Revisión de una Metodología para la Extracción, Limpieza y Organización de Datos SQL con Programación Declarativa en Python para el Desarrollo de Análisis de Datos Descriptivos.

April 22, 2024

1 Procedimientos ETL

Recordemos que los procedimientos ETL constan de Extract, Transform y Load, los cuales nos dan la pauta a seguir para seguir el ciclo de la Ingeniería de Datos. De esta forma seguimos una metodología establecida y evitamos estar trabajando sin una dirección específica.

En este ejemplo, se utilizará la base de datos de Sakila, la cual es una base de datos ejemplo proporcionada por MySQL que es utilizada comúnmente para propósitos educativos, demostraciones y pruebas. Sakila simula una base de datos de una tienda de alquiler de películas, similar a la cadena de alquiler de vídeos Blockbuster que solía existir.

Sakila contiene tablas que representan películas, actores, clientes, tiendas, alquileres, etc.

Para hacer uso de ella, hemos descargado el archivo script SQL a través de la página oficial de MySQL: `https://dev.mysql.com/doc/index-other.html`.

Ahora para instalar la base de datos podremos utilizar los siguientes comandos de MySQL.

```
$ mysql -u root -p sakila < sakila-schema.sql
$ mysql -u root -p sakila < sakila-data.sql
```

Después de haber ejecutado estos comandos, deberíamos ser capaces de ver la base de datos

instalada en nuestro sistema, para comprobarlo podemos ejecutar:

```
mysql> USE sakila;
Database changed
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_sakila           |
+----------------------------+
| actor                      |
| actor_info                 |
| address                    |
| category                   |
| city                       |
| country                    |
| customer                   |
| customer_list              |
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| staff                      |
| staff_list                 |
| store                      |
+----------------------------+
```

Ahora, con ello podremos comenzar a trabajar nuestro procedimiento ETL.

Primeramente, es necesario determinar que incógnita queremos responder sobre los datos disponibles. Esto variará dependiendo de la base de datos y los tipos de datos que ésta almacene. Para este caso, intentaremos responder las preguntas:

- **Patrones de alquier:** ¿Cuáles son las películas más alquiladas?

Una vez planteadas las incógnitas, podemos determinas que tablas serán relevantes para nuestro análisis, las cuales son:

- **Patrones de alquier:**
    1. rental
    2. inventory
    3. film

1.1 Extract.

Para nuestro ejemplo, utilizaremos Python con SQL Alchemy.

```
from sqlalchemy import create_engine, MetaData, Table, select, func

# Configurar la conexión a la base de datos
engine = create_engine('mysql://starboy:starboyc00l@localhost/sakila')
metadata = MetaData()

# Asociar el motor a la Metadata después de su creación
metadata.bind = engine

# Reflejar las tablas
metadata.reflect(bind=engine)

# Obtener las tablas reflejadas
rental_table = metadata.tables['rental']
inventory_table = metadata.tables['inventory']
film_table = metadata.tables['film']
```

En este código hemos creado la conexión respectiva a nuestra base de datos de manera local, donde generamos un "motor" de base de datos que nos permitirá enlazarnos al motor de base de datos de MySQL, podemos pensar en él cómo una "reflejo" de la base de datos original.

1.2 Transform.

Ahora bien, después de haber extraído los datos, es necesario organizarlos de manera que sea más sencillo utilizarlos para el análisis de datos.

```
# Construir la consulta para obtener las películas más alquiladas
# Obtener las 10 películas más alquiladas
query = (
```

```
    select(film_table.c.title, film_table.c.film_id,
        func.count(rental_table.c.inventory_id).label('rental_count'))
    .select_from(
        rental_table.join(inventory_table, rental_table.c.inventory_id ==
            inventory_table.c.inventory_id)
        .join(film_table, inventory_table.c.film_id == film_table.c.film_id)
    )
    .group_by(film_table.c.film_id)
    .order_by(func.count(rental_table.c.inventory_id).desc())
    .limit(10)
)
```

De esta forma, en query se almacenarán los datos extraídos de la consulta en la base de datos, de manera que con SQL Alchemy podemos escribir consultas de SQL a través del lenguaje de programación Python, agregando una capa de programación declarativa.

Para ver los resultados de la consulta anterior podemos realizar lo siguiente:

```
# Ejecutar la consulta y obtener los resultados
with engine.connect() as connection:
  result = connection.execute(query)
  for row in result:
      print(row)
```

Como respuesta tendríamos que ver algo como lo siguiente:

```
$ python3 main.py

metadata.reflect(bind=engine)
('BUCKET BROTHERHOOD', 103, 34)
('ROCKETEER MOTHER', 738, 33)
('RIDGEMONT SUBMARINE', 730, 32)
('GRIT CLOCKWORK', 382, 32)
('SCALAWAG DUCK', 767, 32)
('JUGGLER HARDLY', 489, 32)
('FORWARD TEMPLE', 331, 32)
('HOBBIT ALIEN', 418, 31)
('ROBBERS JOON', 735, 31)
('ZORRO ARK', 1000, 31)
```

1.3 Load.

Una vez que se haya realizado la Organización de los datos, es una buena idea es crear un acceso a esta información de manera que no sea necesario acceder a la información y organizarla nuevamente, para esto son muy útiles las vistas (views) de SQL.

```
# Crear una vista en la base de datos a partir de la consulta
view_name = "top_rented_films"
create_view_query = text(f"CREATE OR REPLACE VIEW {view_name} AS
    {str(query.compile(engine, compile_kwargs={'literal_binds': True}))}")

with engine.begin() as conn:
    conn.execute(create_view_query)
    print(f"Vista '{view_name}' creada exitosamente.")

# Consultar la información de la vista
view = Table(view_name, metadata, autoload_with=engine)

# Consultar la información de la vista
with engine.connect() as connection:
    result = connection.execute(select(view))
    for row in result:
        print(row)
```