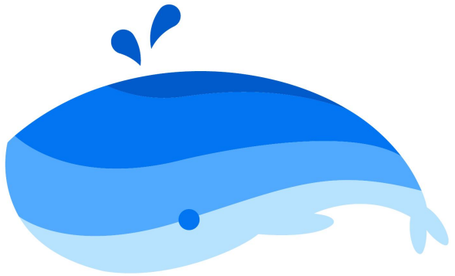
Implement SQL parser and type system in Databend

Databend Cloud is an affordable cloud **data warehouse** developed in Rust. In this exploration, we will dive into Databend's internal, specifically examining its SQL parser and type system that are meticulously crafted for Rust.



by **@AndyLok**

Who am I

骆迪安

https://github.com/andylokandy
https://twitter.com/AndylokandyLok

Databend Planner Team
Distributed transaction

Rust contributor
Idris-lang contributor

Databend

https://databend.rs

Feature-Rich

Instant Elasticity

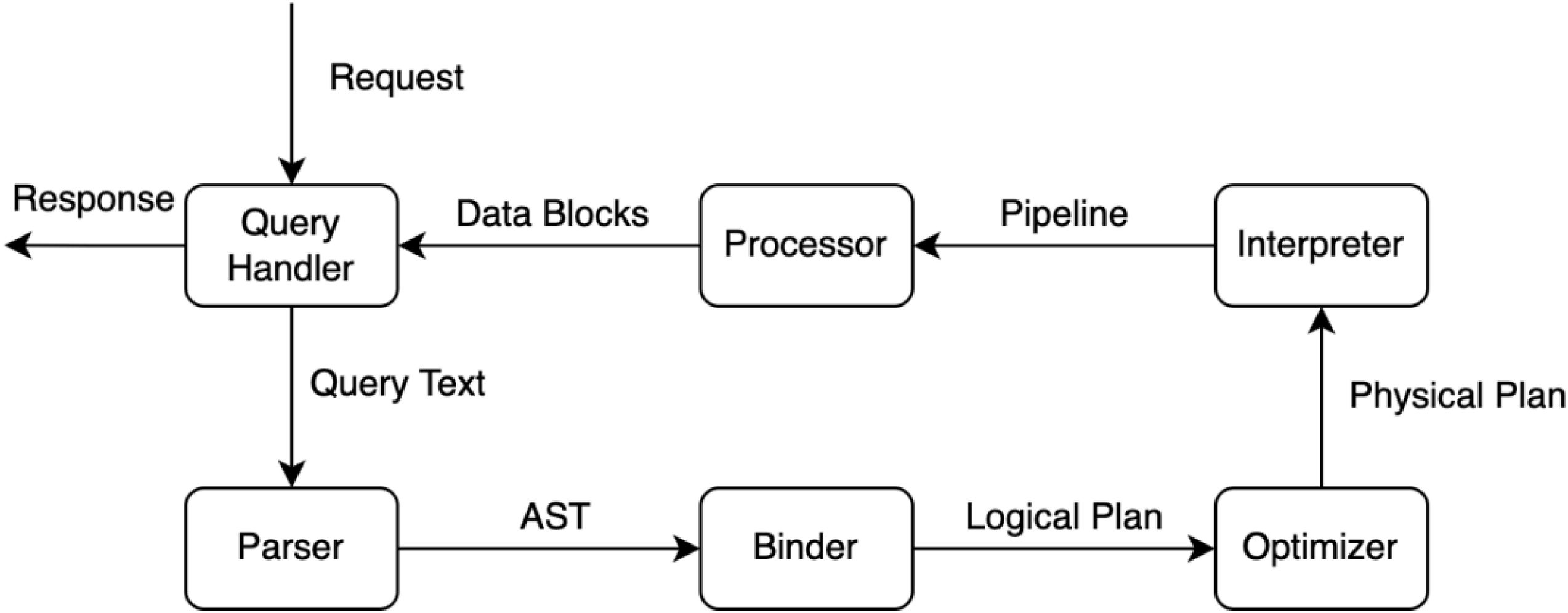
Support for Semi-Structured Data

MySQL/ClickHouse Compatible

Low Cost

Easy To Use

Cloud Native (S3, Azure Blob, Google Cloud Storage, Alibaba Cloud OSS, etc)


Request
Response
Query Handler
Data Blocks
Processor
Pipeline
Interpreter
Query Text
Physical Plan
Parser
AST
Binder
Logical Plan
Optimizer

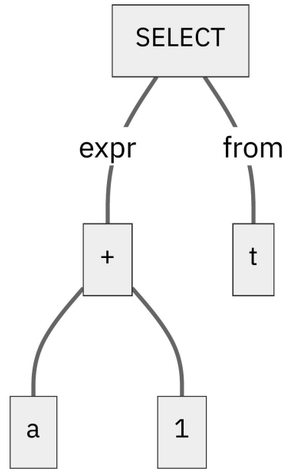
SQL Parser

Check for syntax violation and construct **Abstract Syntax Tree (AST)**

Input

```
SELECT a + 1 FROM t
```

Output



Tokenize

Convert **String** to **Token Stream**

Parse

Convert **Token Stream** to **AST**

Tokenize

Recognise **tokens** by regular expression

Regular expression for each token kind

```
Ident = [_a-zA-Z][_$a-zA-Z0-9]*
Number = [0-9]+
Plus = \+
SELECT = SELECT
FROM = FROM
```

Tokenize

Recognise **tokens** by regular expression

Input

```
SELECT a + 1 FROM t
```

Output

```
[
    Keyword(SELECT),
    Ident(a),
    BinOp(+),
    Number(1),
    Keyword(FROM),
    Ident(t),
]
```

Parse

Define **gramma rule** and construct **AST**

Backus–Naur form (BNF) grammars

```
<select_statement> ::= SELECT <expr> FROM <ident>
<expr> ::= <number>
         | <ident>
         | <expr> <op> <expr>
<op> ::= + | - | * | /
```

Parse

Define **gramma rule** and construct **AST**

Input

```
[
    Keyword(SELECT),
    Ident(a),
    BinOp(+),
    Number(1),
    Keyword(FROM),
    Ident(t),
]
```

Output

```
SelectStatement {
    projection: Expr::BinaryOp {
        op: Op::Plus,
        args: [
            Expr::ColumnRef("a"),
            Expr::Constant(Scalar::Int(1))
        ]
    }
    from: "t",
}
```

Choosing SQL Parser

sqlparser-rs

ANTLR4

LALRPOP

nom

Tokenizer

https://github.com/maciejhirsz/logos

```rust
#[derive(Logos)]
pub enum TokenKind {
    #[regex(r"[ \t\r\n\f]+", logos::skip)]
    Whitespace,

    #[regex(r#"[_a-zA-Z][_$a-zA-Z0-9]*"#)]
    Ident,
    #[regex(r"[0-9]+")]
    Number,

    #[token("+")]
    Plus,

    #[token("SELECT", ignore(ascii_case))]
    SELECT,
    #[token("FROM", ignore(ascii_case))]
    FROM,
}
```

Tokenizer

https://github.com/maciejhirsz/logos

Input

```
SELECT a + 1 FROM t
```

Output

```
[
    Token { kind: TokenKind::Select, text: "SELECT", span: 0..6 },
    Token { kind: TokenKind::Ident, text: "a", span: 7..8 },
    Token { kind: TokenKind::Plus, text: "+", span: 9..10 },
    Token { kind: TokenKind::Number, text: "1", span: 11..12 },
    Token { kind: TokenKind::From, text: "from", span: 13..17 },
    Token { kind: TokenKind::Ident, text: "t", span: 18..19 },
]
```

Span

```
SELECT  a     +     1       FROM    t
0..6    7..8  9..10 11..12  13..17  18..19
```

Error Report

Pretty print the error report thanks to the **span** information

```
error:
  --> SQL:1:19
   |
1  | create table a (c varch)
   | ------          - ^^^^^ expected `BOOLEAN`, `BOOL`, `UINT8`, `TINYINT`, `UINT16`, `SMALLINT`, or 33 more ...
   | |               |
   | |               while parsing `<column name> <type> [DEFAULT <default value>] [COMMENT '<comment>']`
   | while parsing `CREATE TABLE [IF NOT EXISTS] [<database>.]<table> [<source>] [<table_options>]`
```

Terminal

Recognize only one **token** from **token stream**

Combinator

Combine **terminals** and other small parsers into a larger parser

Terminal

Recognize only one **token** from **token stream**

Recongnize a token that has the exactly **text**

```
fn match_text(text: &str)
    -> impl FnMut(&[Token]) -> IResult<&[Token], Token>
{
    satisfy(|token: &Token| token.text == text)(i)
}
```

Recongnize a token that is of the **token kind**

```
fn match_token(kind: TokenKind)
    -> impl FnMut(&[Token]) -> IResult<&[Token], Token>
{
    satisfy(|token: &Token| token.kind == kind)(i)
}
```

Combinator

Combine **terminals** and other small parsers into a larger parser

tuple(a, b, c)

alt(a, b, c)

many0(a)

many1(a)

opt(a)