Relational

A DSL for using Relational Algebra on top of SurrealDB.
Author: Warul Kumar Sinha (2023201045)
GitHub (https://github.com/wksinha/relational)

Motivation

- SurrealDB is a multi-model database.
- It supports both relational and graph queries.
- This support makes it suitable for learning to transition from relational to graph queries.
- This makes it suitable for a learning environment such as academia.
- We design and run a Relational Algebra -> SQL converter on top of SurrealDB.
- The Domain Specific Language for students aims to be the first step in learning DBMS.
- Combined with SurrealDB's multi-model architecture, it builds the right support for learning the transition.

Technology

- Language: Python
- Modules: Lark
    - Run the following in the `relational` directory:

```
pip install -r requirements.txt
```

- Database: SurrealDB
  - Start the database, run:

  ```
  surreal start memory -A --user root --pass root
  ```

  - Run SurrealQL:

  ```
  surreal sql --endpoint http://localhost:8000 --username root --password root
  ```

  - Run the following to populate the DB data from the `relational/src/main` directory:

  ```
  bash init.sh
  ```

- Testing: Bash

Methodology

- Studied SurrealDB syntax and semantics, and references for writing a DSL, specifically Relational Algebra to SQL.
  - Existing works were mostly inconsistent, error prone (improper handling of edge cases) and lacked testing.
  - Referenced multiple works (including those in Haskell/Java), none in Python.
  - Pure Haskell was a good option considering the recursive data structures.
    - This is taken care of with the Lark module in Python, bringing the DSL in a more complete form, to a simpler language.
  - Finalized the expected structure of the Grammar.
  - Designed the application pipeline.

- The application runs in a pipelined manner.
- The RA2SQL module takes in relational algebra (ignoring whitespaces, examples given) and outputs SQL queries.
    - The RA2SQL module, written in Python, uses Lark to create a parser from a grammar.
    - The grammar for the DSL is described using EBNF.
    - A custom interpreter then traverses the parsed AST to create the desired SQL query.
        - An alternative here could be to use Lark's inbuilt transformer module, but that has separate challenges.
- This output is then passed on to SurrealDB and the output is given to the user.
- Error handling done to enforce sane inputs and processing.

Testing

- Used bash scripts to run the application against custom testcases for the DSL.
    - Generated output was compared against expected output.
    - Examples:

RA:

```
 SELECTION (*) (student)
SELECTION (year = 2) (SELECTION  (sid > 10) (student))
PROJECTION (year, sid) (student)
PROJECTION (year) (SELECTION (year=1) (student))
UNION ( SELECTION (year=1) (student) ) ( SELECTION (year=2) (student) )
DIFFERENCE (PROJECTION (sid) (SELECTION (*) (enrollment))) (PROJECTION (sid)
PRODUCT (SELECTION (*) (student)) (SELECTION (*) (course))
```

SQL (Generated):

```
SELECT * FROM (student);
SELECT * FROM (SELECT * FROM (student) WHERE sid > 10) WHERE year = 2;
SELECT year, sid FROM (student);
SELECT year FROM (SELECT * FROM (student) WHERE year=1);
(SELECT * FROM (student) WHERE year=1) UNION (SELECT * FROM (student) WHERE ye
(SELECT sid FROM (SELECT * FROM (enrollment))) EXCEPT (SELECT sid FROM (studen
(SELECT * FROM (student)) CROSS JOIN (SELECT * FROM (course));
```

- From the `relational/src/tests` directory, run:

```
bash test_dsl.sh
```

Expected Output:

```
Test passed: difference_test_1.in
Test passed: product_test_1.in
Test passed: projection_test_1.in
Test passed: projection_test_2.in
Test passed: selection_test_1.in
Test passed: selection_test_2.in
Test passed: union_test_1.in
```

- Checked error-free execution of the pipeline with SurrealDB for supported operations.

  - To check the pipeline, from the `relational/src/main` directory, run:

```
bash check_pipeline.sh
```

Expected Output:

```
Pipeline passed for selection_test_1.in
Pipeline passed for selection_test_2.in
Pipeline passed for projection_test_1.in
Pipeline passed for projection_test_2.in
```

Results

- Created a DSL that takes Relational Algebra and converts it into SQL queries.
- These queries are then run on SurrealDB.
- Handles SELECTION, PROJECTION, UNION, DIFFERENCE, CARTESIAN PRODUCT operations alongside NESTED queries.
- Note: SurrealDB does not support UNION/CARTESIAN PRODUCT operations, but appropriate SQL is generated by the DSL. According to the docs:

    - Instead of pulling data from multiple tables and merging that data together SurrealDB allows you to traverse related records efficiently using record links and graph connections.

Future Scope

- Current implementation has redundancies in the generated SQL queries.

    - Example:

    ```
    SELECT (age, id) FROM (SELECT * FROM student);
    ```

    vs

    ```
    SELECT (age, id) FROM student;
    ```

    - This can be improved to identify and remove redundancies using appropriate reduction rules.

- A custom Interpreter is implemented, we can use Lark's Transformer module instead to make it more extensible.

Challenges

- Familiarization with SurrealDB syntax. SurrealDB does not follow typical DML syntax. We can see an example in the `init.surql` file.
- Deciding on the right tool (PLY vs Lark vs PyParsing) - testing them and familiarization with Lark finally.
- Lark contains the Transformer module that helps write an Interpreter for a DSL.
    - It was challenging to set up and a custom interpreter was written instead.
- Debugging. Debugging specific issues on the tree was challenging due to feast-or-famine of logging information.
    - Added custom logs for specific operations which helped, but intermediate output was not exactly human readable.
    - Example:

```
start
[Tree('query', [Tree('selection', [Tree('condition', [Token('STRING', 'year'),

query
[Tree('selection', [Tree('condition', [Token('STRING', 'year'), Token('STRING'

selection
[Tree('condition', [Token('STRING', 'year'), Token('STRING', '='), Token('STRI

query
[Tree('selection', [Tree('condition', [Token('STRING', 'sid'), Token('STRING',

selection
[Tree('condition', [Token('STRING', 'sid'), Token('STRING', '>'), Token('STRIN

query
[Tree('relation', [Token('KEY', 'student')])]

condition
[Token('STRING', 'sid'), Token('STRING', '>'), Token('STRING', '10')]

condition
[Token('STRING', 'year'), Token('STRING', '='), Token('STRING', '2')]

SELECT * FROM (SELECT * FROM (student) WHERE sid > 10) WHERE year = 2;
```

References

- https://www.doc.ic.ac.uk/~pjm/teaching/student_projects/gc106_report.pdf (https://www.doc.ic.ac.uk/%7Epjm/teaching/student_projects/gc106_report.pdf)
- https://lark-parser.readthedocs.io/en/latest/examples/index.html (https://lark-parser.readthedocs.io/en/latest/examples/index.html)
- https://blog.erezsh.com/how-to-write-a-dsl-in-python-with-lark/ (https://blog.erezsh.com/how-to-write-a-dsl-in-python-with-lark/)
- https://gist.github.com/PH111P/7c8b529c0293d8c35adc#file-relalgsql-hs (https://gist.github.com/PH111P/7c8b529c0293d8c35adc#file-relalgsql-hs)

- https://surrealdb.com/docs/surrealdb/introduction (https://surrealdb.com/docs/surrealdb/introduction)