Diesel

A safe and extensible ORM and Query Builder for Rust

Who am I?

- Sean Griffin
- Rails Committer
- Maintainer of Active Record
- Creator of Diesel
- Bikeshed co-host

Why Diesel?

```
let stmt = try!(conn.prepare("SELECT * FROM users \
                              WHERE users.name = $1 LIMIT 1"));
let rows = try!(stmt.query(&[&name]));
let row = rows.iter().next();
Ok(User::from_row(&row))
```

```
users.filter(name.eq(&name))
    .first(&connection)
```

```
#[derive(Queryable)]
```

```
SELECT * FROM users WHERE users.name = $1
```

```sql
SELECT * FROM users WHERE usres.nme = $3
```

thread '<main>' panicked at 'Sean you are a fucking moron. Give up on life'

— What I see when I get database errors

I hate runtime errors

I hate runtime errors (Especially in Rust)

```
thread '<main>' panicked at 'called `Result::unwrap()`
on an `Err` value: "LOL STACKTRACES"',
../src/libcore/result.rs:741
```

Abstractibility

```rust
let versions = Version::belonging_to(krate)
    .select(id)
    .order(num.desc())
    .limit(5);
let downloads = try!(version_downloads
    .filter(date.gt(now - 90.days()))
    .filter(version_id.eq(any(versions)))
    .order(date)
    .load::<Download>(&conn));
```

```rust
let versions = krate.latest_versions()
    .limit(5);
let downloads = try!(version_downloads
    .filter(date.gt(now - 90.days()))
    .filter(version_id.eq(any(versions)))
    .order(date)
    .load::<Download>(&conn));
```

```rust
let mut versions = try!(krate.versions(tx));
versions.sort_by(|a, b| b.num.cmp(&a.num));
let to_show = &versions[..cmp::min(5, versions.len())];
let ids = to_show.iter().map(|i| i.id).collect::<Vec<_>>();
let versions = krate.latest_versions()
    .limit(5);
let downloads = try!(version_downloads
    .filter(date.gt(now - 90.days()))
    .filter(version_id.eq(any(versions)))
    .order(date)
    .load::<Download>(&conn));
```

```rust
let versions = krate.latest_versions()
    .limit(5);
let downloads = try!(version_downloads
    .filter(Download::recent())
    .filter(version_id.eq(any(versions)))
    .order(date)
    .load::<Download>(&conn));
```

Abstractions are cool

I want to think about my problem domain

How does it all work?

```
users.filter(name.eq("Sean"))
```

Macros and codegen

```
table! {
    users {
        id -> Serial,
        name -> String,
        hair_color -> Nullable<String>,
    }
}
```

```
infer_schema!(env!("DATABASE_URL"))
```

```rust
pub mod users {
    pub struct table;
    pub struct id;
    pub struct name;
    pub struct hair_color;

    /* Boilerplate impls for all of the things */
}
```

```rust
impl Expression for users::name {
    type SqlType = VarChar;
}
```

```rust
impl SelectableExpression<users::table> for users::name {
}
```

```
fn eq<T>(other: T) -> Eq<Self, T::Expression> where
    T: AsExpression<Self::SqlType>,
name.eq("Sean");
name.eq("Sean");
name.eq(String::from("Sean"));
name.eq("Sean");
name.eq(String::from("Sean"));
name.eq(hair_color);
name.eq("Sean");
name.eq(String::from("Sean"));
name.eq(hair_color);
name.eq(lower(coalesce(hair_color, "Jim")));
```

```rust
impl<'a> AsExpression<VarChar> for &'a str {
    type Expression = Bound<&'a str, VarChar>;

    fn as_expression(self) -> Self::Expression {
        // ...
    }
}
```

Things weren't always the cleanest

```rust
fn find<T, U, PK>(&self, source: T, id: PK) -> QueryResult<U> where
    T: Table + FilterDsl<FindPredicate<T, PK>>,
    FindBy<T, T::PrimaryKey, PK>: LimitDsl,
    Limit<FindBy<T, T::PrimaryKey, PK>>: QueryFragment<Self::Backend>,
    U: Queryable<<Limit<FindBy<T, T::PrimaryKey, PK>> as Query>::SqlType, Self::Backend>,
    Self::Backend: HasSqlType<<Limit<FindBy<T, T::PrimaryKey, PK>> as Query>::SqlType>,
    PK: AsExpression<PkType<T>>,
    AsExpr<PK, T::PrimaryKey>: NonAggregate,
```

(Don't worry, that where clause is gone now. It was horrible)

```
pub trait FindDsl<PK> where
    Self: Table + FilterDsl<Eq<<Self as Table>::PrimaryKey, PK>>,
    PK: AsExpression<<Self::PrimaryKey as Expression>::SqlType>,
    Eq<Self::PrimaryKey, PK>: SelectableExpression<Self, SqlType=Bool> + NonAggregate,
```

The end result is safety

```rust
fn main() {
    let _ = users::table.filter(posts::id.eq(1));
    //~^ ERROR SelectableExpression
}
```

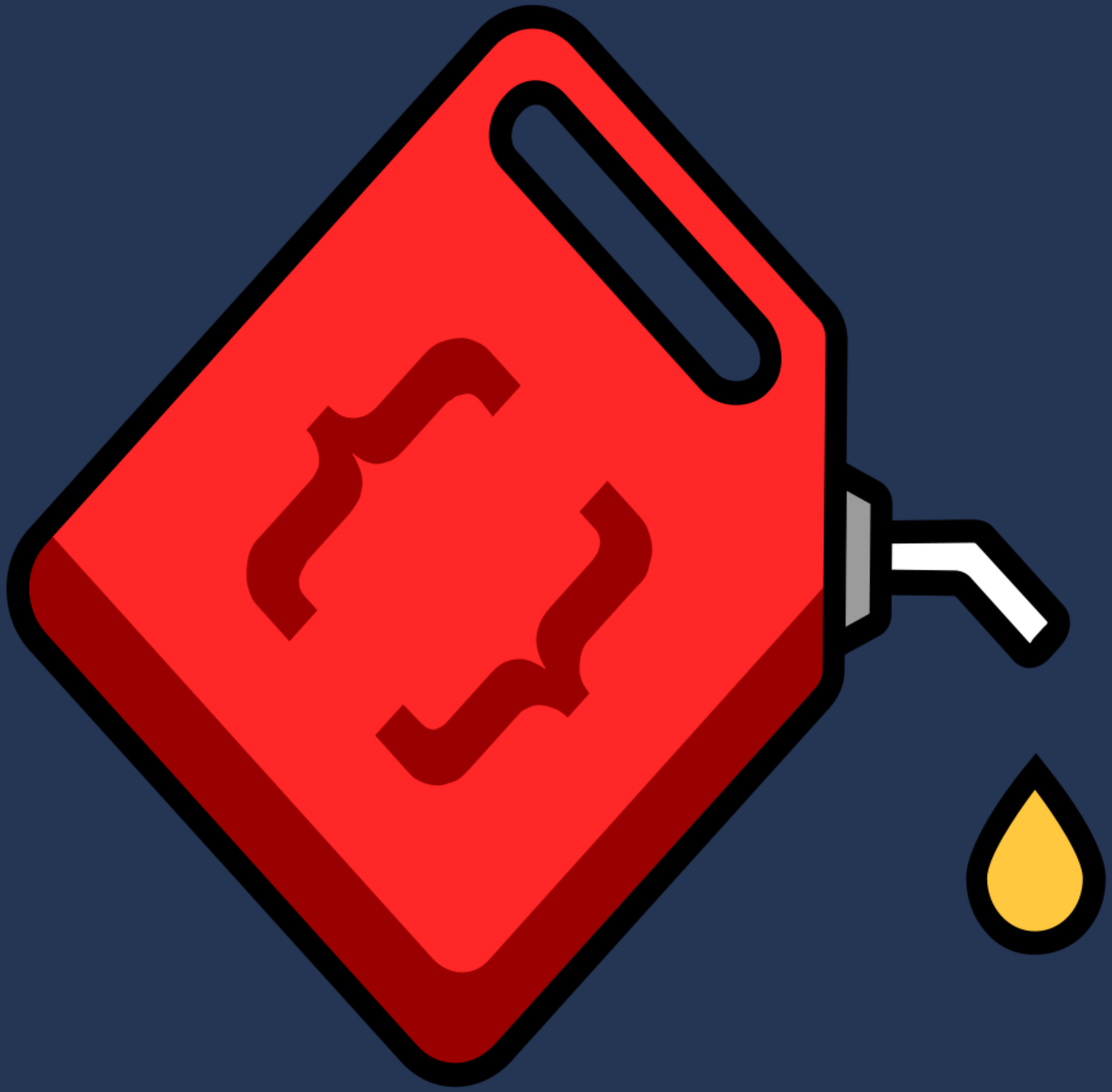
```rust
fn main() {
    use self::users::dsl::*;

    let pred = id.eq("string");
    //~^ ERROR E0277
    let pred = id.eq(name);
    //~^ ERROR type mismatch
}
```

Safety allows speed

Types are fucking cool

http://diesel.rs

I have stickers!



http://diesel.rs