Informe de cobertura

Entregable 3 - Programación Avanzada

DSL

Felipe Ham - Santiago Lozano

Descripción General

El proyecto consiste en el procesamiento de consultas que permite interpretar y traducir un lenguaje de consultas uruguayo (USQL) a SQL estándar para interactuar con una base de datos SQLite. El flujo del proyecto incluye el análisis de la sintaxis de las consultas, su traducción a SQL si se encuentran en USQL, y la ejecución de las consultas en una base de datos, con una salida clara de los resultados. Este trabajo incluye un lexer y un parser para el análisis y la interpretación de las consultas, una base de datos SQLite para almacenar y recuperar datos, y una API fluida para ejecutar y manejar las consultas.

Lexer

El lexer utiliza ply.lex para dividir la consulta en tokens y analizar su estructura léxica, es decir, identificar palabras clave y operadores de USQL. Este paso permite diferenciar consultas en SQL y USQL, procesando las palabras y símbolos específicos de cada tipo de consulta.

Parser

El parser convierte los tokens del lexer en una estructura lógica. Usa un conjunto de reglas definidas para interpretar la sintaxis de USQL, traduciendo las consultas USQL a SQL. Por ejemplo, una consulta USQL que solicita "TRAEME TODO DE LA TABLA usuarios;" se transforma en una consulta SQL equivalente como SELECT * FROM usuarios;. Así, el parser permite realizar traducciones automatizadas y correctas de consultas en USQL a consultas SQL.

Cobertura de Consultas Soportadas

1. **Consultas de Selección** (SELECT / TRAEME)
    - SQL a USQL: Traduce consultas SQL SELECT con soporte para:
        - Selección de todas las columnas (*).
        - Selección de columnas únicas (DISTINCT).
        - Funciones de agregación simples (COUNT(*)).
        - Uso de WHERE y JOIN para condiciones.
        - Agrupación (GROUP BY) y filtrado (HAVING).
    - USQL a SQL: Traduce TRAEME en USQL de forma similar, soportando:
        - Selección de todos los elementos (TODO).
        - Selección de elementos distintos (LOS_DISTINTOS).
        - Función CONTANDO(TODO) para contar registros.
2. **Consultas de Inserción** (INSERT / METE_EN)
    - SQL a USQL: Traduce comandos SQL INSERT INTO a METE_EN, soportando la especificación de columnas y valores en la misma consulta.
    - USQL a SQL: Traduce METE_EN en USQL para realizar inserciones, permitiendo incluir tanto columnas como valores.
3. **Consultas de Actualización** (UPDATE / ACTUALIZA)
    - SQL a USQL: Traduce comandos UPDATE SQL a ACTUALIZA, permitiendo actualizar registros específicos con el uso de SET y WHERE.
    - USQL a SQL: Traduce ACTUALIZA en USQL de manera análoga, permitiendo definir tanto los valores de actualización como las condiciones.
4. **Consultas de Eliminación** (DELETE / BORRA_DE_LA)

    - SQL a USQL: Traduce comandos DELETE FROM a BORRA_DE_LA, permitiendo aplicar condiciones (WHERE).
    - USQL a SQL: Traduce BORRA_DE_LA en USQL, admitiendo el uso de condiciones.
5. **Alteración de Tablas** (ALTER TABLE / CAMBIA_LA_TABLA)
    - SQL a USQL: Traduce ALTER TABLE para agregar (ADD COLUMN) o eliminar columnas (DROP COLUMN).
    - USQL a SQL: Traduce CAMBIA_LA_TABLA con operaciones de AGREGA_LA_COLUMNA y ELIMINA_LA_COLUMNA.

Base de Datos
Para la base de datos decidiomos utilizar SQLite, una base de datos liviana y fácil de configurar. En el archivo *createDB.py* implementamos un código que al ejecutarse genera un archivo local (*mi_base_de_datos.db*) que será la base de datos del entregable.

Fluent API
La clase FluentQueryAPI es la capa de ejecución de consultas y traducción. Esta API permite que las consultas en USQL se identifiquen, traduzcan y ejecuten automáticamente en la base de datos. En caso de una consulta en SQL, la API ejecuta la consulta directamente, y en el caso de una consulta en USQL, primero la traduce a SQL antes de ejecutarla. La clase tiene métodos adicionales, como clean_query, que garantizan que el formato de la consulta sea consistente y seguro antes de ser procesada o ejecutada. Siempre hay que agregar el punto y coma al final de la consulta. Aclaración: Si se corre denuevo la fluent API mientras ya se está corriendo dara un error, pero luego al darle a correr denuevo ahi se arregla.

Informe de Cobertura del Proyecto

La cobertura total del proyecto es del **94%**, indicando un nivel de pruebas muy alto y exhaustivo en la mayoría de las funciones. A continuación, se detalla la cobertura por archivo:

1. `test_parser.py`:
    - **Cobertura**: 87%
    - **Líneas faltantes**: 28 líneas sin cubrir.
    - **Análisis**: Las líneas faltantes en este archivo pertenecen principalmente a casos extremos o situaciones inusuales en el parser, que no afectan de manera significativa el flujo principal de la aplicación.
2. `parser.py`:
    - **Cobertura**: 97%
    - **Líneas faltantes**: 5 líneas sin cubrir.
    - **Análisis**: Las líneas omitidas son irrelevantes para el flujo principal y corresponden a condiciones de borde en reglas gramaticales o manejo de errores poco comunes.
3. `test_lexer.py`:

- **Cobertura**: 93%
- **Líneas faltantes**: 2 líneas sin cubrir.
- **Análisis**: Las funciones clave del lexer están probadas en su totalidad; las omisiones no afectan el comportamiento general.

4. `lexer.py` y `parsetab.py`:
   - **Cobertura**: 100%
   - **Análisis**: Ambos archivos están completamente cubiertos, lo que asegura que sus funciones y definiciones están bien probadas.

Conclusión

El proyecto cuenta con una cobertura excelente del **94%**, donde la mayoría de las omisiones corresponden a casos poco comunes o de bajo impacto en la funcionalidad principal. Esto garantiza una alta confiabilidad en las pruebas y estabilidad en el funcionamiento del parser y lexer principales.