT ON TABLE main.finance.fact_sales TO USER bob@example.com;

Pièges / Tips :

⚠️ Toujours accorder sur le niveau le plus bas nécessaire (principe du moindre privilège).

ℹ️ Regrouper les utilisateurs en roles (CREATE ROLE reporting;) puis GRANT au role.Pour automatiser : intégrer GRANT/REVOKE dans les steps de CI/CD (Databricks CLI ou Terraform).

https://docs.databricks.com/aws/en/data-governance/unity-catalog/manage-privileges/privileges

Row / Column Level Security & Tags

🎯 Objectif :

Appliquer la sécurité fine (données sensibles) et tracer le lineage dans Unity Catalog.

Exemple :

```
-- Autorise chaque analyste à lire seulement sa région
CREATE OR REPLACE ROW FILTER finance_region_filter
AS (region STRING) -> region = CURRENT_USER_REGION();
ALTER TABLE main.finance.fact_sales
SET ROW FILTER finance_region_filter ON (region);
-- Column masking sur l'email
CREATE OR REPLACE COLUMN MASK mask_email
AS (email STRING) -> sha2(email, 256); -- Hash irréversible
ALTER TABLE main.hr.employees
ALTER COLUMN email
SET MASK mask_email;
-- Classification Données Personnelles (PII)
ALTER TABLE main.hr.employees
SET TAGS ('pii' = 'true', 'owner' = 'dpo@entreprise.com');
```

Pièges / Tips :

- ℹ Les tags alimentent Unity Lineage : visibilité dans l'UI et via l'API.
- ✓ Les filtres/masques s'appliquent avant tout SELECT, même pour les vues.
- ℹ Testez toujours avec un compte non-admin pour valider la règle.

https://docs.databricks.com/aws/en/tables/row-and-column-filters