# 

NodeJS ORM Module Transaction Fix Application Design Specification

|  |  |  |
| --- | --- | --- |
| **Author** | **Revision Number** | **Date** |
| Architect | 1.0 | Nov 4, 2014 |
|  |  |  |
|  |  |  |

[NodeJS ORM Module Transaction Fix Application Design Specification 1](#_Toc402854550)

[Application Design Specification 3](#_Toc402854551)

[1. Design 3](#_Toc402854552)

[1.1 Work Flow Description 3](#_Toc402854553)

[1.1.1 Transaction in Single Connection Mode 3](#_Toc402854554)

[1.1.2 Transaction in Connection Pooling Mode 3](#_Toc402854555)

[1.1.3 PoC 3](#_Toc402854556)

[1.1.4 Usage Example 4](#_Toc402854557)

[1.1.4.2 Transaction Usage in Connection Pooling Mode 5](#_Toc402854558)

[1.2 Component Requirements 6](#_Toc402854559)

[1.2.1 Custom Software Components 6](#_Toc402854560)

[1.2.2 TopCoder Generic Components 6](#_Toc402854561)

[1.2.3 Third Party Components 6](#_Toc402854562)

[1.2.4 Assemblies 7](#_Toc402854563)

[1.3 Application Management 7](#_Toc402854564)

[1.3.1 Transaction 7](#_Toc402854565)

[1.3.2 Configuration 7](#_Toc402854566)

[1.3.3 Persistence 7](#_Toc402854567)

[1.3.4 Thread-Safety and Concurrency 7](#_Toc402854568)

[1.3.5 Logging 7](#_Toc402854569)

[1.3.6 Auditing 7](#_Toc402854570)

[1.3.7 Exception Handling 7](#_Toc402854571)

[1.3.8 Internationalization 7](#_Toc402854572)

[1.3.9 Security 7](#_Toc402854573)

[1.3.10 Performance 7](#_Toc402854574)

[1.3.11 Scalability 7](#_Toc402854575)

[1.4 Deployment Constraints 7](#_Toc402854576)

[1.4.1 Technology overview 7](#_Toc402854577)

[1.5 Development Standards: 8](#_Toc402854578)

[1.6 Interfaces Classes Overview 8](#_Toc402854579)

[1.7 Changes to Existing System 8](#_Toc402854580)

[2. User Interface 8](#_Toc402854581)

[3. Included Documentation 8](#_Toc402854582)

[3.1 Architecture Documentation 8](#_Toc402854583)

[4. Future Enhancements 8](#_Toc402854584)

# Application Design Specification

# Design

We are using [node-orm2 module](https://github.com/dresende/node-orm2) in a nodejs application to access PostgreSQL database. Recently we started using transaction and we found that there is a bug in handling transaction in this module when using it in both single connection mode and connection pooling mode. We need them to be addressed properly.

## Work Flow Description

### Transaction in Single Connection Mode

We can only start and end (either commit or rollback) one transaction each time on a single connection. After the previous transaction is completed, we can start a new one, and so on.

The provided single.coffee script has two parallel transactions on the same connection, that’s why it fails. It’s not a bug in the [node-orm2 transaction plugin](https://github.com/dresende/node-orm-transaction).

### Transaction in Connection Pooling Mode

The [node-orm2 transaction plugin](https://github.com/dresende/node-orm-transaction) doesn’t work in connection pooling mode because whenever [node-orm2 module](https://github.com/dresende/node-orm2) executes a query, it fetches an idle connection from the connection pool, which may not be the same connection used by the other queries inside the transaction. Note that when start, commit or rollback a transaction, a query will be executed as well, and it may not use the same connection as the queries inside the transaction as well when using the plugin.

Imagine that you start the transaction on one connection, perform a model update on another connection, and commit the transaction on another different connection, no wonder it will fail.

The fix to this bug is simple: whenever a transaction is started, the connection is kept and reused until the transaction is completed (either committed or rolled-back). After the transaction is completed, the connection will be released back to the connection pool.

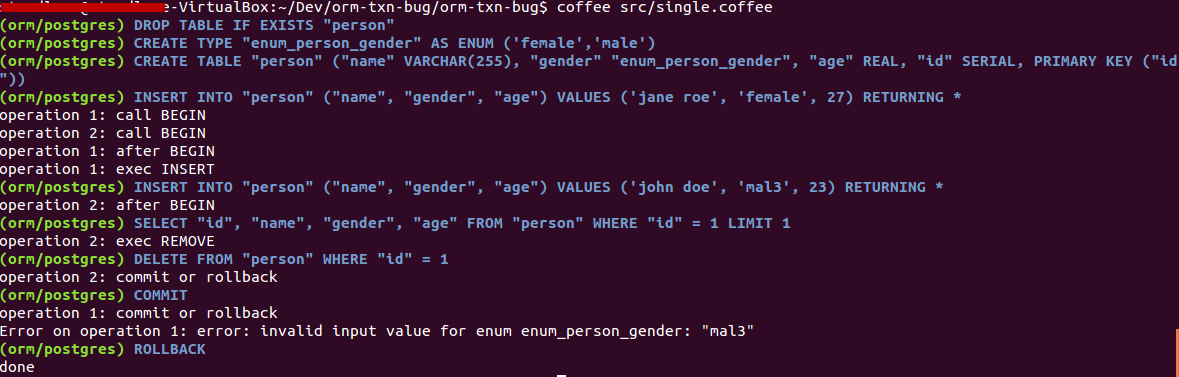
### PoC

I have updated the orm-txn-bug/node\_modules/orm module to apply all necessary changes to fix the transaction issue in connection pooling mode. A git diff file (diff.diff) is also provided to make it easy to track all the changes.

I have also updated single.coffee and multiple.coffee scripts in orm-txn-bug/src directory to make them work. The single.coffee is updated to use connection pool. (To run them, you need to install PostgreSQL, and update the values in config.coffee and env\_sample).

To run single.coffee, execute the following command from shell: coffee single.coffee

The result is like below (The operation-1 is rolled back, and the operation-2 is committed):



To run single.coffee, execute the following command from shell: coffee multiple.coffee

The result is like below (The transaction is rolled back successfully, no records are inserted):



### Usage Example

The transaction variable in all affected methods or functions is optional, so it won’t break the non-transaction mode.

#### *Transaction Usage in Single Connection Mode*

var orm = require(“orm”);

orm.connnect(“postgres://postgres:topcoder@localhost/mydb”, function(err, db) {

if (err) throw err;

var Person = db.define(“person”, {

name : String,

surname : String,

age : Number

gender: [ 'female', 'male' ]

, {

hooks: {

beforeCreate: function(next) {

// the hook automatically joins into the transaction as there is

// only one connection

}

}

});

// start the transaction

db.transaction(function(err, tx) {

Person.find({surname: ‘Doe’}).each(function(person) {

person.remove();

});

tx.commit(function(err) {

if (!err) console.log(‘success’);

});

// rollback example

tx.rollback(function(err) {

});

});

});

In single connection mode, the transaction usage is the same as the [node-orm2 transaction plugin](https://github.com/dresende/node-orm-transaction) except that you don’t need to include the plugin.

### Transaction Usage in Connection Pooling Mode

var orm = require(“orm”);

orm.connnect(“postgres://postgres:topcoder@localhost/mydb? pool=true”, function(err, db) {

if (err) throw err;

var Person = db.define(“person”, {

name : String,

surname : String,

age : Number,

gender: [ 'female', 'male' ]

}, {

hooks: {

beforeCreate: function(next) {

// get transaction

var tx = this.transaction;

}

}

});

var Animal = db.define(“animal”, {

name : String,

});

Animal.hasOne(“owner”, Person, {

reverse: “pets”

});

// start the transaction

db.transaction(function(err, tx) {

// find

Person.find({surname: ‘Doe’}, tx).each(function(person) {

person.remove();

});

// create

Person.create(

{ name: ‘John’, surname: ‘Doe’,

age: 25, gender: ‘male’

}, tx, function(err, person) {

// do something

});

// get, update and remove

Person.get(1, tx, function(err, person) {

person.name = ‘Mark’;

person.getPets();

person.save(function(err) {

});

person.remove(function(err) {

});

});

// aggregate

Person.aggregate({surname: ‘Doe’}, tx).min(“age”).max(“age”).get(err, min, max) {

// do something

});

tx.commit(function(err) {

if (!err) console.log(‘success’);

});

// rollback example

tx.rollback(function(err) {

});

});

});

In connection pooling mode, you have to pass the transaction variable (tx in above example) around in Model’s methods, and you don’t need to pass the transaction variable into Instance’s methods. (Person and Animal above are Models, while the person above is an instance).

The instance contains a transaction field, and you can use it inside the hooks (get it through this.transaction) to allow the hooks to join into the transaction. Note that in order to make the instance use the same connection as the wrapping transaction, the instance must be retrieved inside the transaction.

## Component Requirements

### Custom Software Components

#### *New Custom Components*

None

### TopCoder Generic Components

None

### Third Party Components

* [Node.js 0.10.29](http://nodejs.org)

This application is built on Node.js platform.

* [pg 2.6.2](https://github.com/brianc/node-postgres)

This is used to interact with the PostgreSQL database.

* [sql-query 0.1.15](https://github.com/dresende/node-sql-query)

This is used to build the sql query.

* [lodash 2.0.0](https://github.com/lodash/lodash)

It provides asynchronous utilities.

* [node-orm2 2.1.3](https://github.com/dresende/node-orm2)

This project will modify the orm library to support transaction.

* [enforce 0.1.2](https://github.com/dresende/node-enforce)

It’s used for data validation.

* [hat 0.0.3](https://github.com/substack/node-hat)

It’s used to generate random IDs.

### Assemblies

* NodeJS ORM Module Transaction Fix Assembly

This assembly will update [node-orm2 module](https://github.com/dresende/node-orm2) to fix transaction and write complete tests.

Note that the fix is only applicable for PostgreSQL database.

## Application Management

### Transaction

PostgreSQL database supports transaction. And to make the transaction work with the [node-orm2 module](https://github.com/dresende/node-orm2), the start and end transaction commands and all the queries inside the transaction should be performed on the same connection. No parallel transactions are allowed on the same connection.

### Configuration

No extra configuration is needed for the transaction fix of [node-orm2 module](https://github.com/dresende/node-orm2).

### Persistence

PostgreSQL database is used as the persistence store.

### Thread-Safety and Concurrency

Applications built with JavaScript and Node.js is inherently asynchronous, callbacks(JavaScript) are used extensively to coordinate asynchronous tasks/operations. There's generally no thread safety concern with Node.js powered applications.

### Logging

To follow the pattern in [node-orm2 module](https://github.com/dresende/node-orm2), only the executed queries need to be logged.

### Auditing

There is no auditing requirement.

### Exception Handling

The functions of JavaScript services are implemented in asynchronous manner, i.e. virtually all functions will take a callback function that will be called to notify function caller of result. If error occurs, callback function will be called with a String type parameter "err" detailing the error.

### Internationalization

There is no internationalization requirement.

### Security

There is no security requirement.

### Performance

Transaction is now supported in connection pooling mode, so that we can reuse connections in such cases.

### Scalability

There is no particular scalability problem identified at this stage.

## Deployment Constraints

The updated [node-orm2 module](https://github.com/dresende/node-orm2) will be used as a dependency to access PostgreSQL database data.

### Technology overview

* JavaScript
* [Node.js 0.10.29](http://nodejs.org)
* [pg 2.6.2](https://github.com/brianc/node-postgres)
* [sql-query 0.1.15](https://github.com/dresende/node-sql-query)
* [lodash 2.0.0](https://github.com/lodash/lodash)
* [node-orm2 2.1.3](https://github.com/dresende/node-orm2)
* [enforce 0.1.2](https://github.com/dresende/node-enforce)
* [hat 0.0.3](https://github.com/substack/node-hat)
* [PostgresSQL 9.x](http://www.postgresql.org/)

## Development Standards:

The assembly development must adhere to the guidelines as outlined in the [TopCoder Assembly Competition Tutorial](http://apps.topcoder.com/wiki/display/tc/Assembly+Competition+Tutorials).

## Interfaces Classes Overview

See the TCUML file.

## Changes to Existing System

The [node-orm2 module](https://github.com/dresende/node-orm2) is updated.

# User Interface

There is no UI.

# Included Documentation

## Architecture Documentation

* Class Diagrams
* Sequence Diagrams
* Application Design Specification
* Assembly Specification

# Future Enhancements

None