Type System Tricks for the Real World

```rust
fn main() {
    break rust;
    // Can you read this color scheme?
}
```

Who am I?

- Sean Griffin
- 10x Hacker Ninja Guru at Shopify
- Rails Committer
- Maintainer of Active Record
- Creator of Diesel
- Bikeshed co-host

Programming is full of trade-offs

Haskell

- Pros
  - Expressive type system
- Cons
  - No control over data layout

- Pros
  - Tight control over data layout
- Cons
  - "type" system


Why don’t we have both?

Rust's Fireflower

Rust's Fireflower

Rust's Fireflower

```rust
pub struct HashSet<K> {
    map: HashMap<K, ()>
}

impl<K: Hash + Eq> HashSet<K> {
    pub fn insert(&mut self, value: K) {
        self.map.insert(value, ());
    }

    pub fn contains(&self, value: &K) -> bool {
        self.map.contains_key(value)
    }
}
```

```ruby
class Set
  def initialize
    @hash = {}
  end

  def insert(value)
    @hash[value] = true
  end

  def contains(value)
    @hash.key?(value)
  end
end
```

char

```
struct Foo {
    bar: char,
}
```

```
struct Foo {
    bar: char,
    baz: char,
}
```

```
enum Foo {
    Bar(char),
}
```

```
enum Foo {
    Bar(char),
    Baz,
}
```

```
struct Baz;
```

```
enum Foo {
    Bar(char),
    Baz,
}
```

```
enum Foo {
    Bar(char),
    Baz(u64),
}
```

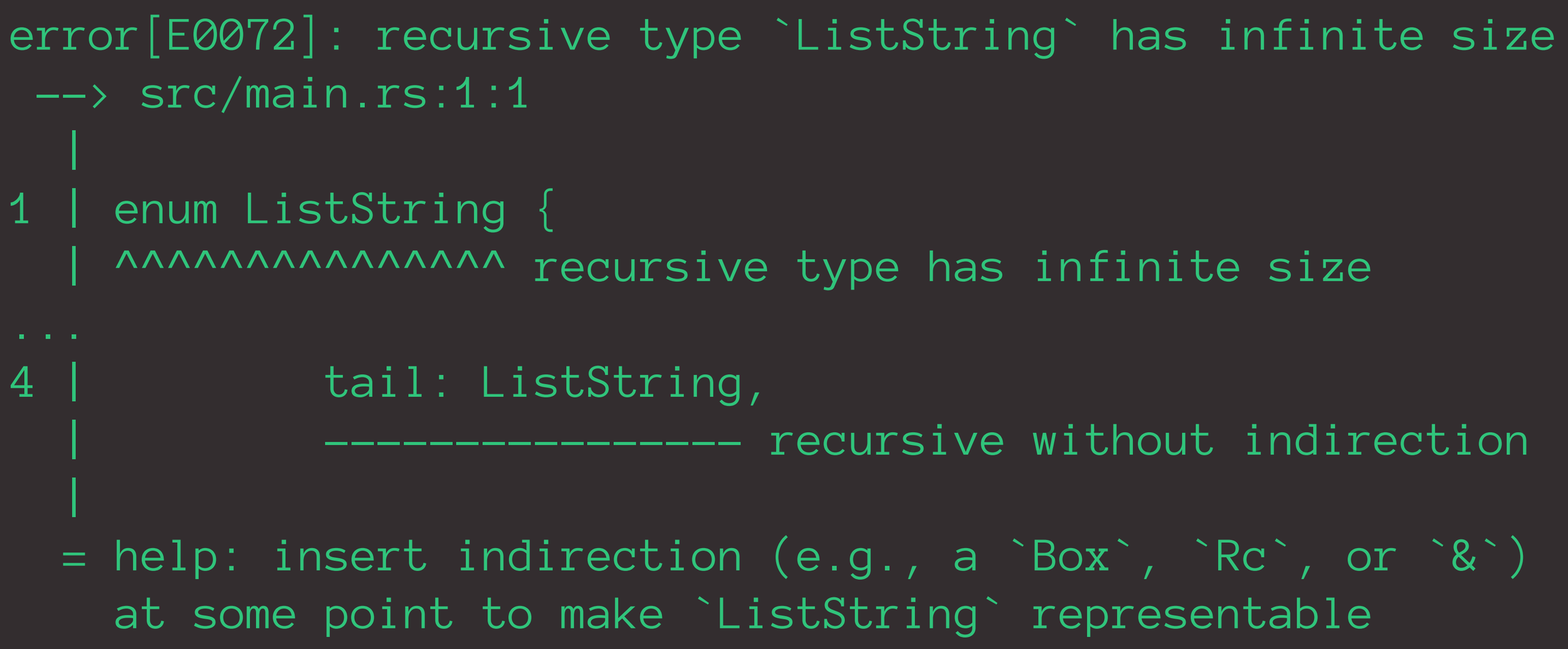
```
enum ListString {
    Cons {
        head: char,
        tail: ListString,
    },
    Nil,
}
```

```
enum ListString {
    Cons {
        head: char,
        tail: ListString,
    },
    Nil,
}
```

```
enum ListString {
    Cons {
        head: char,
        tail: ListString,
    },
    Nil,
}
```

```
enum ListString {
    Cons {
        head: char,
        tail: ListString,
    },
    Nil,
}
```

```
enum ListString {
    Cons {
        head: char,
        tail: ListString,
    },
    Nil,
}
```


error[E0072]: recursive type `ListString` has infinite size
--> src/main.rs:1:1
|
1 | enum ListString {
| ^^^^^^^^^^^^^^^^ recursive type has infinite size
...
4 | tail: ListString,
| ---------------- recursive without indirection
|
= help: insert indirection (e.g., a `Box`, `Rc`, or `&`)
at some point to make `ListString` representable

```
enum ListString {
    Cons {
        head: char,
        tail: ListString,
    },
    Nil,
}
```

```
enum ListString {
    Cons {
        head: char,
        tail: Box<ListString>,
    },
    Nil,
}
```

```rust
pub struct Cons<Tail> {
    head: char,
    tail: Tail,
}

pub struct Nil;
```

```
struct Pizza<Topping> {
    topping: Topping
}
```

```
struct Pinapple;

struct Pizza<Topping> {
    topping: Topping
}

size_of::<Pizza<Pinapple>>()
```

```
struct Pinapple;

struct Pizza<Topping> {
    topping: Topping
}

size_of::<Pizza<Pinapple>>() // => Error:
// Pineapple doesn't go on pizza, Steve. Fite me.
```

```
struct Pizza<Topping> {
    topping: Topping
}

size_of::<Pizza<u8>>()
```

```rust
pub struct Cons<Tail> {
    head: char,
    tail: Tail,
}

pub struct Nil;
```

Nil

```
Cons<Nil>
```

```
Cons<Cons<Nil>>
```

```
Cons<Cons<Cons<Nil>>
>
```

```rust
pub struct Cons<Tail> {
    head: char,
    tail: Tail,
}

pub struct Nil;
```

This isn't without tradeoffs

```
fn do_stuff_with_string(s: ListString) {
    // doing stuff!
}
```

```
fn do_stuff_with_string<T: ListString>(s: T) {
    // doing stuff!
}
```

```
fn string_contains_a(s: ListString) -> bool {
    match s {
        Cons { head: 'a', .. } => true,
        Cons { tail, .. } => string_contains_a(tail),
        Nil => false,
    }
}
```

```
fn string_contains_a<T: ListString>(s: T) -> bool {
    match s.unpack() {
        Some(('a', _)) => true,
        Some((_, tail)) => string_contains_a(tail),
        None => false,
    }
}
```

```rust
trait Robot {
    fn username(&self) -> &str;
}

fn say_hi<T: Robot>(robot: &T) {
    println!("Hi, {}", robot.username());
}
```

```
struct Bors;

impl Robot for Bors {
    fn username(&self) -> &str {
        "@bors"
    }
}
```

```
struct Bors;

impl Robot for Bors {
    fn username(&self) -> &str {
        "@bors"
    }
}

struct Alex;

impl Robot for Alex {
    fn username(&self) -> &str {
        "@alexcrichton"
    }
}
```

```
struct Bors;

impl Robot for Bors {
    fn username(&self) -> &str {
        "@bors"
    }
}

struct Alex;

impl Robot for Alex {
    fn username(&self) -> &str {
        "@alexcrichton"
    }
}
```

```rust
struct Bors;

fn Robot_username_Bors(_: &Bors) -> &str {
    "@bors"
}

struct Alex;

impl Robot for Alex {
    fn username(&self) -> &str {
        "@alexcrichton"
    }
}
```

```
struct Bors;

fn Robot_username_Bors(_: &Bors) -> &str {
    "@bors"
}

struct Alex;

impl Robot for Alex {
    fn username(&self) -> &str {
        "@alexcrichton"
    }
}
```

```rust
struct Bors;

fn Robot_username_Bors(_: &Bors) -> &str {
    "@bors"
}

struct Alex;

fn Robot_username_Alex(_: &Alex) -> &str {
    "@alexcrichton"
}
```

```rust
fn say_hi<T: Robot>(robot: &T) {
    println!("Hi, {}", robot.username());
}
```

```
fn say_hi_Bors(bors: &Bors) {
    println!("Hi, {}", Robot_username_Bors(bors));
}

fn say_hi_Alex(alex: &Alex) {
    println!("Hi, {}", Robot_username_Alex(alex));
}
```

```rust
fn say_hi_Bors(bors: &Bors) {
    println!("Hi, {}", Robot_username_Bors(bors));
}

fn say_hi_Alex(alex: &Alex) {
    println!("Hi, {}", Robot_username_Alex(alex));
}
```

```rust
fn say_hi_Bors(_: &Bors) {
    println!("Hi, {}", "@bors");
}

fn say_hi_Alex(alex: &Alex) {
    println!("Hi, {}", Robot_username_Alex(alex));
}
```

```rust
fn say_hi_Bors(_: &Bors) {
    println!("Hi, {}", "@bors");
}

fn say_hi_Alex(alex: &Alex) {
    println!("Hi, {}", Robot_username_Alex(alex));
}
```

```
fn say_hi_Bors(_: &Bors) {
    println!("Hi, {}", "@bors");
}

fn say_hi_Alex(_: &Alex) {
    println!("Hi, {}", "@alexcrichton");
}
```

```
fn say_hi(robot: &Robot) {
    println!("Hi, {}", robot.username());
}
```


Brown's
by Handi-Craft

A real world example

```rust
#[test]
fn complex_queries_with_no_data_have_no_size() {
    assert_eq!(0, mem::size_of_val(&users.as_query()));
    assert_eq!(0, mem::size_of_val(&users.select(id).as_query()));
    assert_eq!(0, mem::size_of_val(
        &users.inner_join(posts).filter(name.eq(title))
    ));
}
```

List of things you can do with a 0-size type

-

```
users.find(1)
```

```
SelectStatement<
    users::table,
    DefaultSelectClause,
    NoDistinctClause,
    WhereClause<
        Eq<
            users::id,
            Bound<Integer, i32>,
        >,
    >,
    NoOrderClause,
    NoLimitClause,
    NoOffsetClause,
    NoGroupByClause,
>
```

```
SelectStatement<
    users::table,
    DefaultSelectClause,
    NoDistinctClause,
    WhereClause<
        Eq<
            users::id,
            Bound<Integer, i32>,
        >,
    >,
>
```

```
SelectStatement {
    select: DefaultSelectClause,
    from: users::table,
    distinct: NoDistinctClause,
    where_clause: WhereClause(Eq {
        left: users::id,
        right: Bound::new(1),
    }),
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.distinct.walk_ast(out)?;
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.distinct.walk_ast(out)?;
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.distinct.walk_ast(out)?;
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    NoDistinctClause::walk_ast(out)?;
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    Ok(())?;
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    DefaultSelectClause::walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.from.default_selection()
        .walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.from.default_selection()
        .walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    (users::id, users::name)
        .walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    (users::id, users::name)
        .walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    if 0 != 0 {
        out.push_sql(", ");
    }
    users::id.walk_ast(out)?;
    if 1 != 0 {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
```

```
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    if 0 != 0 {
        out.push_sql(", ");
    }
    users::id.walk_ast(out)?;
    if 1 != 0 {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    if false {
        out.push_sql(", ");
    }
    users::id.walk_ast(out)?;
    if 1 != 0 {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    if false {
        out.push_sql(", ");
    }
    users::id.walk_ast(out)?;
    if 1 != 0 {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::id.walk_ast(out)?;
    if 1 != 0 {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::id.walk_ast(out)?;
    if 1 != 0 {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::id.walk_ast(out)?;
    if true {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::id.walk_ast(out)?;
    if true {
        out.push_sql(", ");
    }
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::id.walk_ast(out)?;
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::id.walk_ast(out)?;
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::table.walk_ast(out)?;
    out.push_sql(".");
    out.push_identifier("id");
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    users::table.walk_ast(out)?;
    out.push_sql(".");
    out.push_identifier("id");
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    out.push_identifier("users");
    out.push_sql(".");
    out.push_identifier("id");
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql(&"users".replace('"', "\"\""));
    out.push_sql("\"");
    out.push_sql(".");
    out.push_identifier("id");
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql(&"users".replace('"', "\"\""));
    out.push_sql("\"");
    out.push_sql(".");
    out.push_identifier("id");
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_identifier("id");
    out.push_sql(", ");
    users::name.walk_ast(out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
```

```
out.push_sql("SELECT ");
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_identifier("id");
out.push_sql(", ");
users::name.walk_ast(out)?;
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
Ok(())
```

```
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_sql("\"");
out.push_sql(&"id".replace('"', "\"\""));
out.push_sql("\"");
out.push_sql(", ");
users::name.walk_ast(out)?;
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
```

```
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_sql("\"");
out.push_sql(&"id".replace('"', "\"\""));
out.push_sql("\"");
out.push_sql(", ");
users::name.walk_ast(out)?;
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
```

```
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_sql("\"");
out.push_sql("id");
out.push_sql("\"");
out.push_sql(", ");
users::name.walk_ast(out)?;
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
```

```
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        users::name.walk_ast(out)?;
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
```

```
        out.push_sql("\"");
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        users::name.walk_ast(out)?;
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("\"");
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        users::name.walk_ast(out)?;
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        users::name.walk_ast(out)?;
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        users::table.walk_ast(out)?;
        out.push_sql(".");
        out.push_identifier("name");
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        users::table.walk_ast(out)?;
        out.push_sql(".");
        out.push_identifier("name");
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
out.push_sql("id");
out.push_sql("\"");
out.push_sql(", ");
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_identifier("name");
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
Ok(())
```

```
out.push_sql("id");
out.push_sql("\"");
out.push_sql(", ");
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_identifier("name");
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
Ok(())
```

```
out.push_sql("id");
out.push_sql("\"");
out.push_sql(", ");
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_sql("\"");
out.push_sql("name");
out.push_sql("\"");
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
```

```
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(", ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
```

```
out.push_sql(", ");
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_sql("\"");
out.push_sql("name");
out.push_sql("\"");
out.push_sql(" FROM ");
self.from.walk_ast(out)?;
self.where_clause.walk_ast(out)?;
Ok(())
```

```
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        self.from.walk_ast(out)?;
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        self.where_clause.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(" WHERE ");
        self.where_clause.0.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(" WHERE ");
        self.where_clause.0.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(" WHERE ");
        self.where_clause.0.left.walk_ast(out)?;
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("name");
        out.push_sql("\"");
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(" WHERE ");
        self.where_clause.0.left.walk_ast(out)?;
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" FROM ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(" WHERE ");
        self.where_clause.0.left.walk_ast(out)?;
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(" WHERE ");
        self.where_clause.0.left.walk_ast(out)?;
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        self.where_clause.0.left.walk_ast(out)?;
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        users::table.walk_ast(out)?;
        out.push_sql(".");
        out.push_identifier("id");
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        users::table.walk_ast(out)?;
        out.push_sql(".");
        out.push_identifier("id");
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_identifier("id");
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_identifier("id");
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
        out.push_sql(" WHERE ");
        out.push_sql("\"");
        out.push_sql("users");
        out.push_sql("\"");
        out.push_sql(".");
        out.push_sql("\"");
        out.push_sql("id");
        out.push_sql("\"");
        out.push_sql(" = ");
        self.where_clause.0.right.walk_ast(out)?;
        Ok(())
    }
```

```
out.push_sql(" WHERE ");
out.push_sql("\"");
out.push_sql("users");
out.push_sql("\"");
out.push_sql(".");
out.push_sql("\"");
out.push_sql("id");
out.push_sql("\"");
out.push_sql(" = ");
out.push_bind_param(
    &self.where_clause.0.right.item)?;
Ok(())
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    self.distinct.walk_ast(out)?;
    self.select.walk_ast(&self.from, out)?;
    out.push_sql(" FROM ");
    self.from.walk_ast(out)?;
    self.where_clause.walk_ast(out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
    out.push_sql(", ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("name");
    out.push_sql("\"");
    out.push_sql(" FROM ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(" WHERE ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
    out.push_sql(" = ");
    out.push_bind_param(&self.where_clause.0.right.item, out)?;
    Ok(())
}
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    out.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    if let AstPass::ToSql(ref mut builder) = out {
        builder.push_sql("SELECT ");
    }
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    let AstPass::ToSql(ref mut builder) = out;
    builder.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    let AstPass::ToSql(ref mut builder) = out;
    builder.push_sql("SELECT ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
```

```rust
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    let AstPass::ToSql(ref mut builder) = out;
    builder.sql.push_str("SELECT ");
    out.push_sql("\"");
    out.push_sql("users");
    out.push_sql("\"");
    out.push_sql(".");
    out.push_sql("\"");
    out.push_sql("id");
    out.push_sql("\"");
```

```
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    let AstPass::CollectBinds { collector, metadata_lookup } = out;
    collector.push_bound_value(
        &self.where_clause.0.right.value,
        metadata_lookup,
    )?;
    Ok(())
}
```

```
fn walk_ast(&self, out: AstPass) -> QueryResult<()> {
    Ok(())
}
```

These optimizations are what let us build zero cost abstractions

```rust
pub struct HashSet<K> {
    map: HashMap<K, ()>
}

impl<K: Hash + Eq> HashSet<K> {
    pub fn insert(&mut self, value: K) {
        self.map.insert(value, ());
    }

    pub fn contains(&self, value: &K) -> bool {
        self.map.contains_key(value)
    }
}
```

```rust
pub struct HashSet<K> {
    map: HashMap<K, ()>
}

impl<K: Hash + Eq> HashSet<K> {
    pub fn insert(&mut self, value: K) {
        self.map.insert(value, ());
    }

    pub fn contains(&self, value: &K) -> bool {
        self.map.contains_key(value)
    }
}
```

shopify

Please ask me questions now

Contact

- twitter: @sgrif
- github: @sgrif
- podcast: bikeshed.fm