Thanks Man ORM Documentation

Zachary Metz

May 24, 2019

Quick Start

Getting Set up

- Once Database Abstract is imported call its create function and it will return to you an object that has automaticall set itself up with your table schema

```
const db = DatabaseAbstract.create({
    hostname = "",
    username = "",
    password = "",
    databaseName = "",
    schema = ""
})
```

- Now you can access tables like this :

```
let currentUser = await db.user.where(x=>x.userid == 3).fi
db.users.first(100).toList();
```

Basics

About

Thanks Man is an ORM made for postgres that tries to emulate as many feautrs of entity framework as possible while being dead simple to get started with and support for advanced functionality.

- *The DatabaseAbstract Object* - The object that represents the database in your code and is an nice interface to query and update objects from tables are accesable from db.TableName and all crud operation will happen to this object

How Get Rows From The Database

The most basic way to get objects from the database (using table user as an example) is :

```
var users = await db.user.toList();
```

How to Qurey For Information

You can add more complex queries by adding differnt modifiers that will affect the resulting data pull

```
var selectedUser = await db.user
    // use lambda functions for filtering
    .where(x => x.first_name == "john")
    .orderBy("last_login")
    .desc()
    .limit(20) //   only return 20 rows
    .offset(100)    //  skip the first 100 rows
    .toList();      //  return a list
```

How to Update Rows

Once objects are pulled from the database for us to use any modifications made to the rows can be saved very easily by invoking db.saveChanges(), it will automatically find changes to objects pulled and update the relevant columns in the db.

```
let currentUser = await db.user.firstOrDefault();
currentUser.favouriteColor = "Red";
await db.saveChanges();  // persists changes to db
```

How To Insert Rows

Inserting Records Into the database is also managed by the db.TableName object and a new row object can be aquried by calling db.tableObject.newRow(). This will return a new row object for you to populate before calling db.saveChanges().

```
let newUser = db.user.newRow();
newUser.email = "example@example.ca";
newUser.password = "secret";
newUser.lastLogin = (new Date()).getTime();
await db.saveChanges();
```

How To Delete Rows

Deleteing rows can be done by passing the row object that you want removed from the database to db.delete(rowObject) and then commiting the action with db.saveChanges(). This trigger a delete script that will be sent to the db.

```
let currentUser = await db.user.firstOrDefault();
db.delete(currenUser);
await db.saveChanges();
```

Examples

Retriving

Where

Used to filter results, will accept an sql string, filter object or a lambda function and applies it to the query.

```
.where(x => x.id == 43)
.where(x => array.includes(x.id))
.where("id = 43")
.where("id in (1,2,3,4,5)")
```

Join

Still to be implimented in orm funcionality.

Order By

Determines the column the query results will be ordered by.

```
.orderBy("id")
.orderBy("first_name")
.orderBy("email")
```

Desc / Asc

This modifier works in conjunction with the orderBy to determing if it is in decenting or ascending sort order.

```
//  to order them by decending
.desc()

//  to order by ascending
.asc()
```

Limit

The limit modifer will restric the number of rows returned to you via a query. It is a direct map to the limit sql keyord

```
//  only get 20 rows back
.limit(20)
//  only get 500 rows back
.limit(500)
```

Offset

The offset modifer will ignore the first n rows in the result and start returning them from the offset point instead of at the first one

```
//  skip the first 100 rows when returning
.offset(100)
```

First

This function will run the query and return a list of rows that corrispond to your request with the number of objects you specify. It impliments async and await.

```
//  returns the first 20 rows to you
let users = await db.user.first(20);
//  returns the first 100 rows to you
let users = await db.user.first(100);
```

First Or Default

Similar To the `.first(n)` method this function will run the query and return the first row object that is pulled or if there is null `null`.

```
//  returns the first 20 rows to you
let currentUser = await db.user.firstOrDefault();
```

To List

This function also runs the quries but has not restriction on the number of records returned it will just give everything that matched your query in the database back.

```
//  returns all of the user rows to you
let users = await db.user.toList();
```

Updating

Inserting

Deleting

Reference