

# tbd.rs - an experiment with database interfaces

or: how to use zero-sized types for great effect

Florian Gilcher

CEO and Rust Trainer  
Ferrous Systems GmbH  
Chairperson of a Ruby non-profit

Oct 29, 2018

# WAT?

- ▶ <https://github.com/berlinrs/tbd.rs>
- ▶ A different database interaction library
- ▶ Inspired by Rom.rb, Ecto, LINQ

- ▶ A more explicit (and controllable) mapping to databases
- ▶ NoSQL capabilities
- ▶ Flexible usage to database-specific features

- ▶ All Futures-based
- ▶ `async-await` compatible

## Notice

This is a sketch!

## What does it look like?

```
let gateway = Sqlite3Gateway { connection: RefCell::new(Some(conn)) };
let repos = BlogRepository { gateway: gateway };

let query = select::<Post>().from::<Posts>();

let e1 = query.execute(&repos).for_each(|post| {
    println!("{:?}", post);
    future::ready(())
});

await!(e1);
```

## A model definition

```
struct Post {  
    id: u64,  
    content: String  
}
```

```
struct Comment {  
    id: u64,  
    post_id: u64,  
    content: String  
}
```

How can we teach Rust about the structure of storage?



# Relation

```
pub struct Posts;

impl Relation for Posts {
    type PrimaryKey = i64;
    type Model = Post;

    // ...

    fn name() -> &'static str {
        "posts"
    }
}

pub struct Comments;
// ..
```

Problem: we know that these exist, but in which groups?

# Repository

```
pub trait Repository {  
    type Gateway: Gateway;  
  
    fn gateway(&self) -> &Self::Gateway;  
}  
  
pub trait Stores<Rel> : Repository {  
  
}
```

This is very abstract, let's make this concrete!

```
struct BlogRepository;  
  
impl Stores<Posts> for BlogRepository;  
impl Stores<Comments> for BlogRepository;
```

What does that give us? - Query expressions!

# Select

```
pub fn select<M>() -> Select<M> {  
    //...  
}  
  
impl<M> Select<M> {  
    pub fn from<R>(self) -> SelectFrom<R>  
        where R: Relation<Model = M> {  
        //...  
    }  
}
```

## Query Execution

```
pub trait Execute<Repos, R, ReturnType>
  where R: Relation,
        Repos: Repository,
        Self: Query<ReturnType=ReturnType> {

  type FutureType;

  fn execute(&self, repos: &Repos) -> Self::FutureType
    where Repos: Stores<R>;
}
```



This also allows use to constrain queries to the relations in a repository.

The repository and relations are not coupled to the query language, others can be used.

Storage: mapping Repositories to the storage Gateway. (I have that built for Sqlite3)

Field-level operations such as aggregations or partial queries, a fully fledged mapping step.

Code-generation(macros, derives, proc macros) for this is feasible and wanted at some point, but I want to get the API stable first.

- ▶ See [github.com/berlinrs/tbd.rs](https://github.com/berlinrs/tbd.rs)
- ▶ See "NOTES.md" for implementation notes and thoughts
- ▶ Find me on Discord(@skade)/Twitter(@argorak)/wherever if you want to discuss