Transpilation

D’un dialecte SQL à un autre

Florent JARDIN, Étienne BERSAC

mercredi 13 mars 2024

Qui sommes-nous ?

- @bersace Marmotte 🥔. Prêt à livrer ! code ⛳🗑️
- @fljdin Database inspired, powered by passion and curiosity

Sommaire

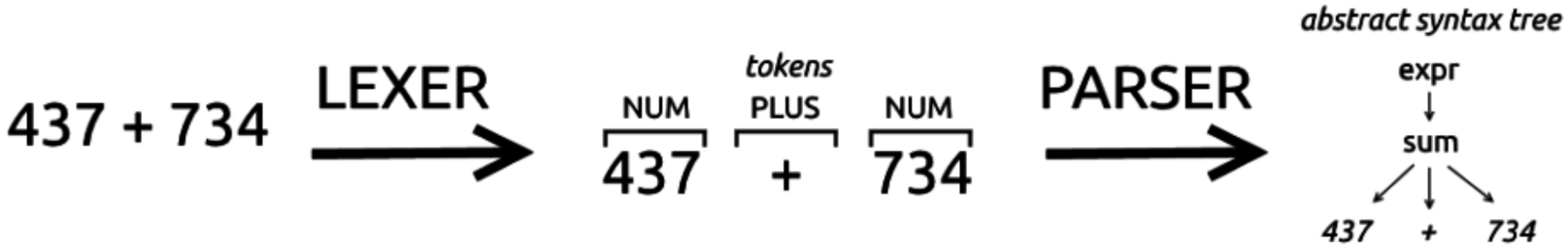
- La transpilation
  - Concepts
  - Usages
- transqlate
  - Raisons d’être et objectifs
  - Moteur de réécriture
  - Cas concrets

1) La transpilation

Analyse lexico-grammaticale

Un langage respecte un lexique et une syntaxe

- L'analyse permet de transformer du code en *arbre*
- Le *lexer* découpe l'expression en *jetons*
- Le *parser* regroupe les *jetons* en *nœuds*



Usages : compiler, interpréter

À la base de tous les langages informatiques

- Validation lexicale et syntaxique
- Compiler du code source en code machine
  - *gcc, gc, javac, WebAssembly*
- Interpréter et exécuter un script
  - *shell, perl, python*

Usages : IDE, doc

Assistance à l'édition de code

- Coloration syntaxique : *tree-sitter*, *pygments*
- Complétion de code : *IntelliSense*
- Remaniement de code (renommage, extraction)
  - *LSP* : *Language Server Protocol*
- Analyse statique : *golangci-lint*, *flake8*
- Documentation : *godoc*, *doxygen*, *docstring*

Usages : transformation du code

- Reformatage: *go fmt*, *prettier*, *black*, etc.
- Minification
- Optimisation

Usage : transpiler

Convertir du code d'un langage à un autre

- Mot-valise : traduire + compiler
- *TypeScript*, *CoffeScript* vers JavaScript
- *SASS* vers CSS
- Python 2 vers Python 3 : *pyupgrade*
- … Un dialecte SQL vers un autre

Dialectes SQL

- Standard ISO/IEC 9075-1:2023
- Syntaxes historiques, antérieures à la norme
- Interprétation ou extension de la norme : `NULL`
- Des fonctions et catalogues systèmes propres

```
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

- Notre nouvelle contribution. ⚠️ Alpha ⚠️
- Dialecte cible : PostgreSQL
- CLI & API Go
- Basé sur l’analyse syntaxique

Besoins

- Transpiler du code SQL arbitraire
- Réécrire lourdement le code
- Préserver indentations, casses et commentaires
- Simplicité de l’implémentation

Hors considération

- Performances
- Interprétation et validation
- Optimisation de requête

Contraintes de validité

- Présomption de validité en entrée
- Le code réécrit doit être grammaticalement valide
- Mais pas forcément compatible (jusqu'à la revue manuelle)

Contraintes de fiabilité

- Gestion d’erreurs
- Traduction incomplète, impossible ou ambiguë
- *lost in translation*
- Indiquer précisément le code en erreur
- Aide aux équipes pour reprendre à la main

Contrainte de préservation

- Indentations
- Casses
- Commentaires

Moteur de réécriture

Transpilation à différentes étapes de l'analyse

- Réécriture des jetons (*tokens*)
- Réécriture des nœuds ou branches
- Réécriture de l'arbre dans son ensemble

Réécriture des jetons

La structure `Token` conserve:

- Le type
  - `Keyword`, `Identifier`, `Operator`, `String`, …
- Le code d'origine, tel qu'il a été écrit
  - `SELECT`, `"id"`, `where`, `employees`
- Le code normalisé
  - `SELECT`, `ID`, `WHERE`, `EMPLOYEES`
- Caractères blancs : espace, commentaires avant et après

Réécriture des identifiants

Par défaut : passer les identifiants en minuscules Les objets sont renommés en minuscules à la migration.

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

devient

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

Réécriture des identifiants

Si les objets sont migrés sans renommage. Préserver la casse Oracle avec `--preserve-case` :

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

devient

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

Réécriture des nœuds

- Chaque nœud de l'arbre contient:
  - les jetons pour écrire l'expression
  - les nœuds enfants
- Une règle doit:
  - tester qu'un nœud doit être traduit
  - appliquer l'heuristique de traduction

Réécriture des nœuds

```
# 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"
```

Exemple : TRUNC

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

devient

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

Erreur de traduction

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

devient:

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

Jointure externe simple

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

devient

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

SELECT * avec inversion de jointure

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

devient

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

Jointure composite

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

devient

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

Jointure hiérarchique

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

devient

```
-- 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
```

Réécriture de l'arbre entier

- Utile pour réindenter le code
- Par défaut, copie naïve de l'indentation
- `--pretty` applique le style de *Simon HOLYWELL*
  - 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 et puissant
- Fiable
- Une contribution à l’écosystème de la migration
- Rejoint la famille Dalibo Labs

gitlab.com/dalibo/transqlate

Questions ?