TP 04
Bases des données orientées colonnes

Ce TP est noté. Vous devez soumettre un notebook par exercice (questions, requêtes et résultats) sur votre dépôt GitLab dans un dossier intitulé TP04.

Il vise à vous familiariser avec les concepts de base et avancés de Cassandra, y compris la création et la gestion des keyspaces, les stratégies de réplication, les niveaux de cohérence, et les opérations avancées.

- Toutes les commandes CQL doivent être exécutées dans l'interface CQLSH.
- Nommez le fichier de vos réponses tp_cassandra_ex1/ex2.

Exercice 1 : Création et Gestion des Keyspacesa

1) Créez un keyspace nommé restaurant_ks avec la stratégie de réplication SimpleStrategy et un facteur de réplication de 1.
2) Vérifiez que le keyspace a été créé avec succès.
3) Créez un deuxième keyspace nommé restaurant_ks2.
4) Affichez tous les keyspaces disponibles dans votre cluster Cassandra.
5) Modifiez la stratégie de réplication du keyspace restaurant_ks pour utiliser NetworkTopologyStrategy avec un datacenter nommé datacenter1 et un facteur de réplication de 1.
6) Vérifiez que la modification a été appliquée correctement..
7) Supprimez le keyspace estaurant_ks2.
8) Vérifiez que le keyspace a été supprimé avec succès (avec deux méthodes).
9) Créez un keyspace nommé restaurant_durable_ks avec la stratégie de réplication SimpleStrategy, un facteur de réplication de 1, et activez l'option de durabilité (durable_writes).
10) Vérifiez que le keyspace a été créé avec l'option de durabilité activée.
11) Expliquez ce que signifie l'option durable_writes dans la création d'un keyspace Cassandra. Dans quel scénario pourriez-vous désactiver cette option ?

Exercice 2 :

1. Créez un keyspace nommé resto_NY WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor': 1};
2. Sous `csqlsh`, vous pouvez maintenant sélectionner resto_NY pour vos prochaines requêtes.
3. Nous pouvons maintenant créer les tables (Column Family pour Cassandra) Restaurant et Inspection à partir du schéma SQL suivant :

```
CREATE TABLE Restaurant (
    id INT, Name VARCHAR, borough VARCHAR, BuildingNum VARCHAR, Street VARCHAR,
  ZipCode INT, Phone text, CuisineType VARCHAR,
  PRIMARY KEY ( id )
) ;
CREATE INDEX fk_Restaurant_cuisine ON Restaurant ( CuisineType ) ;

CREATE TABLE Inspection (
  idRestaurant INT, InspectionDate date, ViolationCode VARCHAR,
  ViolationDescription VARCHAR, CriticalFlag VARCHAR, Score INT, GRADE VARCHAR,
  PRIMARY KEY ( idRestaurant, InspectionDate )
) ;
CREATE INDEX fk_Inspection_Restaurant ON Inspection ( Grade ) ;
```

4. Vérifiez si les tables ont bien été créées.
5. Importez les fichiers CSV ‘restaurants.csv’ et ‘restaurants_inspections.csv’ pour remplir les Column Family.
6. Vérifiez le contenu des tables.
7. Exprimer en CQL les requêtes suivantes :
    A. Liste de tous les restaurants.
    B. - Liste des Noms de restaurants.
    C. - Nom et quartier (borough) du restaurant N° 41569764.
    D. - Dates et grades des inspections de ce restaurant.
    E. - Noms des restaurants de cuisine Française (French).

F. - Noms des restaurants situés dans BROOKLYN (attribut borough).
G. - Grades et scores donnés pour une inspection pour le restaurant n° 41569764 avec un score d'au moins 10.
H. - Grades (non nuls) des inspections dont le score est supérieur à 30.
I. - Nombre de lignes retournées par la requête précédente.
J. - Grades des inspections dont l'identifiant est compris entre 40 000 000 et 40 000 100.
K. Aide: Utiliser la fonction 'token()'.
L. - Compter le nombre de lignes retournées par la requête précédente

8. Utilisons les deux indexes sur Restaurant (borough et cuisineType). Trouvez tous les noms de restaurants français de Brooklyn.
9. Utilisez la commande TRACING ON avant la d'exécuter à nouveau la requête pour identifier quel index a été utilisé.
10. On veut les noms des restaurants ayant au moins un grade 'A' dans leurs inspections. Est-ce possible en CQL?.

Exercice 3 :

1. Considérez l'instruction CQL :

```
CREATE TABLE cars (
    make TEXT,
    model TEXT,
    year INT,
    color TEXT,
    cost INT,
    PRIMARY KEY ((make, model), year, color)
);
```

Laquelle des requêtes suivantes est valide ?

A.
```
SELECT * FROM cars
WHERE make='Ford';
```

B.
```
SELECT * FROM cars
 WHERE year = 1969
 AND color = 'Red';
```

C.
```
SELECT * FROM cars
WHERE make='Ford'
 AND model = 'Mustang'
 AND year = 1969;
```

D.
```
SELECT * FROM cars
WHERE make='Ford'
AND model = 'Mustang'
AND color = 'Red';
```

2. Considérez la définition de table avec une clé primaire omise :

```
CREATE TABLE reviews_by_restaurant (
    name TEXT,
    city TEXT,
    reviewer TEXT,
    rating INT,
    comments TEXT,
    review_date TIMEUUID,
```

```
    PRIMARY KEY (...)
);
```

On sait que :

- Les avis sur les restaurants sont identifiés de manière unique par une combinaison de name, city et reviewer
- Les avis sur les restaurants sont récupérés à partir du tableau à l'aide d'une combinaison de name,city
- La table a des partitions à plusieurs lignes

Quelle clé primaire possède cette table ?

3. Considérez la définition de la table et la requête CQL :

```
CREATE TABLE teams (
   name TEXT PRIMARY KEY,
   wins INT,
   losses INT,
   ties INT
);

SELECT * FROM teams_by_wins WHERE wins = 4;
```

Donner la vue matérialisée pour prendre en charge la requête ?

A.
```
CREATE MATERIALIZED VIEW IF NOT EXISTS
    teams_by_wins AS
   SELECT * FROM teams
     PRIMARY KEY((name), wins);
```

B.
```
CREATE MATERIALIZED VIEW IF NOT EXISTS
    teams_by_wins AS
   SELECT * FROM teams
     PRIMARY KEY((wins), name);
```

C.
```
CREATE MATERIALIZED VIEW IF NOT EXISTS
    teams_by_wins AS
    SELECT * FROM teams
     WHERE name IS NOT NULL AND wins IS NOT NULL
      PRIMARY KEY((name), wins);
```

D.
```
CREATE MATERIALIZED VIEW IF NOT EXISTS
   teams_by_wins AS
    SELECT * FROM teams
   WHERE wins IS NOT NULL AND name IS NOT NULL
   PRIMARY KEY((wins), name);
```

4. Considérez la définition de la table et la requête CQL :

```
CREATE TABLE restaurants_by_city (
   name TEXT,
   city TEXT,
   cuisine TEXT,
   price int,
   PRIMARY KEY ((city), name)
);

SELECT * FROM restaurants_by_city
  WHERE city = 'Sydney'
  AND cuisine = 'sushi';
```

Quel index secondaire peut être utilisé pour prendre en charge la requête ?

A.
```
CREATE INDEX cuisine_restaurants_by_city_2i
ON restaurants_by_city (cuisine);
```

B.
```
CREATE INDEX cuisine_restaurants_by_city_2i
ON restaurants_by_city (city, cuisine);
```

C.
```
CREATE INDEX cuisine_restaurants_by_city_2i
ON restaurants_by_city (cuisine, city);
```

D.
```
CREATE INDEX cuisine_restaurants_by_city_2i
ON restaurants_by_city (city, name, cuisine);
```

5. Quelle affirmation décrit la WHERE clause dans une requête ?

    A. Les clauses A. WHERE doivent référencer tous les champs de la clé de partition.
    B. Les clauses B. WHERE doivent référencer tous les champs de la clé de clustering.
    C. Les clauses C. WHERE doivent référencer tous les champs de la clé primaire.
    D. Les clauses D. WHERE doivent référencer tous les champs de la clé de partition et de la clé de clustering.

6. Considérez les déclarations CQL :

```
CREATE TYPE NAME (
   first TEXT,
   last TEXT
);
```

```
CREATE TABLE people (
   id UUID,
   name NAME,
   email TEXT,
   PRIMARY KEY(id, email)
);
```

Donner l'instruction INSERT pour insérer une ligne dans people ?

7. Considérez les déclarations CQL :

```
CREATE TABLE emails_by_user (
   username TEXT,
   email TEXT,
   description TEXT,
   nickname TEXT STATIC,
   PRIMARY KEY((username), email)
);

INSERT INTO emails_by_user (username, email, description, nickname)
 VALUES ('dc1234', 'david@datastax.com', 'work', 'Dave');

INSERT INTO emails_by_user (username, email, description, nickname)
 VALUES ('dc1234', 'david@gmail.com', 'personal', 'Dave');

UPDATE emails_by_user SET nickname = 'Davey', description = 'school'
 WHERE username = 'dc1234' AND email = 'david@gmail.com';

SELECT * FROM emails_by_user WHERE username = 'dc1234';
```

Quel est le résultat de l'exécution de ces instructions CQL ?

A.
```
 username  | email              | nickname | description
-----------+--------------------+----------+-------------
    dc1234 | david@datastax.com |     Dave |        work
    dc1234 |    david@gmail.com |    Davey |      school
```

B.
```
 username  | email              | nickname | description
-----------+--------------------+----------+-------------
    dc1234 | david@datastax.com |    Davey |        work
    dc1234 |    david@gmail.com |    Davey |      school
```

C.
```
 username  | email              | nickname | description
-----------+--------------------+----------+-------------
```

dc1234 | david@gmail.com | Davey | school

D.
```
 username  | email              | nickname | description
-----------+--------------------+----------+-------------
  dc1234 | david@datastax.com |     Dave |        work
```

Pour le reste de cette séance, vous devez créer une feuille de résumé sur les 4 bases de données NoSQL que nous avons déjà vues.

Bon travail