ADQL PEG Grammar and Validation

IVOA Interoperability Meeting

June 2025 – College Park

Grégory Mantelet


Rewriting the
ADQL grammar

New grammar language: PEG

PEG

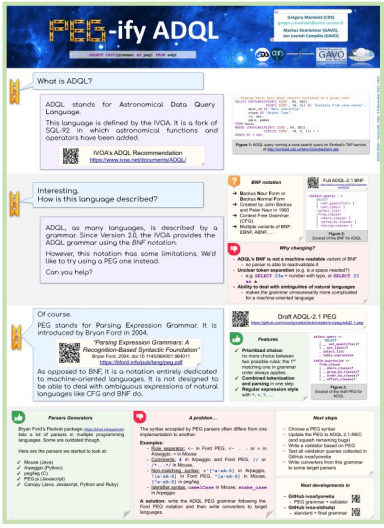
```
QuerySpecification <- WithClause? _
                      QueryExpression _ EOF

QueryExpression <- SelectQuery
                   ( _ set_operator
                     _ set_query_expression )*

[…]
```

[Poster P919]

ADASS XXXIII in Tucson

Novembre 2023

G. Mantelet, M. Demleitner, J.Juaristi Campillo



A PEG grammar already exists

[lyonetia](#)

Thanks to
Jon Juaristi Campillo
(GAVO)

ivoa / lyonetia

Type / to search

Code | Issues 5 | Pull requests | Discussions | Actions | Projects | Wiki | Security 2 | Insights

Files

master

Go to file t

- .github
- notes
- src
  - adql-validator
  - adql
  - bnf
  - docker
  - peg
    - README
    - adql2.1.peg
    - debug.py
    - testpeg.py
  - sql
- .gitignore
- LICENSE
- README.md

lyonetia / src / peg / **adql2.1.peg**

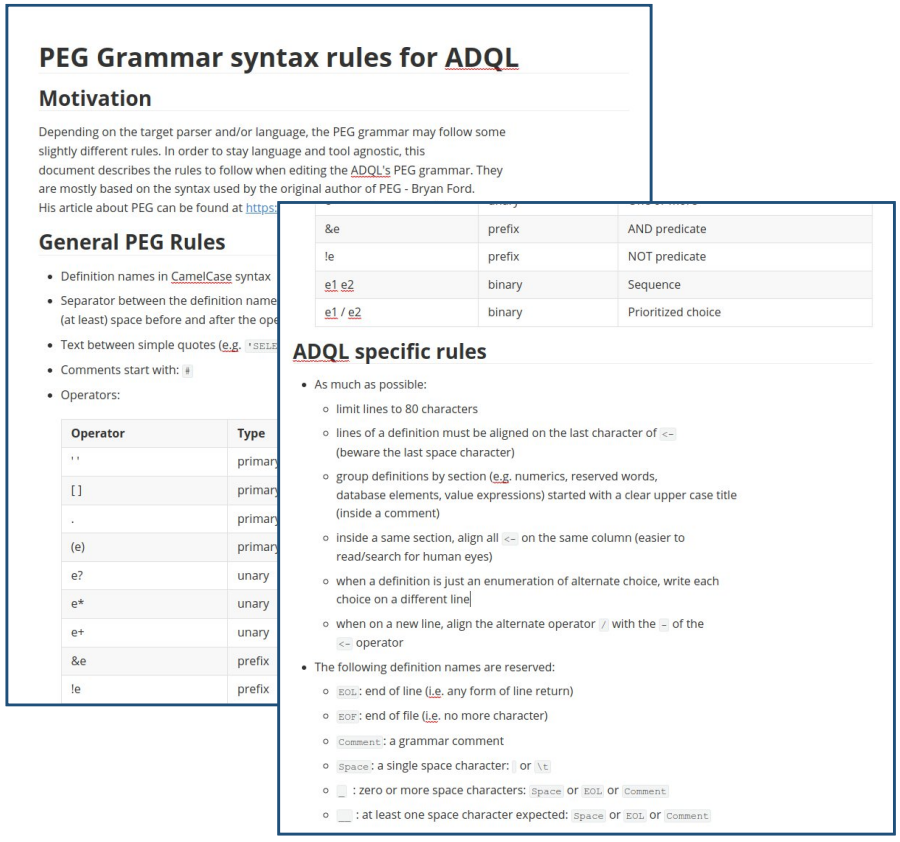
jontxu Fix typo — 216a3d8 · 2 years ago — History

Code | Blame | 702 lines (541 loc) · 14.3 KB — Raw

```
1   # Note: in the actual PEG definition comments start with #
2
3   # ============================ The Grammar's root symbol
4
5   query_specification <-
6           with_clause? _
7           query_expression _ EOF
8
9   # ============================ Top level query parts
10
11  with_clause <-
12          'WITH' _ with_query
13                  _ (',' _ with_query )*
14
15  with_query <-
16          identifier __ as
17          _ '(' _ query_expression _ ')'
18
19  # Rewriting query_expression to have set operator syntax like
20  # postgres.
21  query_expression <-
22          select_query
23          (_ set_operator
24          _ set_query_expression)*
25
26  set_operator <-
27          ('EXCEPT' / 'INTERSECT' / 'UNION') _a
28          (__ 'ALL' _a)?
29
30  set_query_expression <-
31          query_expression
32          / '('   query_expression   ')'
```

...but, need to be rewritten

Syntax guideline



PEG Grammar syntax rules for ADQL

Motivation

Depending on the target parser and/or language, the PEG grammar may follow some slightly different rules. In order to stay language and tool agnostic, this document describes the rules to follow when editing the ADQL's PEG grammar. They are mostly based on the syntax used by the original author of PEG - Bryan Ford.
His article about PEG can be found at https:

General PEG Rules

- Definition names in CamelCase syntax
- Separator between the definition name
(at least) space before and after the ope
- Text between simple quotes (e.g. 'SELE
- Comments start with: #
- Operators:

Operator	Type
''	primary
[ ]	primary
.	primary
(e)	primary
e?	unary
e*	unary
e+	unary
&e	prefix
!e	prefix

&e	prefix	AND predicate
!e	prefix	NOT predicate
e1 e2	binary	Sequence
e1 / e2	binary	Prioritized choice

ADQL specific rules

- As much as possible:
  - limit lines to 80 characters
  - lines of a definition must be aligned on the last character of <- (beware the last space character)
  - group definitions by section (e.g. numerics, reserved words, database elements, value expressions) started with a clear upper case title (inside a comment)
  - inside a same section, align all <- on the same column (easier to read/search for human eyes)
  - when a definition is just an enumeration of alternate choice, write each choice on a different line
  - when on a new line, align the alternate operator / with the - of the <- operator
- The following definition names are reserved:
  - EOL: end of line (i.e. any form of line return)
  - EOF: end of file (i.e. no more character)
  - Comment: a grammar comment
  - Space: a single space character: ' ' or \t
  - _ : zero or more space characters: Space or EOL or Comment
  - __ : at least one space character expected: Space or EOL or Comment

Fix broken rules

- Reserved keywords
- Identifiers
- Spaces

⇒ Rules name mostly different from BNF

There are different syntax

	Java	Python	C	JS	Ruby
Mouse	■				
Arpeggio		■			
peg/leg			■		
Peggy				■	
Canopy	■	■	■		■

Let's try to convert with sed!

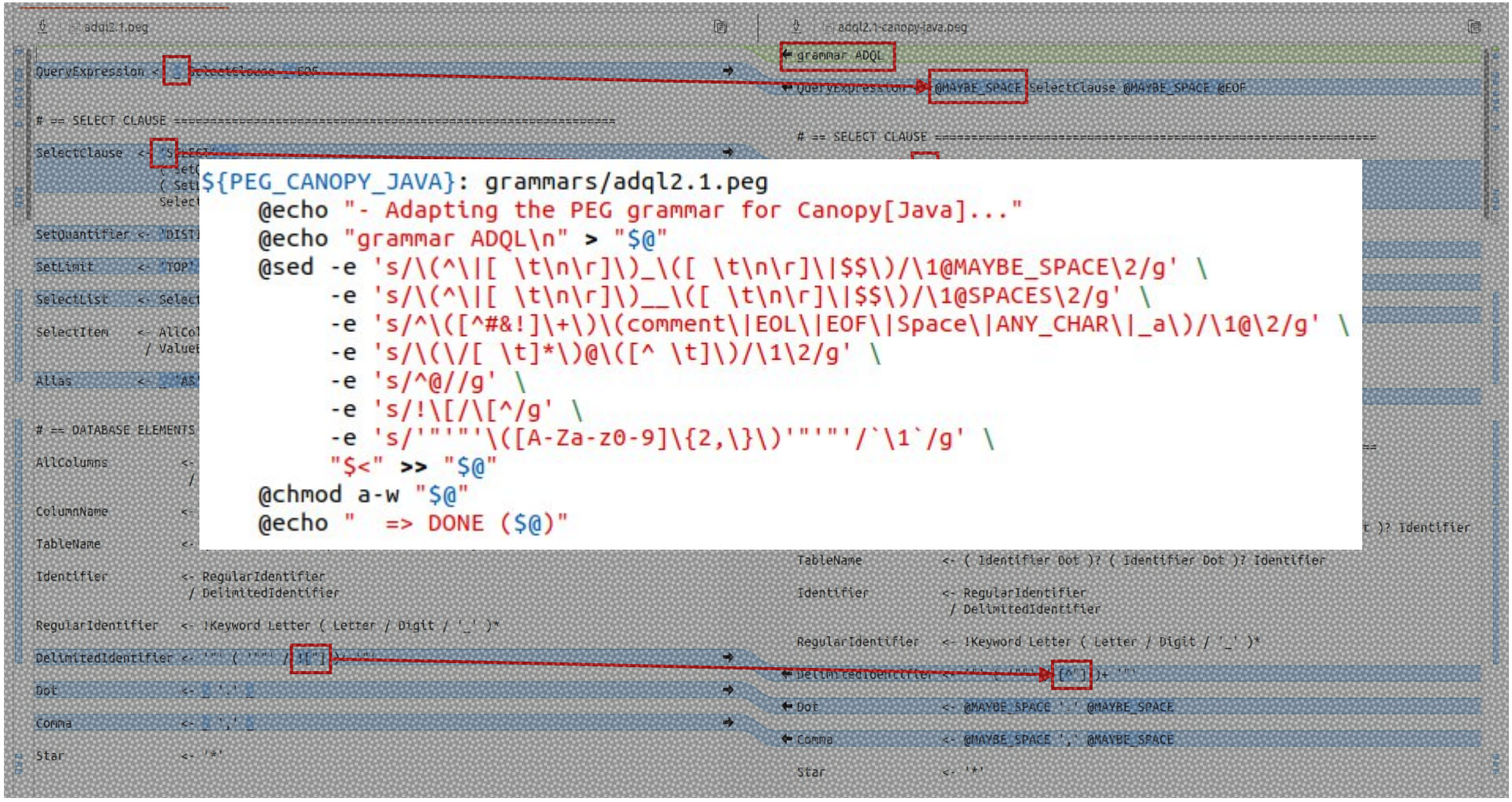
adql2.1.peg

```
QueryExpression <- _ SelectClause _ EOF

# == SELECT CLAUSE ================================================================

SelectClause  <- 'SELECT' __
                 ( SetQuantifier __ )?
                 ( SetLimit __ )?
                 SelectList

SetQuantifier <- 'DISTINCT' / 'ALL'

SetLimit      <- 'TOP' __ UnsignedInteger

SelectList    <- SelectItem ( _ Comma _ SelectItem )*

SelectItem    <- AllColumns
               / ValueExpression Alias?

Alias         <- _ 'AS'? _ Identifier


# == DATABASE ELEMENTS (e.g. tables, columns) ======================================

AllColumns          <- Star
                     / TableName Dot Star

ColumnName          <- ( Identifier Dot )? ( Identifier Dot )? ( Identifier Dot )? Identifier

TableName           <- ( Identifier Dot )? ( Identifier Dot )? Identifier

Identifier          <- RegularIdentifier
                     / DelimitedIdentifier

RegularIdentifier   <- !Keyword Letter ( Letter / Digit / '_' )*

DelimitedIdentifier <- '"' ( '""' / ![\"] )+ '"'

Dot                 <- _ '.' _

Comma               <- _ ',' _

Star                <- '*'
```

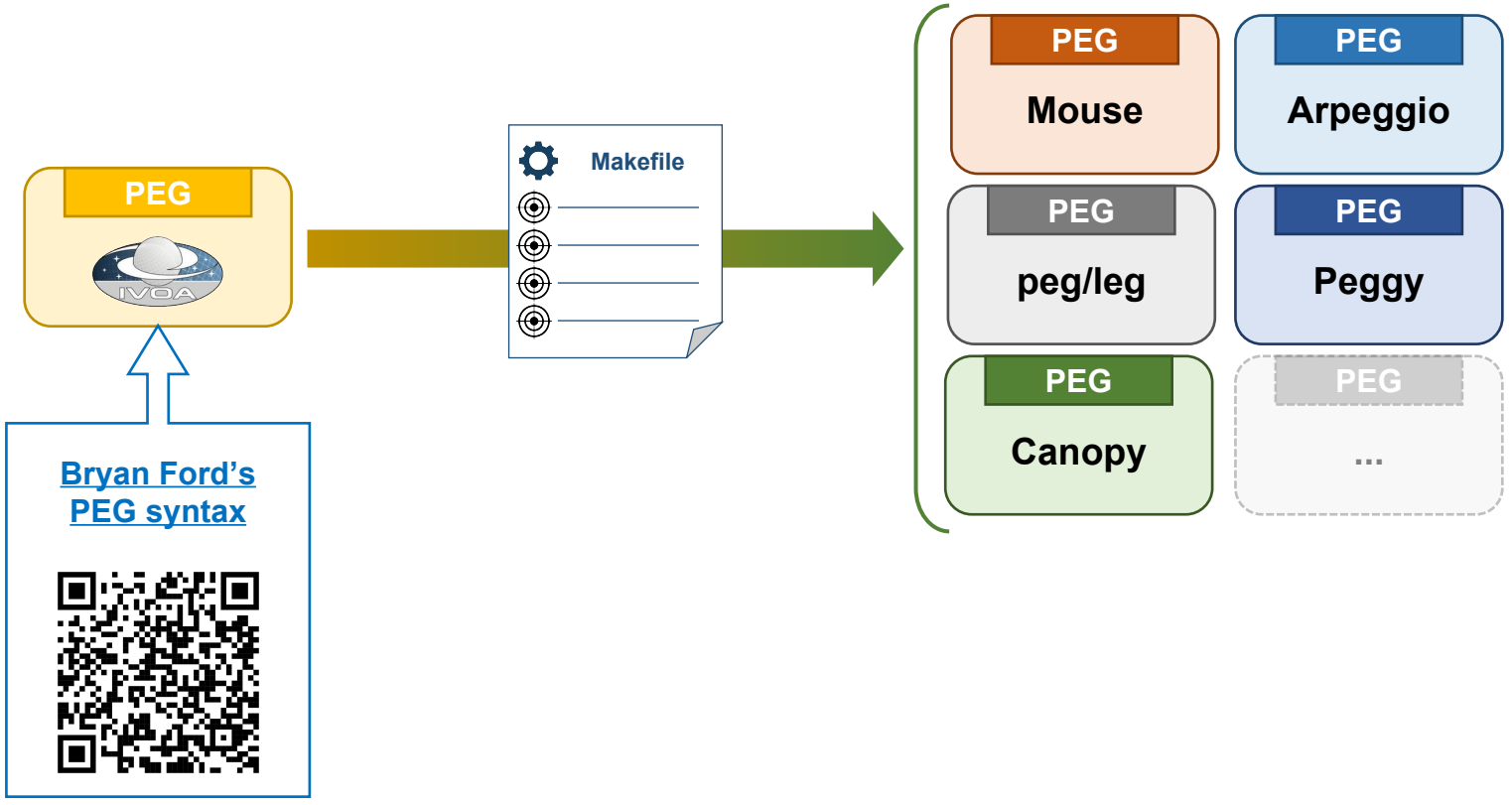
adql2.1-canopy-java.peg

```
grammar ADQL

QueryExpression <- @MAYBE_SPACE SelectClause @MAYBE_SPACE @EOF

# == SELECT CLAUSE ================================================================

SelectClause  <- `SELECT` @SPACES
                 ( SetQuantifier @SPACES )?
                 ( SetLimit @SPACES )?
                 SelectList

SetQuantifier <- `DISTINCT` / `ALL`

SetLimit      <- `TOP` @SPACES UnsignedInteger

SelectList    <- SelectItem ( @MAYBE_SPACE Comma @MAYBE_SPACE SelectItem )*

SelectItem    <- AllColumns
               / ValueExpression Alias?

Alias         <- @MAYBE_SPACE `AS`? @MAYBE_SPACE Identifier


# == DATABASE ELEMENTS (e.g. tables, columns) ======================================

AllColumns          <- Star
                     / TableName Dot Star

ColumnName          <- ( Identifier Dot )? ( Identifier Dot )? ( Identifier Dot )? Identifier

TableName           <- ( Identifier Dot )? ( Identifier Dot )? Identifier

Identifier          <- RegularIdentifier
                     / DelimitedIdentifier

RegularIdentifier   <- !Keyword Letter ( Letter / Digit / '_' )*

DelimitedIdentifier <- '"' ( '""' / [^"] )+ '"'

Dot                 <- @MAYBE_SPACE '.' @MAYBE_SPACE

Comma               <- @MAYBE_SPACE ',' @MAYBE_SPACE

Star                <- '*'
```

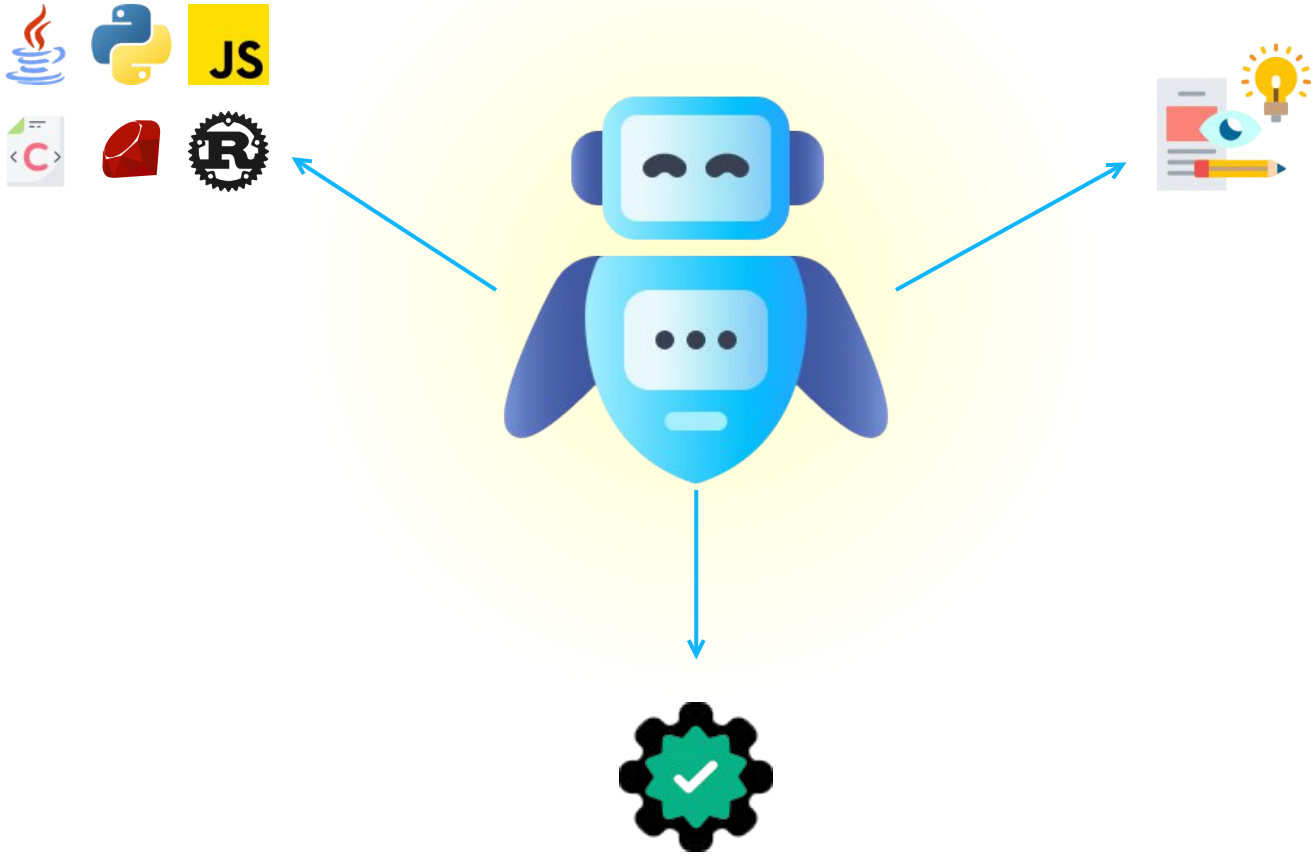
Let’s try to convert with sed!



Solution: use a unique syntax

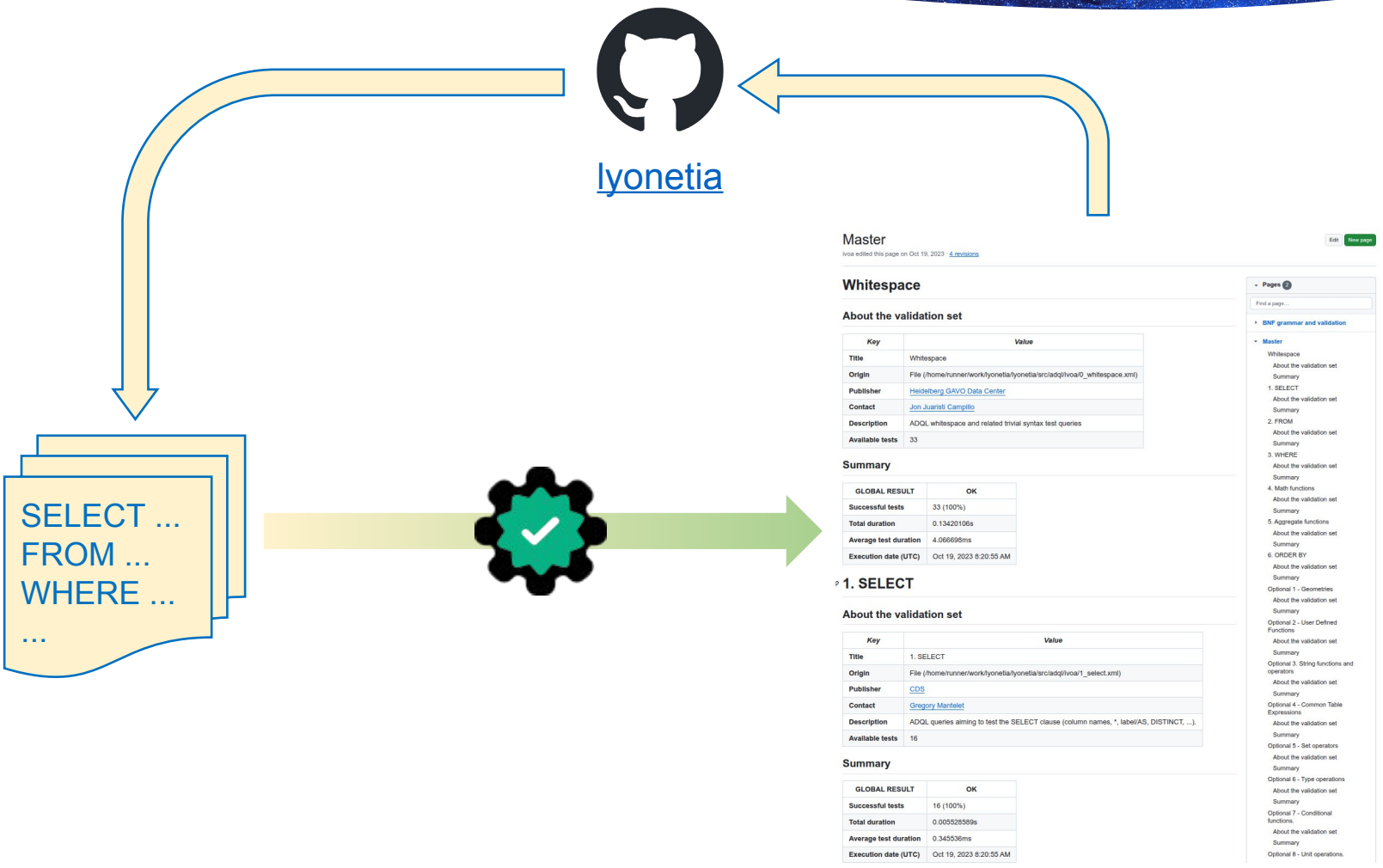


PEG is machine-readable

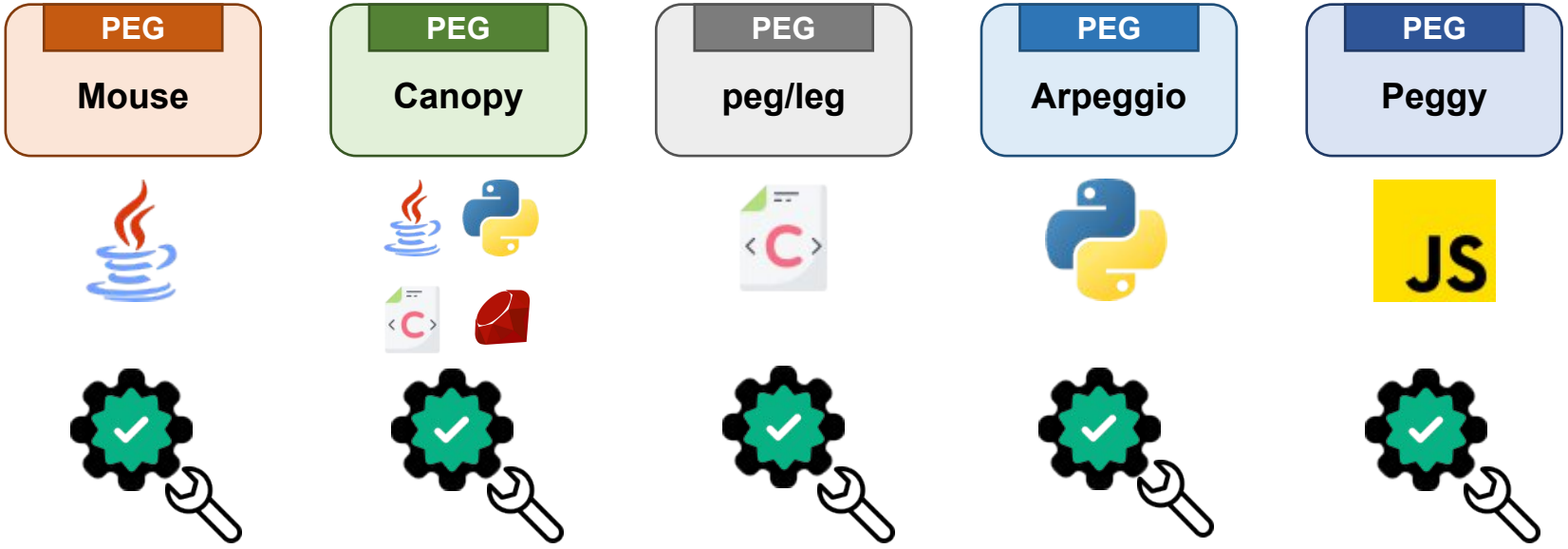



Building the
ADQL validator

It validates queries sets

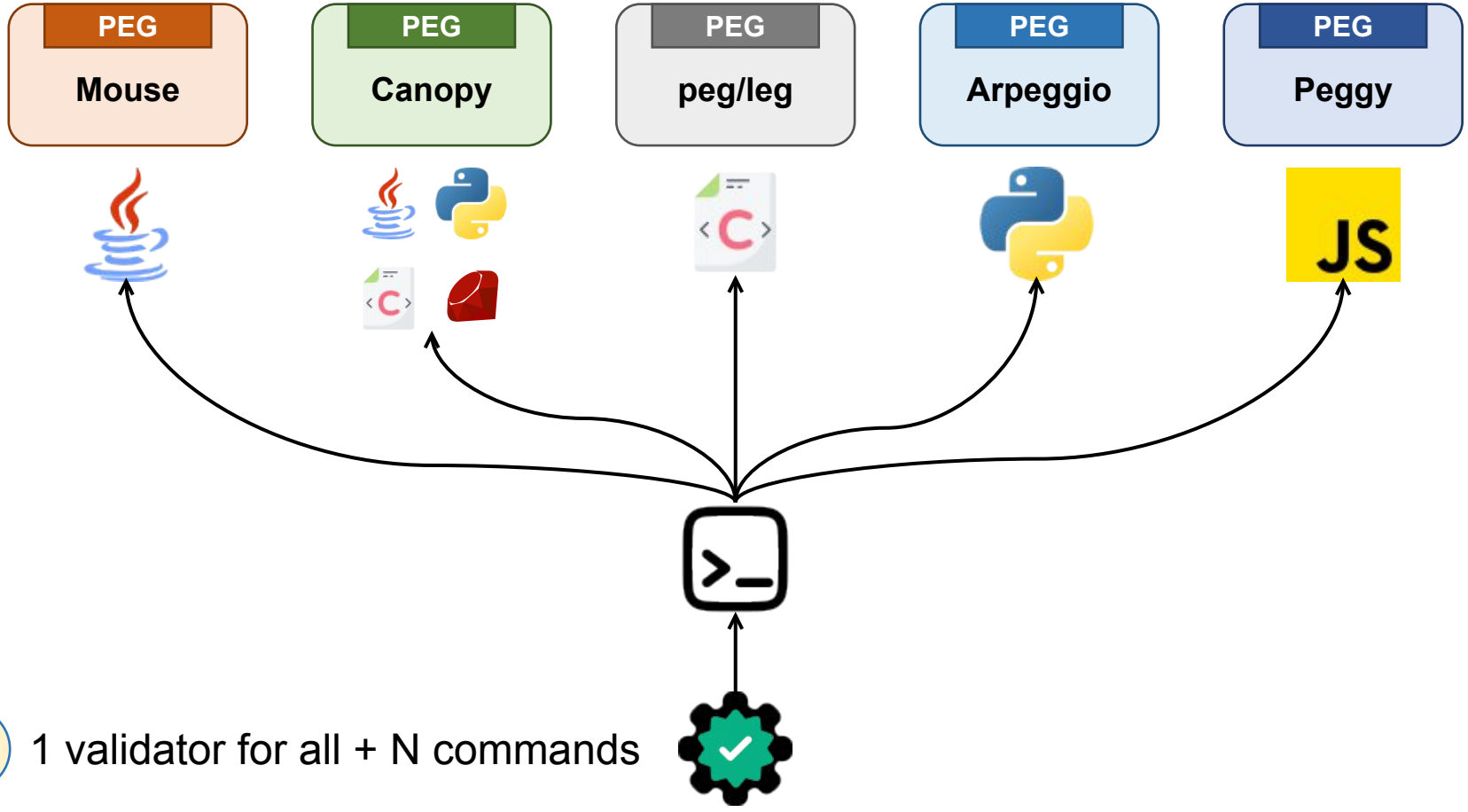


2 possible architectures



1 N validators (1 for each parser)

2 possible architectures



2 1 validator for all + N commands

A common API is required

Parameters

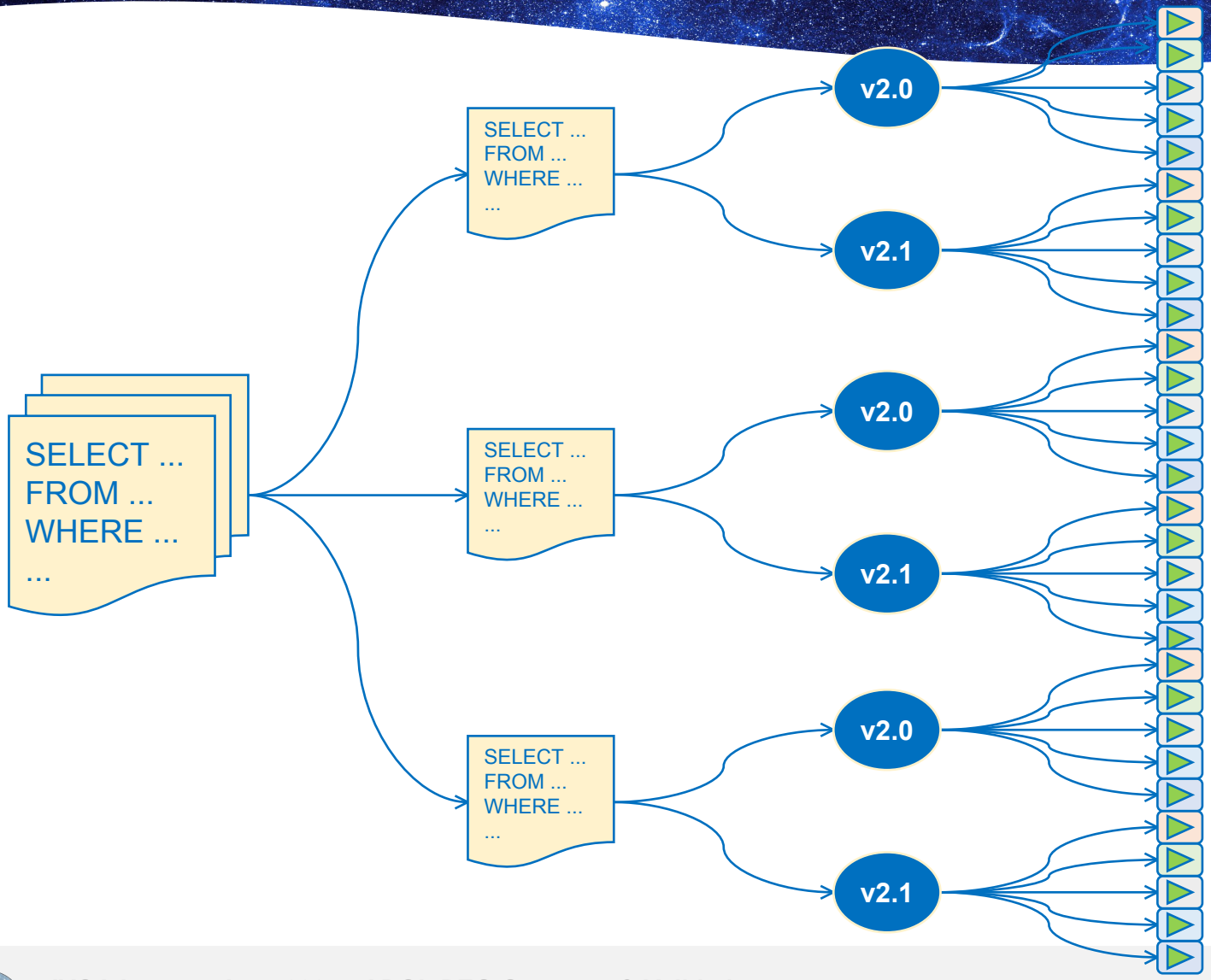
- ADQL version (optional)
- ADQL query

JSON output

```
{
  "success": "true"
}
```

```
{
  "success": "false",
  "error"  : "sorry!"
}
```

It should be multi-threaded




Let’s wrap up!

What’s next?


Del
ScrLk
Shift
PgUp
PgDn
End
thank you