

NOTIONS BIGQUERY



Sommaire

01.Les CTE

02. Les CTE Syntaxe et exemples

03. Les CTE avec Case statement

04. Comprendre les cases statement dans bigquery

05. Visulaisation de données avec pandas et matplotlib

06. Les champs imbriqués dans bigquery Nested Fields

07.BigQuery what to know ?

08.Bonnes Pratiques

Les CTE

Common Table Expression

- Utilisés pour manipuler un ensemble de données complexe.
- Utilisés avec WITH statement et ne sont pas stockés quelque part.
- Agissent comme des subquerrys ou des tables temporaires.



Syntaxe

Requête CTE définit l'ensemble de résultats qui sera disponible pour la requête principale. Il peut s'agir de toute instruction SELECT valide.

Requête principale suit la définition de la CTE.
Elle utilise l'ensemble de résultats de la CTE en référant cte_nom.

SQL ▾

```
WITH cte_name AS (
    -- CTE query
    SELECT ...
)
-- Main query
SELECT ...
FROM cte_name
```



Exemple 1

Utilisons la bdd : bigquery-public-data.samples.natality pour créer une CTE, qui calcule le poids moyen pour chaque année de naissance

```
WITH yearly_birth_weights AS (
  SELECT
    year,
    AVG(weight_pounds) AS avg_weight
  FROM
    `bigquery-public-data.samples.natality`
  GROUP BY
    year
)
SELECT
  year,
  avg_weight
FROM
  yearly_birth_weights
ORDER BY
  year;
```

La requête CTE

La requête principale



Exemple 2

- Part 1

Requête CTE qui crée une table temporaire avec les **périodes de confinement + données de mobilité** (2 CTE)

Les 2 CTE sont séparés par des virgules

```
WITH confinement_periods AS (
    SELECT 'France' AS country, DATE('2020-03-17') AS start_date, DATE('2020-05-11')
    ) AS end_date, 'First Lockdown' AS period UNION ALL
    SELECT 'France', DATE('2020-10-30'), DATE('2020-12-15'), 'Second Lockdown' UNI
ON ALL
    SELECT 'France', DATE('2021-04-03'), DATE('2021-05-03'), 'Third Lockdown'
),
mobility_data AS (
    SELECT
        country_region,
        date,
        retail_and_recreation_percent_change_from_baseline AS retail_recreation,
        grocery_and_pharmacy_percent_change_from_baseline AS grocery_pharmacy,
        parks_percent_change_from_baseline AS parks,
        transit_stations_percent_change_from_baseline AS transit_stations,
        workplaces_percent_change_from_baseline AS workplaces,
        residential_percent_change_from_baseline AS residential
    FROM `bigquery-public-data.covid19_google_mobility.mobility_report`
    WHERE country_region = 'France'
)
```

ooo

</>

Exemple 2

• Part 2

Requête principale :

1. Sélectionne les données des 2 CTE.
2. Fait une jointure de la CTE cp avec CTE md basé sur la condition des dates.
3. Sélectionne les colonnes de période, date et les données de mobilité et les classe par période de confinement et date.

```
SELECT
    cp.period,
    md.date,
    md.retail_recreation,
    md.grocery_pharmacy,
    md.parks,
    md.transit_stations,
    md.workplaces,
    md.residential
FROM
    confinement_periods AS cp
JOIN
    mobility_data AS md
ON
    md.date BETWEEN cp.start_date AND cp.end_date
ORDER BY
    cp.period, md.date;
```

CTE avec Case statement

Case statement ?

- ⟨/⟩ Expression conditionnelle
- ⟨/⟩ Effectue différentes actions en fonctions de différentes conditions
- ⟨/⟩ Similaire au if-then-else

Exemple 1 :

Voici un exemple simple utilisant le jeu de données bigquery-public-data.samples.natality pour classer les poids de naissance.

classe les valeurs weight_pounds en 3 catégories

```
SELECT  
    year,  
    weight_pounds,  
    CASE  
        WHEN weight_pounds < 5.5 THEN 'low'  
        WHEN weight_pounds BETWEEN 5.5 AND 8.8 THEN 'normal'  
        ELSE 'high'  
    END AS weight_class  
FROM  
    `bigquery-public-data.samples.natality`  
WHERE  
    year IS NOT NULL  
    AND weight_pounds IS NOT NULL  
LIMIT 10;
```

Exemple 2 : CTE avec Case

On prend le même exemple que précédemment

Ce code SQL utilise une sous-requête commune (CTE) nommée **BirthWeightCategories** pour classer les poids de naissance en trois catégories : 'low', 'normal', et 'high'. Ensuite, il compte le nombre de naissances dans chaque catégorie par année et affiche ces résultats.

Une seule CTE dans ce code

```
WITH BirthWeightCategories AS (
  SELECT
    year,
    weight_pounds,
    CASE
      WHEN weight_pounds < 5.5 THEN 'low'
      WHEN weight_pounds BETWEEN 5.5 AND 8.8 THEN 'normal'
      ELSE 'high'
    END AS weight_class
  FROM
    `bigquery-public-data.samples.natality`
  WHERE
    year IS NOT NULL
    AND weight_pounds IS NOT NULL
)

SELECT
  year,
  weight_class,
  COUNT(*) AS count
FROM
  BirthWeightCategories
GROUP BY
  year,
  weight_class
ORDER BY
  year,
  weight_class;
```

Visualisation de données

- Avec les bibliothèques Python

Matplotlib

Pandas

Exemple 1 du brief

```
# Supposons que vous avez exporté les résultats de BigQuery vers un DataFrame Pandas nommé `mobility_data`
# mobility_data = pd.read_csv('path_to_your_exported_data.csv')

# Exemple de structure de DataFrame attendu
data = {
    'year': [2020, 2020, 2020, 2020, 2021, 2021, 2021, 2021],
    'month': [1, 2, 3, 4, 1, 2, 3, 4],
    'avg_retail_and_recreation': [-10, -20, -30, -40, -15, -25, -35, -45],
    'avg_grocery_and_pharmacy': [5, 10, 15, 20, 6, 12, 18, 24],
    'avg_parks': [0, -5, -10, -15, -2, -7, -12, -17],
    'avg_transit_stations': [-5, -10, -15, -20, -6, -12, -18, -24]
}

    'avg_workplaces': [-10, -15, -20, -25, -12, -17, -22, -27],
    'avg_residential': [5, 10, 15, 20, 6, 11, 16, 21]
}

mobility_data = pd.DataFrame(data)
```

un dictionnaire nommé data est créé. Chaque clé du dictionnaire représente une colonne de données (par exemple, year, month, avg_retail_and_recreation, etc.), et chaque valeur associée à une clé est une liste contenant les données pour cette colonne.

Le dictionnaire data est ensuite passé en argument à la fonction **pd.DataFrame()**. Cette fonction convertit le dictionnaire en un DataFrame, qui est une structure de données tabulaire avec des lignes et des colonnes, similaire à une feuille de calcul.

Focus sur DataFrame Pandas

Un DataFrame pandas est particulièrement utile pour l'analyse de données car il offre de nombreuses fonctionnalités pour la manipulation, la filtration, l'agrégation et la visualisation des données.

Exemple 1 du brief

```
# Convertir les colonnes year et month en une colonne de date
mobility_data['date'] = pd.to_datetime(mobility_data[['year', 'month']].assign(day=1))

# Définir les types de lieux pour l'analyse
places = ['avg_retail_and_recreation', 'avg_grocery_and_pharmacy',
           'avg_parks', 'avg_transit_stations', 'avg_workplaces', 'avg_residential']

# Créer des graphiques pour chaque type de lieu
for place in places:
    plt.figure(figsize=(10, 5))
    plt.plot(mobility_data['date'], mobility_data[place], marker='o', linestyle='-')
    plt.title(f'Seasonal Trends in {place.replace("_", " ")}.title()')
    plt.xlabel('Date')
    plt.ylabel('Percentage Change from Baseline')
    plt.grid(True)
    plt.show()
```

- **mobility_data[['year', 'month']]** sélectionne les colonnes **year** et **month** du DataFrame.
- **.assign(day=1)** ajoute une colonne supplémentaire **day** avec une valeur de 1 pour chaque entrée, ce qui permet de créer une date complète avec année, mois, et jour.
- **pd.to_datetime(...)** convertit ces colonnes en un objet de type **datetime**, créant ainsi une nouvelle colonne **date** dans le DataFrame **mobility_data**. Cela facilite la manipulation et l'analyse des données temporelles.

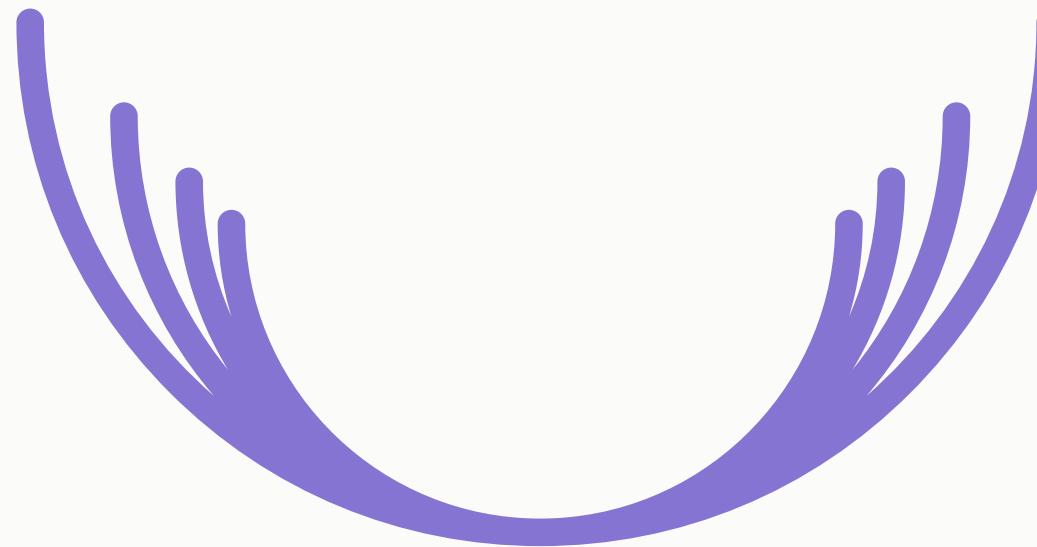
La liste **places** contient les noms des colonnes correspondant à différents types de lieux pour lesquels les données de mobilité seront analysées et visualisées.

Exemple 1 du brief

```
# Créer des graphiques pour chaque type de lieu
for place in places:
    plt.figure(figsize=(10, 5))
    plt.plot(mobility_data['date'], mobility_data[place], marker='o',
             linestyle='--')
    plt.title(f'Seasonal Trends in {place.replace("_", " ").title()}')
    plt.xlabel('Date')
    plt.ylabel('Percentage Change from Baseline')
    plt.grid(True)
    plt.show()
```

Écrivez quelques chose, appuyez sur la touche entrée pour l'IA ou sur la touche /

- **for place in places:** itère à travers chaque type de lieu dans la liste **places**.
- **plt.figure(figsize=(10, 5))** crée une nouvelle figure pour chaque graphique avec une taille de 10x5 pouces.
- **plt.plot(mobility_data['date'], mobility_data[place], marker='o', linestyle='--')** trace un graphique des données de mobilité pour le lieu courant (**place**) en fonction de la colonne **date**. Les marqueurs **o** (cercles) sont utilisés pour les points de données, et les lignes sont dessinées entre les points.



Nested Fields

Dans Google BigQuery, les champs imbriqués et répétés (nested and repeated fields) permettent de stocker des données structurées de manière efficace tout en maintenant la relation entre les données associées.

Comment ça marche ?

- **Structuration**

Les champs imbriqués sont définis dans le schéma de la table. Un champ imbriqué peut contenir plusieurs sous-champs, chacun avec son propre type de données. Par exemple, une table peut avoir un champ address qui contient des sous-champs comme street, city, state, et zip_code.

```
CREATE TABLE my_dataset.my_table (
    id INT64,
    name STRING,
    address STRUCT<
        street STRING,
        city STRING,
        state STRING,
        zip_code STRING
    >
);
```

Comment ça marche ?

- ***Insertion de Données***

Les données peuvent être insérées en fournissant des valeurs pour chaque sous-champ du champ imbriqué. On utilise des structures JSON pour représenter ces données.

```
INSERT INTO my_dataset.my_table (id, name, address)
VALUES
    (1, 'John Doe', STRUCT('123 Main St', 'Springfield', 'IL', '6270
1')),
    (2, 'Jane Smith', STRUCT('456 Maple Ave', 'Lincoln', 'NE', '6850
8'));
```

Comment ça marche ?

- **Requêtes et Accès aux Champs Imbriqués**

Pour interroger des champs imbriqués, on utilise la notation pointée pour accéder aux sous-champs. Par exemple, pour sélectionner le city de chaque address, on écrit :

```
SELECT id, name, address.city  
FROM my_dataset.my_table;
```

Comment ça marche ?

- ***On peut utiliser les fonctions comme UNNEST pour travailler avec des champs imbriqués contenant des listes d'éléments.***

la fonction UNNEST en BigQuery permet de convertir un champ de type tableau (ARRAY) en une série de lignes, facilitant ainsi les requêtes sur des données imbriquées contenant des listes.

Voici une explication détaillée de son fonctionnement

```
CREATE TABLE my_dataset.orders (
    order_id INT64,
    customer_name STRING,
    items ARRAY<STRUCT<item_id INT64, item_name STRING, quantity INT
64>>
);
```

```
SELECT order_id, customer_name, item.item_name, item.quantity
FROM my_dataset.orders, UNNEST(items) as item;
```

Focus sur UNNEST

- ***Définition***

La fonction UNNEST est utilisée pour décomposer un tableau en lignes individuelles. Chaque élément du tableau devient une ligne distincte dans le résultat de la requête. Cela est particulièrement utile pour travailler avec des champs de type ARRAY contenant des structures complexes.

EXEMPLE PRATIQUE UNNEST

Considérons une table **orders** avec la définition suivante :

Cette table contient des commandes, chaque commande ayant un champ items qui est un tableau de structures représentant les articles commandés.

```
CREATE TABLE my_dataset.orders (
    order_id INT64,
    customer_name STRING,
    items ARRAY<STRUCT<item_id INT64, item_name STRING, quantity INT
64>>
);|
```

EXEMPLE PRATIQUE UNNEST

Utilisation de UNNEST pour Interroger les Données

Pour interroger cette table et obtenir une ligne par article commandé, on utilise la fonction UNNEST comme suit :

```
SELECT order_id, customer_name, item.item_name, item.quantity  
FROM my_dataset.orders, UNNEST(items) as item;
```

UNNEST(items) as item : Décompose le tableau items en lignes individuelles et associe chaque élément du tableau à l'alias item. Cela permet d'accéder aux sous-champs de chaque élément du tableau en utilisant l'alias item.

EXEMPLE PRATIQUE UNNEST

Le résultat de UNNEST :

Le résultat de cette requête sera une table où chaque ligne représente un article commandé, associée à son identifiant de commande et au nom du client. Par exemple :

order_id	customer_name	item_name	quantity
1	John Doe	Widget A	2
1	John Doe	Widget B	1
2	Jane Smith	Gadget C	3

BigQuery what to know ?

Concepts et Architecture



Comment BigQuery stocke et traite les données



Comment BigQuery stocke et traite les données



Différentes types de tables (tables partitionnées, tables fractionnées).

BigQuery what to know ?

SQL et Requêtes

SQL Standard

Maîtriser le dialecte SQL de BigQuery, y compris les commandes de base (SELECT, INSERT, UPDATE, DELETE), les fonctions analytiques, les jointures et les sous-requêtes.

Fonctions spécifiques à BigQuery

Utiliser les fonctions spécifiques comme ARRAY, STRUCT, et les fonctions analytiques avancées (OVER(), WINDOW FUNCTIONS).

Fonctions spécifiques à BigQuery

Savoir comment optimiser les requêtes pour améliorer les performances et réduire les coûts (ex : en utilisant les tables partitionnées et fractionnées).

BigQuery what to know ?

Gestion des Données



Chargement et exportation, via GCS, des fichiers locaux, flux de données



Transformation des données avec ETL en utilisant des outils comme DataFlow ou avec des transformations SQL



Gestion des schémas (ajout/suppression de colonnes)

BigQuery what to know ?

Intégration et Outils complémentaires

Google Cloud Storage GCS

Comprendre comment BigQuery interagit avec GCS pour le chargement et l'exportation des données.

DataFlow et Apache Beam

Utiliser Dataflow pour des pipelines de données ETL complexes.

Looker, Data Studio

Intégrer BigQuery avec des outils de visualisation et de BI pour créer des tableaux de bord et des rapports.

API BigQuery

Utiliser l'API BigQuery pour automatiser les tâches et intégrer BigQuery dans des applications.

BONNES PRATIQUES

</>

Utiliser le partitionnement et le clustering pour optimiser les performances des requêtes et réduire les coûts.

</>

Utiliser les outils de monitoring comme Stackdriver pour surveiller les performances des requêtes et ajuster les configurations pour une meilleure efficacité.

</>

Automatiser les tâches récurrentes en utilisant des outils comme Cloud Composer (basé sur Apache Airflow).