Test Data Engineering - Test 1

Sujet: Data Engineering Fundamentals
Niveau: Intermédiaire
Nombre de questions: 25

Questions et Réponses

Q1. Quelle est la différence entre ETL et ELT?

R1. | ETL | ELT | |—-|—-| | Extract → Transform → Load | Extract → Load → Transform | | Transformation dans le pipeline | Transformation dans le Data Warehouse | | Données propres à l'arrivée | Données brutes stockées d'abord | | Outils: Informatica, Talend, Beam | Outils: dbt, BigQuery SQL |

ELT est préféré avec les cloud data warehouses modernes (BigQuery, Snowflake) car ils ont la puissance de calcul pour transformer.

Q2. Qu'est-ce que le problème N+1 query et comment le résoudre en SQLAlchemy?

R2. Problème: 1 requête pour la liste principale + N requêtes pour les détails = N+1 total.

Solutions SQLAlchemy:

```
# joinedload - 1 requête avec JOIN
query.options(joinedload(Client.transactions))

# selectinload - 2 requêtes avec IN clause
query.options(selectinload(Client.transactions))
```

Détection: Activer `echo=True` sur l'engine et chercher les patterns répétitifs.

Q3. Pourquoi utiliser Parquet plutôt que CSV?

R3. | Parquet | CSV | |———|—-| | **Columnar** (lecture partielle) | Row-based (tout charger) | | **Compressé** (snappy, gzip) | Non compressé | | **Schéma intégré** | Pas de schéma | | **Types préservés** | Tout est string | | Idéal pour analytics | Idéal pour échange simple |

```
# Lecture partielle en Parquet
df = pd.read_parquet('data.parquet', columns=['id', 'montant'])
```

Q4. Qu'est-ce qu'une PCollection dans Apache Beam?

R4. Une **PCollection** est: - Une collection **distribuée** d'éléments - **Immuable** (chaque transformation crée une nouvelle PCollection) - L'abstraction de base pour les données dans Beam - Peut être **bornée** (batch) ou **non-bornée** (streaming)

```
# Création
lines = p | beam.io.ReadFromText('input.txt')  # PCollection<string>
numbers = lines | beam.Map(int)  # PCollection<int>
```

Q5. Quelle est la différence entre Map et FlatMap dans Beam?

R5.

```
# Map: 1 élément → 1 élément
| beam.Map(lambda x: x * 2)
# [1, 2, 3] → [2, 4, 6]

# FlatMap: 1 élément → 0, 1, ou N éléments
| beam.FlatMap(lambda x: x.split(','))
# ["a,b", "c"] → ["a", "b", "c"]
```

Q6. Comment optimiser une requête BigQuery?

R6. 1. **Partitionnement:** Diviser par date

```
PARTITION BY DATE(created_at)
```

2. **Clustering:** Organiser par colonnes fréquemment filtrées

```
CLUSTER BY client_id, agence
```

3. **Sélection de colonnes:** Éviter SELECT *
4. **Filtrer sur partition:** Utiliser la colonne de partition dans WHERE
5. **APPROX functions:** `APPROX_COUNT_DISTINCT` au lieu de `COUNT(DISTINCT)`

Q7. Qu'est-ce que le CDC (Change Data Capture)?

R7. CDC capture les modifications (INSERT, UPDATE, DELETE) d'une source pour les répliquer vers une cible.

Méthodes: - **Timestamp:** Colonne `updated_at`, polling périodique - **Trigger:** Triggers DB qui écrivent les changements - **Log-based:** Lecture du WAL/Binlog (Debezium)

Usage: Synchronisation temps réel entre systèmes.

Q8. Quelle est la différence entre Data Lake et Data Warehouse?

R8. | Data Lake | Data Warehouse | |———-|———-| | Données **brutes** (raw) | Données **structurées** | | Schema-on-read | Schema-on-write | | Stockage **économique** | Stockage **optimisé** pour queries | | Formats variés (JSON, Parquet, images) | Tables relationnelles | | Ex: GCS, S3, ADLS | Ex: BigQuery, Snowflake, Redshift |

Q9. Comment gérer les erreurs dans un pipeline Beam?

R9.

```
class ProcessWithErrors(beam.DoFn):
    def process(self, element):
        try:
            result = transform(element)
            yield result
        except Exception as e:
            yield beam.pvalue.TaggedOutput('errors', {
                'element': element,
                'error': str(e)
```

```
            })

# Utilisation
results, errors = (
    input_pcoll
    | beam.ParDo(ProcessWithErrors()).with_outputs('errors', main='valid')
)

# Écrire les erreurs
errors | beam.io.WriteToText('errors/')
```

Q10. Qu'est-ce que le windowing dans le traitement streaming?

R10. Le **windowing** divise un flux non-borné en fenêtres finies pour l'agrégation.

Types: - **Fixed Windows:** Intervalles fixes (ex: toutes les 5 minutes) - **Sliding Windows:** Fenêtres qui se chevauchent - **Session Windows:** Basé sur l'activité (gap d'inactivité)

```
| beam.WindowInto(window.FixedWindows(300))  # 5 minutes
| beam.CombinePerKey(sum)
```

Q11. Comment assurer la qualité des données dans un pipeline?

R11. **Dimensions à vérifier:** 1. **Complétude:** Pas de NULL critiques 2. **Unicité:** Pas de doublons 3. **Exactitude:** Valeurs dans les ranges attendus 4. **Fraîcheur:** Données récentes

```
def quality_check(df):
    issues = []
    if df['id'].duplicated().any():
        issues.append("Duplicates found")
    if df['montant'].isnull().mean() > 0.05:
        issues.append("Too many null montants")
    if (df['montant'] < 0).any():
        issues.append("Negative amounts")
    return issues
```

Q12. Quelle est la différence entre batch et streaming processing?

R12. | Batch | Streaming | |——-|———-| | Données **bornées** | Données **non-bornées** | | Traitement **périodique** | Traitement **continu** | | Latence: minutes/heures | Latence: secondes/ms | | Ex: Rapport journalier | Ex: Alertes fraude temps réel | | Outils: Spark Batch, Airflow | Outils: Kafka, Flink, Beam |

Q13. Comment fonctionne le partitionnement dans BigQuery?

R13. **Partitionnement:** Division physique de la table par une colonne (généralement date).

```
CREATE TABLE transactions
PARTITION BY DATE(created_at)
AS SELECT * FROM source;

-- Requête optimisée (scanne seulement la partition)
SELECT * FROM transactions
WHERE DATE(created_at) = '2024-01-15';
```

Avantages: - Réduit les données scannées - Réduit les coûts - Améliore la performance

---

Q14. Qu'est-ce que dbt et pourquoi l'utiliser?

R14. **dbt (data build tool)** est un outil de transformation ELT: - Écrit des transformations en **SQL** - **Versionné** avec Git - **Tests** intégrés - **Documentation** automatique - **Lineage** des données

```sql
-- models/staging/stg_transactions.sql
SELECT
    id,
    client_id,
    amount,
    created_at
FROM {{ source('raw', 'transactions') }}
WHERE amount > 0
```

Q15. Comment monitorer un pipeline de données?

R15. **Métriques clés:** - **Latency:** Temps entre source et destination - **Throughput:** Records par minute - **Error rate:** % d'erreurs - **Data freshness:** Âge des données

Alertes: - Pipeline failure - Latence > seuil - Volume anormal (trop ou trop peu) - Qualité sous le seuil

---

Q16. Qu'est-ce que l'idempotence et pourquoi est-ce important?

R16. **Idempotent:** Exécuter N fois donne le même résultat qu'une fois.

Importance: Si le pipeline échoue et redémarre, il ne doit pas dupliquer les données.

Stratégies:

```sql
-- MERGE/UPSERT
MERGE INTO target
USING source
ON target.id = source.id
WHEN MATCHED THEN UPDATE ...
WHEN NOT MATCHED THEN INSERT ...

-- DELETE + INSERT
DELETE FROM target WHERE date = @date;
INSERT INTO target SELECT * FROM source WHERE date = @date;
```

Q17. Comment choisir entre joinedload et selectinload en SQLAlchemy?

R17. | `joinedload` | `selectinload` | |—————-|—————-| | 1 requête avec JOIN | 2 requêtes avec IN | | Bon si **peu** de données liées | Bon si **beaucoup** de données liées | | Peut dupliquer les données parent | Pas de duplication | | Plus efficace réseau | Plus efficace mémoire |

Règle: Commencer avec `selectinload`, basculer vers `joinedload` si peu de relations.

Q18. Qu'est-ce que le schema evolution?

R18. Schema evolution: Capacité à modifier le schéma sans casser les pipelines existants.

Changements compatibles: - Ajouter une colonne (avec default) - Supprimer une colonne optionnelle - Élargir un type (INT → BIGINT)

Changements incompatibles: - Supprimer une colonne requise - Changer le type incompatiblement - Renommer une colonne

Solutions: Avro, Protobuf avec versioning.

Q19. Comment gérer les late data en streaming?

R19. Late data: Événements qui arrivent après la fermeture de leur fenêtre.

Solutions: 1. **Allowed lateness:** Garder la fenêtre ouverte plus longtemps 2. **Watermarks:** Estimer la progression du temps d'événement 3. **Triggers:** Émettre des résultats partiels, puis mettre à jour

```
| beam.WindowInto(
    window.FixedWindows(60),
    allowed_lateness=Duration(seconds=300)  # 5 min de tolérance
)
```

Q20. Qu'est-ce qu'un DAG dans Airflow?

R20. DAG (Directed Acyclic Graph): Graphe de tâches avec dépendances, sans cycles.

```
from airflow import DAG
from airflow.operators.python import PythonOperator

with DAG('my_dag', schedule='0 2 * * *') as dag:
    extract = PythonOperator(task_id='extract', python_callable=extract_fn)
    transform = PythonOperator(task_id='transform', python_callable=transform_fn)
    load = PythonOperator(task_id='load', python_callable=load_fn)

    extract >> transform >> load  # Dépendances
```

Q21. Comment détecter le problème N+1 dans les logs?

R21. Pattern à chercher:

```
SELECT * FROM clients
SELECT * FROM transactions WHERE client_id = 1
SELECT * FROM transactions WHERE client_id = 2
SELECT * FROM transactions WHERE client_id = 3
... (répété N fois)
```

Détection automatique:

```
import re
from collections import Counter

def detect_n_plus_1(queries):
    patterns = Counter()
    for q in queries:
        pattern = re.sub(r'\d+', 'N', q)
        patterns[pattern] += 1
```

```
    for pattern, count in patterns.most_common():
        if count > 5:
            print(f"  N+1 detected: {count} similar queries")
```

Q22. Quelle est la différence entre WRITE_TRUNCATE et WRITE_APPEND dans BigQuery?

R22. | WRITE_TRUNCATE | WRITE_APPEND | |—————-|—————-| | **Remplace** toutes les données | **Ajoute** aux données existantes | | Idempotent naturellement | Risque de doublons | | Pour refresh complet | Pour incremental load |

```
beam.io.WriteToBigQuery(
    'table',
    write_disposition=beam.io.BigQueryDisposition.WRITE_TRUNCATE
    # ou WRITE_APPEND
)
```

Q23. Comment implémenter un backfill?

R23. Backfill: Retraiter des données historiques.

```
# Airflow avec catchup
with DAG(
    'daily_etl',
    start_date=datetime(2024, 1, 1),
    catchup=True  # Exécute pour chaque jour manqué
):
    ...

# Manuel avec paramètre de date
def backfill(start_date, end_date):
    current = start_date
    while current <= end_date:
        process_date(current)
        current += timedelta(days=1)
```

Q24. Qu’est-ce que le data lineage?

R24. Data lineage: Traçabilité de l’origine et des transformations des données.

Questions auxquelles ça répond: - D’où viennent ces données? - Quelles transformations ont été appliquées? - Quels systèmes sont impactés si cette source change?

Outils: dbt (built-in), Apache Atlas, DataHub

Q25. Décrivez un pipeline ETL complet pour un rapport quotidien de transactions.

R25.

```
1. EXTRACT (02:00)
   - Source: MySQL (core banking)
   - Méthode: CDC ou full extract avec date filter
   - Output: fichiers Parquet sur GCS
```

```
2. TRANSFORM (02:30)
   - Nettoyage: valeurs NULL, formats
   - Enrichissement: JOIN avec dimensions (clients, agences)
   - Agrégations: totaux par client/agence/jour
   - Outil: Apache Beam ou dbt

3. LOAD (03:00)
   - Destination: BigQuery
   - Mode: WRITE_TRUNCATE pour table de staging
           MERGE pour table finale
   - Partitionnement: par date

4. QUALITY CHECK (03:15)
   - Row count vs source
   - Null percentage
   - Business rules validation

5. NOTIFY (03:30)
   - Slack/Email si succès ou échec
   - Rafraîchir le dashboard Looker Studio
```

Scoring

Score	Niveau
0-10	À améliorer
11-17	Intermédiaire
18-22	Avancé
23-25	Expert