I built a tool I use

In Rust

The problem

- Writing SQL by hand is annoying
- Most of SQL can be inferred

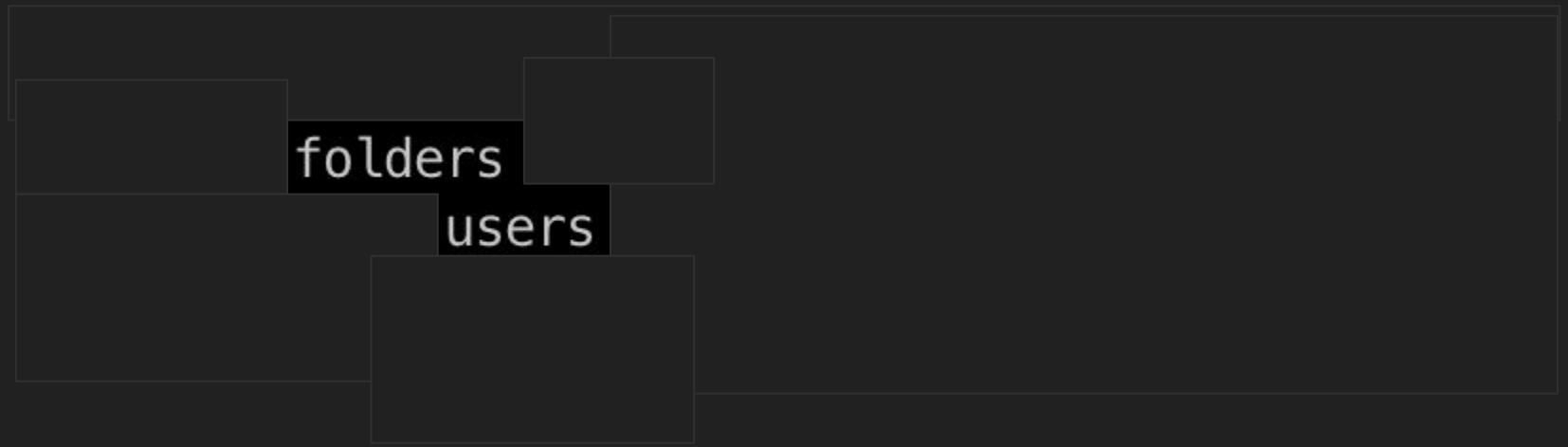
The problem

- Writing SQL by hand is annoying
- Most of SQL can be inferred
- I’m very lazy

TL;DR

```
1 SELECT folders.*
2 FROM folders
3 LEFT JOIN users ON users.id = folders.userId
4 LIMIT 10;
```

TL;DR



<insert demo here>

TL;DR

- Takes something like `users id = 3 | s: id name`
- Gives me results

- In vim
- Takes selection
- Passes to a script, script calls rust binary
- Scripts also calls DB, prints results
- Vim inserts results under the current line

Showcasing

- What it’s like to work on the code base
- Errors are awesome
- Errors are awesome
- Pest
- Adding it to vim

What it’s like to work on the codebase

- Fix compiler errors until it works
- Easy ¯_(ツ)_/¯

What it’s like to work on the codebase

```rust
#[derive(Debug)]
pub enum Operation<'a> {
    From(Node<TableName<'a>>),
    Join(Node<TableName<'a>>),
    ExplicitJoin(Node<TableName<'a>>, Node<ColumnName<'a>>),
    Select(Vec<Node<Operand<'a>>>),
    Unselect(Vec<Node<Operand<'a>>>),
    Filter(Vec<Node<Filter<'a>>>),
    // GroupBy(Vec<Node<Operand<'a>>>),
    Order(Vec<Node<Order<'a>>>),
    Limit(Node<Value<'a>>),
    Meta(MetaOperation),
}
```

What it’s like to work on the codebase

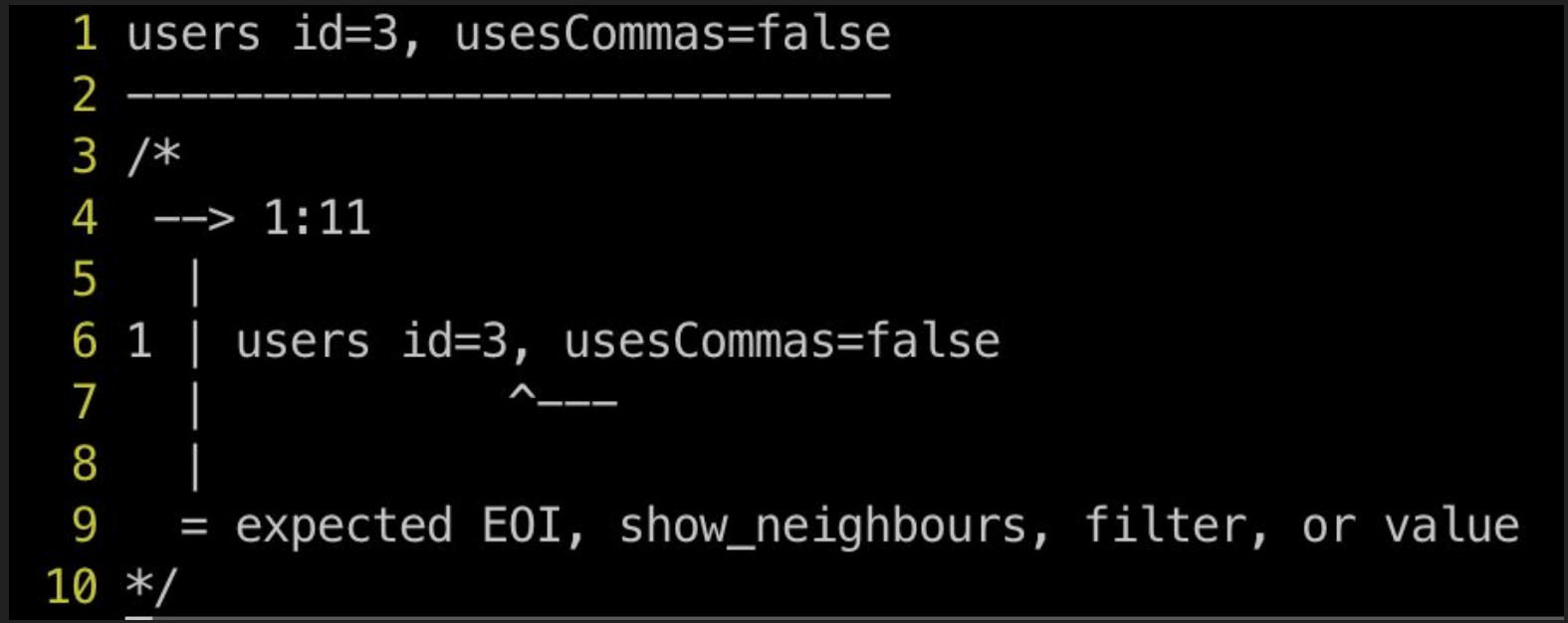
```
match operation_node.inner {
    AstOperation::From(ref table : &Node<TableName> ) => self.apply_from(table),
    AstOperation::Join(ref table : &Node<TableName> ) => self.apply_join(table),
    AstOperation::ExplicitJoin(ref table : &Node<TableName> , ref column : &Node<Colum
        self.apply_explicit_join( new_from_table: table, column)
    }
    AstOperation::Select(ref selections : &Vec<Node<Operand>> ) => self.apply_select:
    AstOperation::Unselect(ref selections : &Vec<Node<Operand>> ) => self.apply_unse
    AstOperation::Filter(ref filters : &Vec<Node<Filter>> ) => self.apply_filters(filt
    AstOperation::GroupBy(ref group_by) => self.apply_group_by( groups: group_by)?,
    AstOperation::Order(ref orders : &Vec<Node<Order>> ) => self.apply_orders(orders)
    AstOperation::Limit(ref limit : &Node<Value> ) => self.apply_limit( value: limit)?

    AstOperation::Meta(ref meta : &MetaOperation ) => self.apply_meta_operation(meta),
};
```

Showcasing

- ~~What it's like to work on the code base~~
- Errors are awesome
- Errors are awesome
- Pest
- Adding it to vim

Errors are awesome



Showcasing

- ~~What it's like to work on the code base~~
- ~~Errors are awesome~~
- Errors are awesome
- Pest
- Adding it to vim

Errors are awesome

- The code is full of ?

```
fn transpile(self, input: I) -> Result<O, PineError> {
    let pine = self.parser.parse(input.into())?;
    let query = self.builder.build(&pine)?;

    self.renderer.render(&query)
}
```

Errors are awesome

- ? does automatic conversion for your Result::Err

```rust
impl From<SyntaxError> for PineError {
    fn from(error: SyntaxError) -> PineError {
        let message : String = error.to_string();

        let cause: Box<dyn Error> = Box::new( x: error);
        let cause = Some(cause);

        PineError { message, cause }
    }
}
```

Errors are awesome

- Step 1: Implement `From<OtheErrors> for YourError`
- Step 2: Implement `Display for YourError`
- Step 3: `eprintln!("{}", your_error)`
- Step 4: no more cognitive load!

Showcasing

- ~~What it's like to work on the code base~~
- ~~Errors are awesome~~
- ~~Errors are awesome~~
- Pest
- Adding it to vim

Pest

- “pest. The Elegant Parser” - pest.rs
- Build your own language

```
operator = _{ optr_eq | optr_ne | optr_gte | optr_gt | optr_lte | optr_lt }
optr_eq  = { "="  }
optr_ne  = { "!=" }
optr_gt  = { ">"  }
optr_gte = { ">=" }
optr_lt  = { "<"  }
optr_lte = { "<=" }
```

Pest

- This is what you get

```
Pair {
    rule: pine,
    span: Span {
        str: "users | select: id",
        start: 0,
        end: 18,
    },
    inner: [
        Pair {
            rule: simple_compound_expression,
            span: Span {
                str: "users ",
                start: 0,
                end: 6,
            },
            inner: [
                Pair {
                    rule: table_name,
                    span: Span {
                        str: "users",
                        start: 0,
                        end: 5,
                    },
```

Pest

- pest::error::Error implements Display!

Pest

- pest::error::Error implements Display!

Showcasing

- ~~What it's like to work on the code base~~
- ~~Errors are awesome~~
- ~~Errors are awesome~~
- ~~Pest~~
- Adding it to vim

Adding it to vim

- Because you won’t use it unless it’s ergonomic

Adding it to vim

Keybinding config:

```
:vmap <C-P><C-P> :'<,'>! pipes-to-query2 <CR><CR><Esc>

:vmap <C-L><C-L> :'<,'>! pipes-to-query2 '\G'<CR><CR><Esc>
```

Adding it to vim

```
#!/usr/bin/env bash

input="$(cat)"

query="$(rusty-pine penneo "$input")"


result=$(echo "$query" | mysql -h <your db host>)

echo "${result}"
```

Showcasing

- ~~What it's like to work on the code base~~
- ~~Errors are awesome~~
- ~~Errors are awesome~~
- ~~Pest~~
- ~~Adding it to vim~~

About

The original pine:

- http://pine-lang.org

Rusty pine:

- https://github.com/fabianbadoi/rusty-pine/

Me:

- https://www.linkedin.com/in/fabian-badoi-98588149/
- https://github.com/fabianbadoi/