Ingeniería en Sistemas Computacionales

Bases de Datos No Relacionales

Cassandra Lab 3 – Cassandra y Python

Marco Ricardo Cordero Hernández

Tlaquepaque, Jal., 26 de febrero de 2023

Como se ha podido comprobar en prácticas anteriores, Cassandra ofrece bondades a través de la alta disponibilidad de datos y una rápida indexación de tablas; esto ha sido visualizado por medio de cqlsh o “el Shell de CQL”, es decir, únicamente se ha manejado a través de una interfaz de línea de comandos, pero ¿Cuál es el punto de una base de datos si los datos que contiene no sirven para nada? ¿En qué momento es que los datos se transforman en información? Cualquier estudiante primerizo de alguna carrera afín a las ciencias computacionales y de las tecnologías de la información encontrará en sus primeros periodos del aprendizaje con el hecho de que las aplicaciones funcionan a través de varias tecnologías entrelazadas entre sí para brindar una agradable experiencia al usuario, pero sobre todo, una herramienta utilitaria completamente pragmática.

En esta última práctica relacionada con Cassandra, se pretende realizar una sencilla aplicación que sirve como el punto de entrada a lo que podría llegar a convertirse una aplicación administrativa, un visualizador de datos, o inclusive un sistema entero de finanzas. La flexibilidad de Python como “pseudolenguaje” de producción sale a relucir una vez más, ya que, en esta ocasión, lo que se trata de demostrar será realizado a través de una colección de módulos construidos sobre este mismo.

Otro punto importante a resaltar es que este laboratorio a sido desarrollado en base a los laboratorios pasados, tomándoles como punto de referencia y pauta para la funcionalidad e inicio del desarrollo que se presenta a continuación.

Desarrollo de aplicación

Para la elaboración del entregable se hizo uso de un repositorio base proporcionado por el profesor de la asignatura, proveyendo una serie de scripts elaborados en Python, los cuales fueron modificados para ser completados hacía el propósito de la aplicación en cuestión.

Se modificaron los siguientes archivos:

- *populate.py*
  La utilería proporcionada fue modificada con el fin de insertar datos en las nuevas tablas *tardes_by_a_td*, *tardes_by_a_std* y *tardes_by_a_sd*. Similar al laboratorio anterior, estas tablas deben ser creadas posteriormente en *model.py* (siguiente archivo).
  Manteniendo el propósito original de la herramienta, la función que cumple este archivo es la de generar datos aleatorios con fines demostrativos.

- *model.py*
  Aquí se define el esquema inicial de las tablas *accounts_by_user*, *possitions_by_account* y *tardes_by_a_d*. Dentro de este se definen las nuevas tablas (especificadas en el punto anterior) y los esquemas de las nuevas consultas. También, se anexan un para de nuevas funciones para mostrar los resultados de los puntos faltantes de *menu.py*.

- *app.py*
  Finalmente, se modificó la interfaz de interacción con el usuario para mostrar una aplicación sencilla pero totalmente funcional. De este punto no hay demasiados detalles que listar, simplemente se incluyen sentencias de despliegue visual para la comodidad de los clientes potenciales.

Resultados

Según lo solicitado para la práctica, se puede decir que se ha logrado cumplir con el objetivo de este laboratorio, con consultas como las siguientes:

```
SELECT_ACCOUNT_POSITIONS = """
    SELECT symbol, quantity
    FROM positions_by_account
    WHERE account = ?
"""

SELECT_TRADES = """
    SELECT toDate(trade_id) as date, amount, price, shares, symbol, type
    FROM trades_by_a_d
    WHERE account = ?
"""

SELECT_TRADES_DATE_RANGE = """
    SELECT toDate(trade_id) as date, amount, price, shares, symbol, type
    FROM trades_by_a_d
    WHERE account = ?
    AND trade_id >= minTimeuuid(?)
    AND trade_id <= maxTimeuuid(?)
"""

SELECT_TRADES_TYPE = """
    SELECT toDate(trade_id) as date, amount, price, shares, symbol, type
    FROM trades_by_a_d
    WHERE account = ?
    AND type = ? ALLOW FILTERING
"""

SELECT_TRADES_SYMBOL = """
    SELECT toDate(trade_id) as date, amount, price, shares, symbol, type
    FROM trades_by_a_d
    WHERE account = ?
    AND symbol = ? ALLOW FILTERING
"""
```

También, con el fin de mantener un paradigma de programación funcional, se construyó la siguiente función con parámetros opcionales:

```
def get_trade_history(session, account, date_range=None, t_type=None, symbol=None):
    log.info(f"Retrieving trade history from account {account} || " +
             f"Date range: start = {date_range[0]}, end = {date_range[1]} " if date_range else ' ' +
             f"Type: {t_type} " if t_type else ' ' +
             f"Symbol: {symbol} " if symbol else ' ')

    stmt = None
    rows = None

    if (not date_range and not t_type and not symbol):
        stmt = session.prepare(SELECT_TRADES)
        rows = session.execute(stmt, [account])
    elif (date_range and len(date_range) == 2):
        stmt = session.prepare(SELECT_TRADES_DATE_RANGE)
        s = datetime.strptime(date_range[0], '%Y-%m-%d')
        e = datetime.strptime(date_range[1], '%Y-%m-%d')
        rows = session.execute(stmt, [account, s, e])
    elif (t_type):
        stmt = session.prepare(SELECT_TRADES_TYPE)
        rows = session.execute(stmt, [account, t_type])
    elif (symbol):
        stmt = session.prepare(SELECT_TRADES_SYMBOL)
        rows = session.execute(stmt, [account, symbol])
    else:
        print('Invalid arguments. Try again.')
        return

    print(f"=== Showing trade history for account {account} ===")
    for row in rows:
        print(f"\tDate: {row.date}; amount: {row.amount}; price: {row.price}; shares: {row.shares}; symbol:
{row.symbol}; type: {row.type}")
    print('\n')
```

Como se puede ver, existe una llamada al método *info* de una clase log, perteneciente al módulo logging, el cual simplemente genera y/o anexa registros de operaciones a un archivo que lleva el control de las acciones de los usuarios, como ingreso, consultas, errores, y también conexiones entre los sistemas que dan soporte a la aplicación.

Conclusión

Como se venía presagiando desde el inicio del módulo en curso, la aplicación “creada” brinda un ejemplo casi perfecto de cómo interactuar con este tipo de bases, logrando el propósito de la materia de una manera sumamente didáctica y fungiendo también como un punto de práctica de los principios de programación aplicada.

Es importante visualizar cómo es posible realizar aplicaciones a través de diversos lenguajes, ya que estas tecnologías están creadas para proveer herramientas útiles para el manejo de datos en el ámbito de las tecnologías de la información.

Para finalizar con este módulo, es importante recalcar el uso de Cassandra, puesto que, como opinión personal, las bondades de esta herramienta no fueron explotadas del todo, dejando campos de aprendizaje por aprender y oportunidades por aprovechar en cuanto a las bases de datos columnares.