

Transpilation

From a SQL dialect to another

Florent JARDIN, Étienne BERSAC

March 13, 2024

Who are we?

- [@bersace](#) Marmotte 🥔. Prêt à livrer ! code 🚩🗑️
- [@fljdin](#) Database inspired, powered by passion and curiosity

Contents

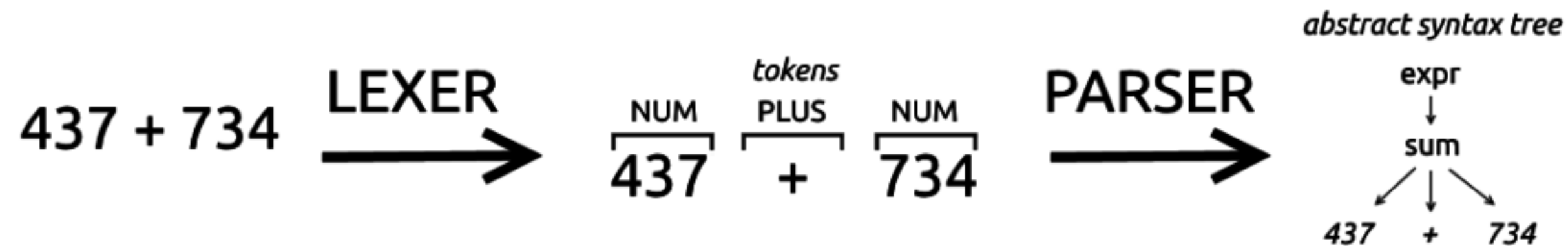
- Transpilation
 - Concepts
 - How it works
- transqlate
 - Purpose and objectives
 - Rewriting engine
 - Case studies

1) Transpilation

Lexico-grammatical analysis

A language respects a lexicon and a syntax

- Analysis transforms code into *tree*.
- The *lexer* splits the expression into *tokens*.
- The *parser* groups the *tokens* into *nodes*.



Usecase: compiling, interpreting

The basis of all computer languages

- Lexical and syntactic validation
- Compile source code into machine code
 - *gcc, gc, javac, WebAssembly*
- Interpreting and executing scripts
 - *shell, perl, python*

Usecase: IDE, doc

Code editing assistance

- Syntax highlighting : *tree-sitter*, *pygments*
- Code completion : *IntelliSense*
- Code reworking (renaming, extraction)
 - *LSP : Language Server Protocol*
- Static analysis : *golangci-lint*, *flake8*
- Documentation: *godoc*, *doxygen*, *docstring*

Usecase: code transformation

- Reformatting: *go fmt*, *prettier*, *black*, etc.
- Minification
- Optimization

Usecase: transpiler

Converting code from one language to another

- Hyphenation: translate + compile
- *TypeScript*, *CoffeScript* to JavaScript
- *SASS* to CSS
- Python 2 to Python 3: *pyupgrade*
- ... One SQL dialect to another

SQL dialects

- Standard ISO/IEC 9075-1:2023
- Historical syntaxes, prior to the standard
- Interpretation or extension of the standard: `NULL`
- Own functions and system catalogs

```
SELECT `price` * IFNULL(`discount`, 1) FROM `products`; -- MySQL
SELECT [Price] * ISNULL([Discount], 1) FROM [Products]; -- SQL Server
SELECT "PRICE" * NVL("DISCOUNT", 1) FROM "PRODUCTS"; -- Oracle
SELECT "price" * COALESCE("discount", 1) FROM "products"; -- PostgreSQL
```

2) transqlate

- Our new contribution. ⚠ Alpha ⚠
- Target dialect: PostgreSQL
- CLI & API Go
- Based on parsing

Requirements

- Transpile arbitrary SQL code
- Extensive code rewriting
- Preserve indentations, breaks and comments
- Simplicity of implementation

Out of consideration

- Performance
- Interpretation and validation
- Query optimization

Validity constraints

- Presumption of input validity
- Rewritten code must be grammatically valid
- But not necessarily compatible (until manual review)

Reliability constraints

- Error handling
- Incomplete, impossible or ambiguous translation
- *Lost in translation*
- Precise indication of error code
- Help teams to take over manually

Preservation constraints

- Indentations
- Breaks
- Comments

Rewriting engine

Transpilation at different stages of analysis

- Token rewriting
- Node or branch rewriting
- Rewriting the whole tree

Rewriting tokens

The `Token` structure is retained:

- The type
 - `Keyword`, `Identifier`, `Operator`, `String`, ...
- The original code, as written
 - `SELECT, "id", where, employees`
- The standardized code
 - `SELECT, ID, WHERE, EMPLOYEES`
- Blank characters: space, comments before and after

Rewriting identifiers

Default: lowercase identifiers Objects are renamed to lowercase on migration.

```
SELECT ID, UPPER("Name"), "PHONE" FROM Contacts; -- Oracle
```

becomes

```
SELECT id, upper("Name"), phone FROM contacts; -- PostgreSQL
```

Rewriting identifiers

If objects are migrated without renaming. Preserve Oracle case with `--preserve-case`:

```
SELECT ID, UPPER("Name"), "PHONE" FROM Contacts;
```

becomes

```
SELECT "ID", upper("Name"), "PHONE" FROM "CONTACTS";
```

Rewriting nodes

- Each node of the tree contains:
 - the tokens for writing the expression
 - child nodes
- A rule must:
 - test that a node must be translated
 - apply the translation heuristic

Rewriting nodes

```
# Oracle: SELECT SYSDATE FROM DUAL
ast.Select:
  Select: Keyword "SELECT"
  List ast.Leaf: Identifier "SYSDATE"
  From ast.From:
    From: Keyword "FROM"
    Tables ast.Grouping:
      Items:
        - ast.Leaf: Identifier "DUAL"

# Postgres: SELECT LOCALTIMESTAMP
ast.Select:
  Select: Keyword "SELECT"
  List ast.Leaf: Identifier "LOCALTIMESTAMP"
```

Example : TRUNC

```
-- Oracle  
SELECT TRUNC(HIRED_DATE, 'Y') FROM EMPLOYEES;
```

becomes

```
-- PostgreSQL  
SELECT date_trunc('year', hired_date) FROM employees;
```

Translation error

```
-- Oracle  
SELECT TRUNC(HIRED_DATE, DATEFMT) FROM EMPLOYEES;
```

becomes

```
-- PostgreSQL  
-- TRANSLATION ERROR at +1:8: not a literal format rule="replace trunc()"  
SELECT date_trunc(datefmt, hired_date) FROM employees;
```


Simple outer join

```
SELECT *  
FROM employees, departements  
WHERE employees.deparment_id = departments.id (+);
```

becomes

```
-- PostgreSQL  
SELECT *  
FROM employees  
LEFT OUTER JOIN departments ON employees.department_id = departments.id;
```

SELECT * with join inversion

```
-- Oracle  
SELECT * FROM employees, jobs  
WHERE jobs.id = employees.job_id(+);
```

becomes

```
-- PostgreSQL  
SELECT * FROM employees  
RIGHT OUTER JOIN jobs ON jobs.id = employees.job_id;
```

Composite join

```
-- Oracle
SELECT DISTINCT job.name
FROM employees, jobs
WHERE employees.job_id(+) = jobs.id AND employees.salary(+) > 2000;
```

becomes

```
-- PostgreSQL
SELECT DISTINCT job.name
FROM jobs
LEFT OUTER JOIN employees ON employees.job_id = jobs.id AND employees.salary > 2000;
```

Hierarchical join

```
-- Oracle
SELECT empno, ename, job, mgr
FROM emp
START WITH mgr IS NULL
CONNECT BY PRIOR empno = mgr
```

becomes

```
-- PostgreSQL
WITH RECURSIVE hierarchy(empno, ename, job, mgr) AS (
    SELECT empno, ename, job, mgr
    FROM emp
    WHERE mgr IS NULL
    UNION ALL
    SELECT recursion.empno, recursion.ename, recursion.job, recursion.mgr
    FROM emp AS recursion
    JOIN hierarchy AS "prior"
    ON "prior".empno = recursion.mgr
)
SELECT empno, ename, job, mgr
FROM hierarchy AS emp
```

Rewriting the entire tree

- Useful for re-indenting code
- By default, naive copy of indentation
- `--pretty` applies *Simon HOLYWELL* style
 - sqlstyle.guide

```
SELECT r.last_name, max(year(championship_date))
FROM champions AS c
JOIN riders AS r ON c.last_name = r.last_name
WHERE c.confirmed = 'Y'
AND riders.age > 30
```

Conclusion

- Simple and powerful
- Reliable
- A contribution to the migration ecosystem
- Joins the [Dalibo Labs](#) family

gitlab.com/dalibo/transqlate

Any questions?