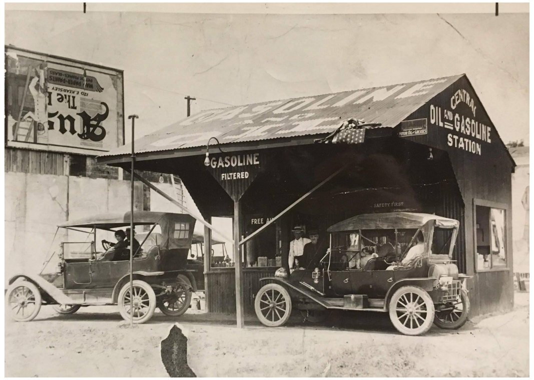


DieSL

Final Presentation
Group DSL Essentials

6 September 2021

Goals

- Create a DSL for table processing that is compatible with the Nim language and easy to use for non-programmers

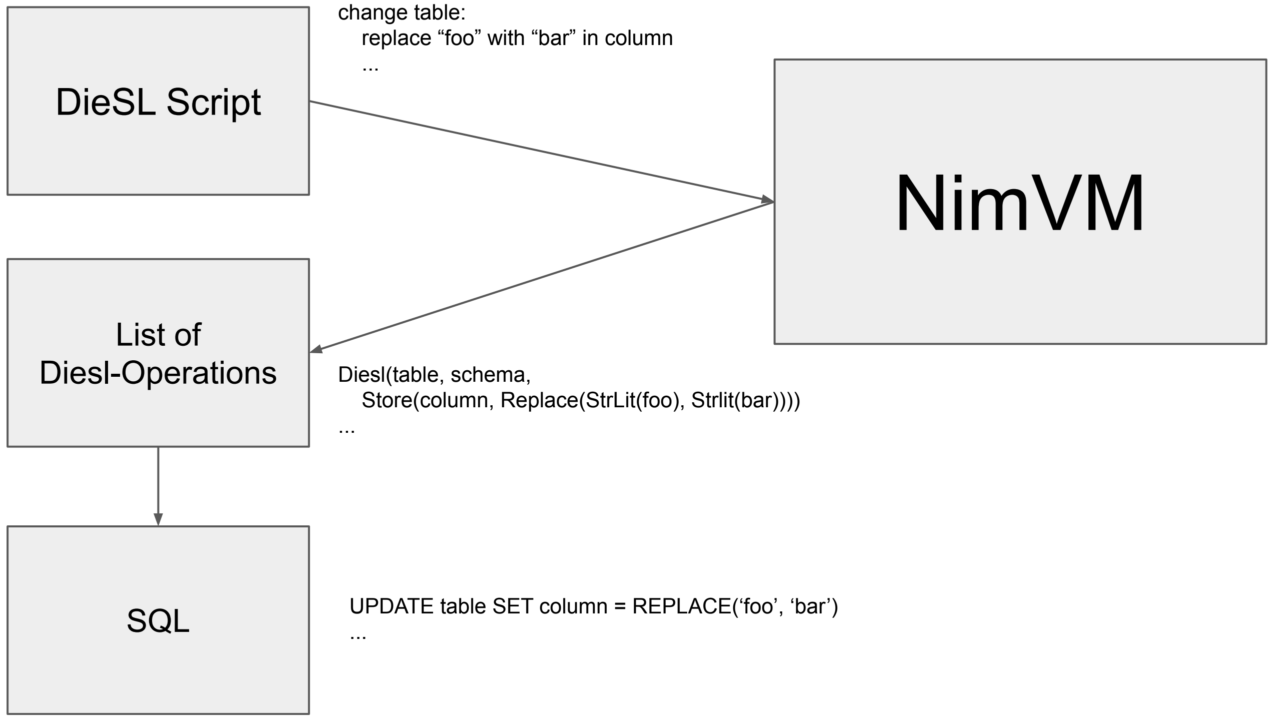
Summary of Results

- Main goal has been achieved!
- DieSL is a Nim compatible DSL that is easy to use
- ~1.6k LoC, 400 Commits, 58 Closed Issues
- 50 Tests with 724 LoC
- Estimated test coverage: ~85%

Feature Overview

- Nim API for data manipulation
- Natural API based on `change` block macro
- Execution through Nim and NimScript
- Translation of operations to Sqlite compatible SQL code
  - supports both directly executable queries and creation of SQL views
  - combination of operations into one SQL statement where possible
- String operations:
  - string literals
  - trim, uppercase, lowercase
  - concatenation, split, replace, remove
  - padding, substring
  - pattern matching and replacement (regex and predefined patterns)

Architecture



```
change table:
    replace “foo” with “bar” in column
    ...
```

Macro Expension

```
table.column = table.column.replace(“x”, “y”)
...
```

```
Diesl(table, schema,
    Store(column, Replace(StrLit(foo), Strlit(bar))))
...
```

```
UPDATE table SET column = REPLACE(‘foo’, ‘bar’)
...
```

Natural Syntax: Change Block

```
change db.students:
  trim beginning of name
  replace "foo" with "bar" in name
  take 1 to 3 from name
```

Nim Syntax

```
db.students.name = db.students.name.trim(left)

db.students.name = db.students.name.replace("foo", "bar")

db.students.name = db.students.name[0..2]
```

Generated Sqlite

```
CREATE VIEW students_vrsmi8c0dwojyb1bg_0
  (name, firstName, secondName, lastName, age)
AS SELECT
  SUBSTR(REPLACE(LTRIM(name), 'foo', 'bar'), 0, 2),
  firstName, secondName, lastName, age
FROM students
```

What Could Not Be Implemented?

- Intuitive operation-level macro building blocks not possible:
  - Nim limitation: parsing precedence / keyword overwriting
  - Solution: `change`-block as entry point and parser, leaving input as-is on error

- Map-Reduce with anonymous NimScript functions:
  - Too complex for first iteration of DieSL, likely inefficient
  - But you can expose Nim functions as SQL operations (`exportToSqlite3` macro)

- Automatic test coverage reports (CI pipeline)
  - “coco” broke mid-project due to cryptic “lcov” errors we weren’t able to fix

Technical Difficulties

- NimScript/VM
  - Running database manipulations inside the NimVM was slow and hard to implement
  - Solution: running DiesL-Script in NimVM generates object representing the changes
  - Changes are translated into SQL inside the binary (not the VM)
- Clean Code for parsing DiesL
  - A lot of parsing approaches did not work
    - Parser generator: work with string not Nim AST
    - Parser combinator: not feasible with Nim’s type system
  - Solution: better pattern matching using fusion/matching
- SQLite has no regex functions without loading extensions

Extending Sqlite through Nim

```nim
import exporttosqlite3
import db_sqlite

proc myNimFunction(greeting: string, name: string, age: int32): string {.exportToSqlite3.} =
  greeting & " " & name & " (age " & $age & ")"

when isMainModule:
  let db = open("test.db", "", "", "")
  defer:
    db_sqlite.close(db)
  db.registerFunctions()
  db.exec(sql"DROP TABLE IF EXISTS students")
  db.exec(sql"CREATE TABLE students (name TEXT, age INT)")
  db.exec(sql"INSERT INTO students (name, age) VALUES (?, ?), (?, ?)",
      "Peter Parker", 23, "John Good", 19)
  db.exec(sql"UPDATE students SET name = myNimFunction('Hello', name, age)")
```

Links

- Repository:

  https://gitlab.com/pvs-hd/ot/diesl

- Documentation, tutorials, accounting, demo etc.:

  https://gitlab.com/pvs-hd/ot/diesl/-/blob/develop/README.md

- Our library for extending Sqlite through nim:

  https://github.com/niklaskorz/nim-exporttosqlite3

DEMO HERE

“That’s all Folks!”

Backup Slides

Natural Syntax: Change Block

Additional syntax for working on a single column:

```
# Equivalent to previous slide

change name of db.students:

  trim beginning

  replace "foo" with "bar"

  take 1 to 3
```

Sqlite Views Target

- Table access map contains all views belonging to a table:

  `"students": @["students_vrsmi8c0dwojyb1bg_0"]`

- Sequence of views contains all views of one DSL execution:

  `@["students_vrsmi8c0dwojyb1bg_0"]`

- `removeSqliteViews(views, tableAccessMap)` generates `DROP VIEW` queries in reverse order and deletes all removed views from the access map

Pattern Matching

```
# Determine tweet category based on first hashtag

db.tweets.category = db.tweets.text.extractOne("{hashtag}")

CREATE VIEW tweets_16mf64iomruh1wmc6_0

  (text, category)

AS SELECT

  text, extractOne(text, '(?<=\s|^)#(\w*[A-Za-z_]+\w*)')

FROM tweets
```