OPTIMIZAREA OPERAȚIUNILOR SQL CU AJUTORUL LIMBAJULUI PYTHON

Daria NEDELCO

Departamentul Ingineria Software și Automatică, grupa TI-191 F/R, Facultatea Calculatoare, Informatică și Microelectronică, Universitatea Tehnică a Moldovei, Chișinău, Republica Moldova

Autorul corespondent: Daria NEDELCO, e-mail: nedelco.daria@isa.utm.md

Conducător științific: Dorian SARANCIUC, DISA, FCIM, UTM

Rezumat. *În lucrare sunt prezentate metode de optimizare a operațiunilor SQL cu ajutorul limbajului Python. Combinarea posibilităților SQL cu limbajul Python adaugă flexibilitate și utilitate care crește performanța și reduce consumul de resurse necesare pentru execuția interogărilor.*

Cuvinte cheie*: Operațiuni SQL, limbajul Python, optimizare, flexibilitate.*

Introducere

SQL este un limbaj structurat de date care permite manipularea datelor in sistemele de manipulare a a bazelor de date relaționale. Cu toate că, SQL este un instrument puternic de manipulare, extragere și scriere a datelor intr-un sistem de bază de date, acestuia îi lipsește flexibilitatea și utilitatea limbajului de scriptare Python [1].

Ciclare prin multiple Tabele SQL

Cea mai evidentă metodă prin care limbajul Python ar putea optimiza codul SQL este incorporarea unui șir de interogare intr-o structură de ciclare Python pentru a itera mai multe interogări consecutiv.

Folosind variabilele definite in Python, putem crea o interogare de bază folosind atît text SQL cît și variabile Python.

De exemplu, pentru a extrage o listă a tuturor tabelelor și seturi de date dintr-un proiect, va fi necesar de scris următorul cod SQL:

```
SELECT * FROM
`my_project.dataset_1.INFORMATION_S
CHEMA`
UNION ALL
SELECT * FROM
`my_project.dataset_2.INFORMATION_S
CHEMA`
UNION ALL
SELECT * FROM
`my_project.dataset_3.INFORMATION_S
CHEMA`
```

Figura 1. Codul SQL care permite extragerea listei tabelurilor și seturilor de date din proiect

Pentru a evita introducerea manuală a interogărilor putem integra un script Python.

```
from google.cloud import bigquery

datasets = ['dataset_1',
'dataset_2', 'dataset_3']

bq_client = bigquery.Client()

for dataset in datasets:
    get_datasets =
bq_client.query("SELECT dataset_id,
table_id,         size_bytes,
ROUND(size_bytes / 10000000000), 2)
AS gb_size
FROM
`"+dataset.dataset_id+"`.__TABLES__
GROUP BY 1, 2, 3")

    tables = get_datasets.result()
    for table in tables:
        dataset_id = table.dataset
        table_id = table.table_id
        size = table.size_bytes
        gb_size = table.gb_size

        print(dataset_id, table_id,
size, gb_size)
```

Figura 2. Scriptul Python de formare a interogărilor

Chiar daca pare a fi mai complex, cu două cicluri, acest script rulează prin toată lista de șiruri de date, înlocuind UNION. Cicluri in SQL pot fi create, dar de obicei, sunt necesare pasuri adăugatoare ca definirea variabilelor si crearea UDF (User Defined Functions) [2].

Definirea Schemei Automate

The BigQuery Python Client permite definirea schemelor ca o listă, care poate fi transmisă către funcția de încărcare. Definirea unei scheme BigQuery in SQL manual poate duce la schimbarea tipului de date cînd este încărcată pe GCP (Google Cloud Platform):

```
schema = [

bigquery.SchemaField("first_name",
"STRING"),

bigquery.SchemaField("last_name",
"STRING"),
     bigquery.SchemaField("age",
"INTEGER")

]
```

Figura 3. Schema BigQuery

Introducerea manuală a datelor devine anevoiasă în caz că este nevoie de minim 100 de coloane. Soluția ar fi funcția Python care permite auto-completarea cîmpurilor.

```
def create_schema(field_list: list,
type_list: list):

    schema_list = []

    for fields, types in
zip(field_list, type_list):
        schema =
bigquery.SchemaField(fields, types)
        schema_list.append(schema)

    return schema_list
```

Figura 4. Funcția Python cu auto-completarea cîmpurilor

Convertirea în Data Frame intr-o linie de script Python

Există o metodă simplă de creare a Data Frame derivată din interogări SQL și este necesar doar de o linie de script Python, în special dacă stocăm interogările extern într-un fișier config.

```
query = """ SELECT * FROM
`my_project.dataset.table` """

query_job =
bq_client.query(cfg.query).to_dataf
rame()
```

Figura 5. Linia de script Python

În caz că dorim să salvăm rezultatul interogărilor, putem converti în fișier CSV în aceeași linie.

```
query_job =
bq_client.query(cfg.query).to_dataf
rame().to_csv('query_output.csv')
```

Figura 6. Convertirea în fișier CSV

Operațiuni Append/Truncate

Să presupunem că avem un tabel, care necesită editarea multiplă a datelor în timpul zilei și la sfîrșit de zi avem nevoie de toate datele introduse, în urma căruia pot apărea cîmpuri duplicate. În mod ideal, ar fi bine de eliminat toate duplicatele cînd încărcăm datele. Cu părere de rău, BigQuery nu suportă operațiile APPEND/TRUNCATE și în caz că dorim să rescriem un tabel SQL pentru o perioadă anumită de timp putem combina Python cu SQL. Odată cu pornirea script-ului, acesta arată toate schimbările introduse în timpul real [3].

```
crud_statement = """ DELETE FROM
table WHERE date = CURRENT_DATE()
"""

bq_client.query(crud_statement)

df = df[(df['date'] ==
date.today())]

bq_client.load_table_from_dataframe
(df, job_config)
```

Figura 7 - Scriptul Prezintă schimbările în timp real

Concluzii

Combinarea a unui limbaj script ca Python cu SQL oferă posibilitatea de a efectua operațiuni imposibile doar pentru limbajul SQL, cum ar fi:

1. Ciclare prin multiple tabele SQL;
2. Definirea schemei automate;
3. Convertirea în Data Frame intr-o linie de script Python;
4. Operațiuni Append/Truncate

Referințe

1. SQL BigQuery [online]. [accesat 10.12.2022]. Disponibil: https://cloud.google.com/bigquery/docs/reference/standard-sql/introduction
2. Combinarea SQL cu limbajul Python [online]. [accesat 10.12.2022]. Disponibil: https://towardsdatascience.com/improve-your-sql-skills-using-pure-python-9e50736b3d9c
3. SQL Queries in Python [online]. [accesat 10.12.2022]. Disponibil: https://towardsdatascience.com/a-simple-approach-to-templated-sql-queries-in-python-adc4f0dc511