Simple Oracle ORM for Node.js

Rob Tucker
08/27/2018

Contents

Introduction 3
Project Layout 3
Design Concepts 4
Orm 5
Model 5
Repository 8
MetaData 9
Test 12
Application Configuration 13
appconfig.json: 13
testconfig.json: 14
Example Code 15
MetaData Details 15
Field Definition Details 16
Join Strategy 17
Reference Field Definitions 17
One-to-One Definition 17
One-to-Many Definition 17
Many-to-One 17
One-to-Many (with many-to-many configuration) 18
Repository Details 19
Object Based Database Operations 26
Provided Repository Object Operations 27
findOne 27
find 27
exists 27
count 27
save 27
delete 28
Provided Repository SQL Operations 28
executeSqlQuery 28
executeSql 28
Repository Operation options Parameter 28

WhereComparison.js 29
OrderByEntry.js 30
Database Configuration 31
Getting Started 31
Some Example Code 31
findOne 32
find 32
count 32
exists 32
getAll 33
save 33
delete 33
executeSqlQuery 34
executeSql 34
REST Access 34

Introduction

This document describes a simple JavaScript-based Oracle ORM for Node.js. The application is available in the npm repository.

The ORM models JPA concepts to some extent and allows Object based database access. OQL-like queries can be created like simple example below:

```
select Account o
from Account
where o.accountNbr = :accountNbr
```

The framework uses the node-oracledb driver for database access and consists of 5 main areas:

- Orm – the application entry point. Loads models, metadata, repositories and the database interface. Also provides REST based access to database if desired.
- Model – standard object mapping of database tables
- Repository – provides CRUD functionality to/from database using the model
- MetaData – table-to-model definitions
- Test – unit test logic

Table access and testing is provided by JavaScript objects that extend provided base objects which must adhere to a standard naming convention. For example, and account table would be represented by the following JavaScript files.

- Account.js – model
- AccountRepository.js – database access
- AccountMetaData.js – metadata definitions
- AccountTest.js – model tests
- AccountRepositoryTest.js – repository tests

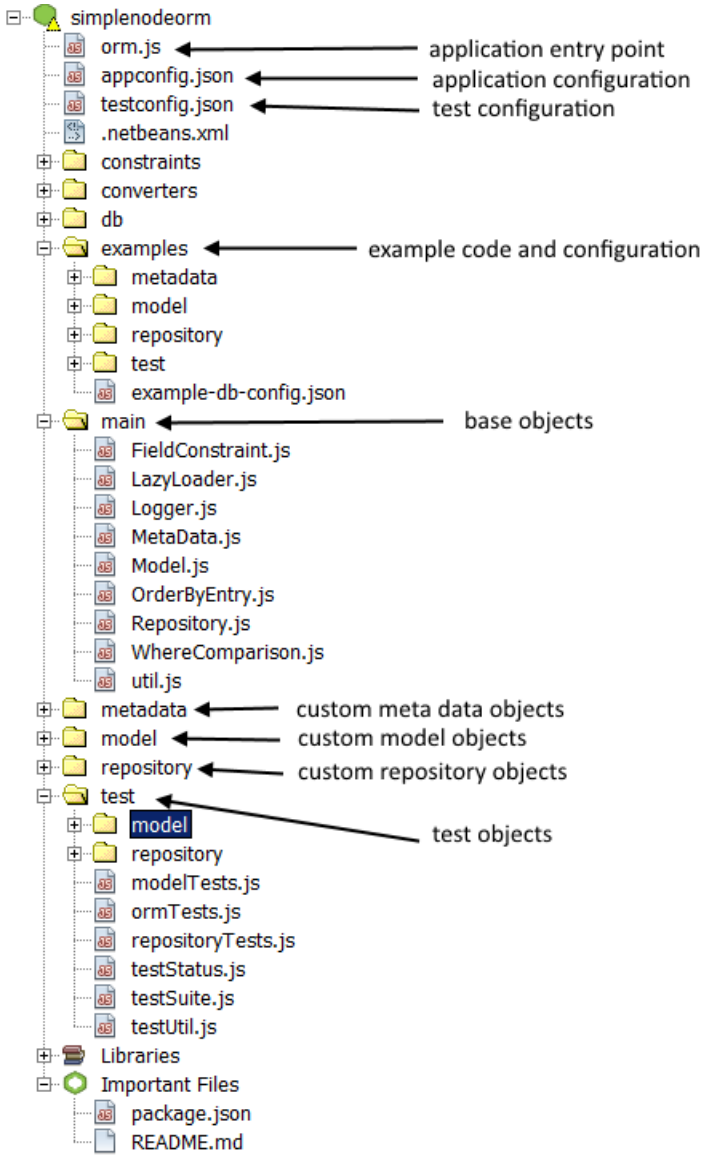
Each repository module supports the following object-based methods out of the box:

- findOne – find object by primary key
- find – find objects by input where parameters
- getAll – return all objects from associated table
- count – count based on where or count all
- exists – object exists in database
- save – performs one or more inserts/updates based on input
- delete – performs one or more delete operations based in input

Standard SQL queries and result sets are also supported.

Project Layout

The layout of the simplenodeorm project is shown in the diagram below. It is expected that custom model, metadata, repository and test objects will be found under the directory hierarchy in the model, metadata, repository and test/model, /test/repository folders:



The simplenodeorm project is available on github at:

https://github.com/rbtucker/simplenodeorm

Design Concepts

This ORM is designed to provide simple, configurable, efficient database access to Oracle databases via JavaScript objects and Node. As stated in the introduction, the application consists of 5 major areas – orm, model, repository, metadata and test. Each of these areas are built with JavaScript using the node module system. A general description of each area is found below:

Orm

The orm.js is the application module for simplenodeorm. The orm initiates the database interface by creating configured database pools then loads the repository and metadata modules. These modules are made available by model name via the methods below:

orm.getMetaData('modelName');

orm.getRepository('modelName');

The orm also provides database connections from the pool via the orm.getConnection('poolAlias') method.

REST access to the database is provided by the orm.js based on application configuration.

Model

A model object is the familiar data container that maps to a database table. Each Model object will extend Model.js to provide basic functionality:

```
"use strict";

const util = require("./util.js");
var lazyLoader;

class Model {
    constructor(metaData) {
        this.__model__ = metaData.objectName;
        this.metaData = metaData;
        this.modified = false;
        this.newModel = true;
        this.constraintsEnabled = false;
        this.data = new Object();
        this.initializeData();
    }

    isModified() {
        return this.modified;
    }

    setModified(modified) {
        this.modified = modified;
    }

    isNew() {
        return this.newModel;
    }

    setNew(newModel) {
        this.newModel = newModel;
    }

    getData() {
        return this.data;
    }

    setData(data) {
        this.data = data;
    }

    getFieldValue(fieldName, ignoreLazyLoad) {
        let retval =  this.data[fieldName];

        // only lazy load when running under node
        if (!ignoreLazyLoad
```

```
            && util.isUndefined(retval)
            && util.isNodeEnv()
            && this.metaData.isLazyLoad(fieldName)) {
            if (util.isUndefined(lazyLoader)) {
                lazyLoader = require('./LazyLoader.js');
            }

            lazyLoader.lazyLoadData(this, fieldName);
        }

        return retval;
    }

    setFieldValue(fieldName, value) {
        // if constraints are enabled then check incoming data
        if (this.metaData && this.constraintsEnabled) {
            let constraints = this.metaData.getFieldConstraints(fieldName);

            if (util.isValidObject(constraints)) {
                for (let i = 0; i < constraints.length; ++i) {
                    constraints[i].check(this.metaData.getObjectName(), fieldName, value);
                }
            }
        }

        if (!this.modified) {
            this.modified = (this.data[fieldName] !== value);
        }

        this.data[fieldName] = value;
    }

    getFields() {
        return this.metaData.getFields();
    }

    beforeLoad() {};
    beforeSave() {};
    afterLoad() {};
    afterSave() {};
    enableConstraints(enabled) { this.constraintsEnabled = enabled; };

    addFieldConstraint(fieldName, constraint) {
        this.metaData.addFieldConstraint(fieldName, constraint);
    };

    isLengthConstraintRequired(field) {
        return this.metaData.isLengthConstraintRequired(field);
    }

    getMaxLength(field) {
        return this.metaData.getMaxLength(field);
    }

    getObjectName() {
        return this.metaData.getObjectName();
    }

    getTableName() {
        return this.metaData.getTableName();
    }

    getModule() {
        return this.metaData.getModule();
    }

    getMetaData() {
        return this.metaData;
    }
```

```
    getOneToOneDefinitions() {
        return this.metaData.getOneToOneDefinitions();
    }

    getOneToManyDefinitions() {
        return this.metaData.getOneToManyDefinitions();
    }

    getManyToOneDefinitions() {
        return this.metaData.getManyToOneDefinitions();
    }

    initializeData() {
        // initialize related reference fields to undefined for
        // lazy load check later
        let reldefs = this.metaData.getOneToOneDefinitions();
        if (util.isValidObject(reldefs)) {
            for (let i = 0; i < reldefs.length; ++i) {
                this.data[reldefs[i].fieldName] = undefined;
            }
        }

        reldefs = this.metaData.getOneToManyDefinitions();
        if (util.isValidObject(reldefs)) {
            for (let i = 0; i < reldefs.length; ++i) {
                this.data[reldefs[i].fieldName] = undefined;
            }
        }

        reldefs = this.metaData.getManyToOneDefinitions();
        if (util.isValidObject(reldefs)) {
            for (let i = 0; i < reldefs.length; ++i) {
                this.data[reldefs[i].fieldName] = undefined;
            }
        }

        let fields = this.metaData.getFields();
        for (let i = 0; i < fields.length; ++i) {
            if (fields[i].lazyLoad) {
                this.data[fields[i].fieldName] = undefined;
            }
        }
    }
};

module.exports = Model;
```

A few remarks about the code above:

- “use strict” – enforces strict JavaScript coding, for example variable must be declared with “var” or “let”. If a variable is used and has not been declared an error will be thrown.
- “require” keyword – this is how you import other node modules. In this case I am importing the util.js module
- “class” - the is JavaScript ES6 construct that formalizes the common JavaScript pattern of simulating class-like inheritance hierarchies using functions and prototypes.
- metaData – database metadata definition associated with this object. When the object is serialized to JSON this information will be removed.
- __model__ - object name of extending class (Account, Chart etc.) with this name we can reconstitute an object from JSON data.
- modified – modified flag that tracks whether object has been updated
- newModel – flag to determine if this is a new model object (not in database).

- constraintsEnabled – model constraints can be enabled (field length checks etc.). Constraints will be checked when data is set in the object based on the constraintsEnabled flag which is false by default.
- data – this is where the object data values are stored. This is your standard JavaScript object used as a map = data['fieldName'] = someValue. In the extending classes there will be get/set data access method which allow access to the data – for example getAccountNbr(). These calls just pass through to this storage. This design allows easy access by field name for generic data processing and testing.

Below is a code snippet from the example Account.js module:

```
"use strict";

const Model = require('../../main/Model.js');

class Account extends Model {
    constructor(metaData) {
        super(metaData);
    }

    getFinCoaCd() { return this.getFieldValue("finCoaCd"); };
    setFinCoaCd(value) { this.setFieldValue("finCoaCd", value); };

    getAccountNbr() { return this.getFieldValue("accountNbr"); };
    setAccountNbr(value) { this.setFieldValue("accountNbr", value); };

    getAcCstmIcrexclCd() { return this.getFieldValue("acCstmIcrexclCd"); };
    setAcCstmIcrexclCd(value) { this.setFieldValue("acCstmIcrexclCd", value); };

    getAccountNm() { return this.getFieldValue("accountNm"); };
    setAccountNm(value) { this.setFieldValue("accountNm", value); };

    getAcctCityNm() { return this.getFieldValue("acctCityNm"); };
    setAcctCityNm(value) { this.setFieldValue("acctCityNm", value); };
```

Repository

The bulk of the work in the ORM goes on in the Repository.js object, this is where the Object-to-SQL logic is carried out and the database access is supported. Repository.js also contains the complex logic for the SQL result set to object graph conversion. Each repository object will extend the base class Repository.js a portion of which is shown below:

```
module.exports = class Repository {
    constructor(poolAlias, metaData) {
        this.metaData = metaData;
        this.poolAlias = poolAlias;
        this.selectedColumnFieldInfo = new Array();
        this.columnPositions = new Array();
        this.pkPositions = new Array();
        this.namedDbOperations = new Map();
        this.generatedSql = new Array();

        // default named db operations
        this.namedDbOperations.set(util.FIND_ONE, this.buildFindOneNamedOperation(metaData));
        this.namedDbOperations.set(util.GET_ALL, this.buildGetAllNamedOperation(metaData));
        this.namedDbOperations.set(util.DELETE, this.buildDeleteNamedOperation(metaData));
        this.selectClauses = new Array();
        this.joinClauses = new Array();

        // load custom db operations in extening classes. These are object-based db operations,
```

```
        // below is an example of Account findOne():
        // select Account o from Account where o.finCoaCd = :finCoaCd
        // and o.accountNbr = :accountNbr
        // currently the 'o' alias is important becausr the sql generator will
        // key on 'o.'. Specify
        // with dot notations for example o.subAccounts.subAcctNbr = :subAcctNbr
        this.loadNamedDbOperations();
    }
```

A few comments about the code above:

- const orm – this is the kfsnodeorm application module. When the application starts it builds maps of all repositories and metaData modules which can be accessed via the orm.getMetaData("objectName") and orm.getRepository("objectName") calls.
- poolAlias – each extending repository object will have an associated pool alias. A pool is created for each database connection that is configured. In this way we can run multiple database connections and access different databases based on the alias – KFS and RICE for example.
- Other maps and arrays defined – most of these are created for performance . There is a lot of work when converting 2 dimensional SQL result set to an object graph so we try to cache as much of this information as we can – for example, the primary key positions for various joined tables in the result rows.
- namedDbOperations – as stated in the introduction, pre-defined object queries can be defined and used – they will end up in this map. As you can see findOne and getAll object queries are created by default. Extending repository classes can override the loadNamedDbOperations() to add custom object queries.

  Below is the example AccountRepository.js module:

```
const orm = require('../../orm.js');
const poolAlias = 'mydb';
const Repository = require('../../main/Repository.js');

class AccountRepository extends Repository {
    constructor(metaData) {
        super(poolAlias, metaData);
    };

    loadNamedDbOperations() {
        // define named database operations here - the convention is as follows
        // namedDbOperations.set('functionName', 'objectQuery')
        // example: select Account o from Account where o.finCoaCd = :finCoaCd and
o.accountNbr := accountNbr
    };
};

module.exports = function(metaData) {
    return new AccountRepository(metaData);
};
```

MetaData

The metadata object contains object relational-to-object database definitions as well as the object relationship definitions. Each MetaData object will extend the base class MetaData.js:

```
const util = require("../main/util.js");

/**
 * this is the base class for defing the sql to object mapping definitions
 */
class MetaData {
```

```
    constructor(
        objectName,
        module,
        tableName,
        fields,
        oneToOneDefinitions,
        oneToManyDefinitions,
        manyToOneDefinitions) {
        this.objectName = objectName;
        this.module = module;
        this.tableName = tableName;
        this.fields = fields;
        this.oneToOneDefinitions = oneToOneDefinitions;
        this.oneToManyDefinitions = oneToManyDefinitions;
        this.manyToOneDefinitions = manyToOneDefinitions;
        this.fieldConstraints = new Map();
        this.lazyLoadFields = new Set();

        // map of column name to field definitions
        this.columnToFieldMap = new Map();

        // map of field name to field definitions
        this.fieldMap = new Map();

        // map of field name to field definitions
        this.referenceMap = new Map();

        // add some default constraints - will be disabled by default in the model object
        for (let i = 0; i < fields.length; ++i) {
            if (fields[i].lazyLoad) {
                this.lazyLoadFields.add(fields[i].fieldName);
            }

            this.columnToFieldMap.set(fields[i].columnName, fields[i]);
            this.fieldMap.set(fields[i].fieldName, fields[i]);
            if (fields[i].required) {
                let l = null;
                if (!this.fieldConstraints.has(fields[i].fieldName)) {
                    l = new Array();
                    this.fieldConstraints.set(fields[i].fieldName, l);
                } else {
                    l = this.fieldConstraints.get(fields[i].fieldName);
                }
                l.push(new (require("../constraints/NotNullConstraint.js")));
            }

            if (this.isLengthConstraintRequired(fields[i])) {
                let l = null;
                if (!this.fieldConstraints.has(fields[i].fieldName)) {
                    l = new Array();
                    this.fieldConstraints.set(fields[i].fieldName, l);
                } else {
                    l = this.fieldConstraints.get(fields[i].fieldName);
                }
                l.push(new (require("../constraints/LengthConstraint.js"))
                    (this.getMaxLength(fields[i])));
            }

            // add ref fields to lazy load set
            for (let i = 0; i < this.oneToOneDefinitions.length; ++i) {
                this.lazyLoadFields.add(this.oneToOneDefinitions[i].fieldName);
                this.referenceMap.set(this.oneToOneDefinitions[i].fieldName,
                    this.oneToOneDefinitions[i]);
            }

            for (let i = 0; i < this.oneToManyDefinitions.length; ++i) {
                this.lazyLoadFields.add(this.oneToManyDefinitions[i].fieldName);
                this.referenceMap.set(this.oneToManyDefinitions[i].fieldName,
                    this.oneToManyDefinitions[i]);
```

```
            }

            for (let i = 0; i < this.manyToOneDefinitions.length; ++i) {
                this.lazyLoadFields.add(this.manyToOneDefinitions[i].fieldName);
                this.referenceMap.set(this.manyToOneDefinitions[i].fieldName,
                   this.manyToOneDefinitions[i]);
            }

            this.loadConstraints();
        }
    }

    /**
     *
     * @returns ordered array of primary key field definitions
     */
    getPrimaryKeyFields() {
        let retval = new Array();
        for (let i = 0; i < this.fields.length; i++) {
            if (this.fields[i].primaryKey) {
                retval.push(this.fields[i]);
            }
        }
        return retval;
    }
```

As you can see this is a generic class that stores the definitions. The extending class will pass in table/field definitions in constructor as JSON in the super() call. Also note that in this base class we are creating LengthConstraint and a NotNullConstraint for fields by default if required. These will be checked when data is set on the model if the model.enableConstraints flag is true. Custom constraints can be added in extending classes by overriding the loadConstraints() method.

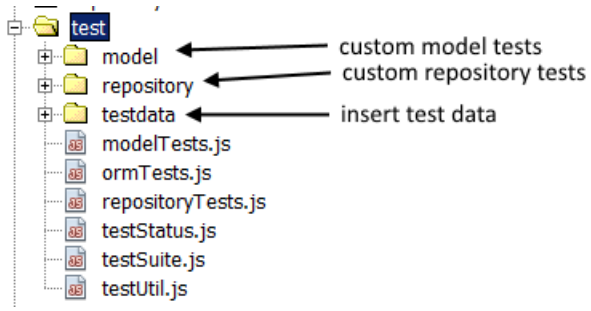
A portion of the example AccountMetaData.js module is shown below:

```
var MetaData = require('../../main/MetaData.js').MetaData;

class AccountMetaData extends MetaData {
    constructor() {
        super(
        'Account', // object name,
        'model/ca/Account.js', // relative module path,
        'CA_ACCOUNT_T', // table name
        [ // field definitions - order is important,
          //selected data will be in this order, primary key fields shoud be first
            { // 0
                fieldName: "finCoaCd",
                type: "VARCHAR2",
                length: 2,
                columnName: "FIN_COA_CD",
                required: true,
                primaryKey: true
            },
            { // 1
                fieldName: "accountNbr",
                type: "VARCHAR2",
                length: 7,
                columnName: "ACCOUNT_NBR",
                required: true,
                primaryKey: true
            },
            { // 2
                fieldName: "acCstmIcrexclCd",
                type: "VARCHAR2",
                length: 1,
```

```
                columnName: "AC_CSTM_ICREXCL_CD"
            },
        ],
        .
        .
        .
        [ // one-to-one definitions
            { // 0
               fieldName: "organization",
               type: 1,
               targetModelName: "Organization",
               targetModule: "../model/ca/Organization.js",
               targetTableName: "CA_ORG_T",
               status: "enabled",
               joinColumns : {
                   sourceColumns : "FIN_COA_CD,ORG_CD",
                   targetColumns : "FIN_COA_CD,ORG_CD"
               }
            },
            { // 1
               fieldName: "accountType",
               type: 1,
               targetModelName: "AccountType",
               targetModule: "../model/ca/AccountType.js",
               targetTableName: "CA_ACCOUNT_TYPE_T",
               status: "enabled",
               joinColumns : {
                   sourceColumns : "ACCT_TYP_CD",
                   targetColumns : "ACCT_TYP_CD"
               }
            },
        ],
        [ // one-to-many definitions
            { // 0
               fieldName: "subAccounts",
               type: 2,
               targetModelName: "SubAccount",
               targetModule: "../model/ca/SubAccount.js",
               targetTableName: "CA_SUB_ACCT_T",
               cascadeUpdate: true,
               cascadeDelete: true,
               status: "enabled",
               joinColumns : {
                   sourceColumns : "FIN_COA_CD,ACCOUNT_NBR",
                   targetColumns : "FIN_COA_CD,ACCOUNT_NBR"
               }
            }
        ]); // many-to-one definitions

    }
```

Test

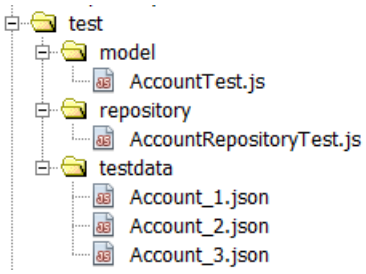
Below is the test hierarchy in the project:



If testMode in appconfig.json is set to true, then tests will be run after ORM startup. The entry point for all tests is testSuite.js.

```
const util = require("../main/util.js");
const testUtil = require("./testUtil.js");
const ormTests = module.require("./ormTests.js");
const modelTests = module.require("./modelTests.js");
const repositoryTests = module.require("./repositoryTests.js");
const assert = require('chai').assert;
const orm = require("../orm.js");

module.exports.run = async function() {
    testUtil.logInfo("running testSuite...");

    try {
        await ormTests.run(orm);
        await modelTests.run(orm);
        await repositoryTests.run(orm);
    }

    catch (e) {
        testUtil.logError('Exception in testSuite ' + e.stack);
    }

    testUtil.logInfo("testSuite complete");
};
```

The underlying test logic uses the data in an associated test database to dynamically test the CRUD capabilities of the ORM. This logic works well for the read, update and delete if each test table contains valid data. This logic does not work so well for the create/insert tests. To test create/insert functionality, insert files are created using the model name such as the example account test files shown below:



Each file contains the JSON representation of an account to be used for insert. To create a file, use the results returned from a Repository.findOne() call and modify as required.

Application Configuration

There are 2 JSON files for application and testing configuration:

appconfig.json:

```
{
  "testMode" : true,
  "dbConfiguration" : "path-to-dbconfig-json",
  "defaultMaxJoinDepth" : 4,
  "startRestServer" : false,
  "restUrlBase" : "/orm",
  "restPort" : 8888,
  "applicationName" : "Simple ORM",
```

```
  "logFile" : "orm.log",
  "logLevel" : "info",
  "maxRowsForGetAll" : 100000,
  "aliases" : {
        "simplename1" : "somelongobjectnamingconvention1",
        "simplename2" : "somelongobjectnamingconvention2"
  }
}
```

name	description
testMode	If set to true, unit tests will run after orm startup
dbConfiguration	Path to database configuration file
defaultMaxJoinDepth	Will determine how deep to create parent-to-child joins in generated SQL
startRestServer	If set to true, REST server for database access will be started after ORM initiation
restUrlBase	URL base path for REST calls
restPort	Port for REST access
applicationName	Display name for application
logFile	Path to log file
logLevel	Default log level for application
maxRowsForGetAll	Limit on number of rows returned from getAll() calls
aliases	By default, REST calls are of the form localhost:8888/kfsorm/account/findOne?finCoaCd=EA&accountNbr=4769235 Where account is the desired model to access. In cases where long names become unwieldy you can use an alias by defining it here.

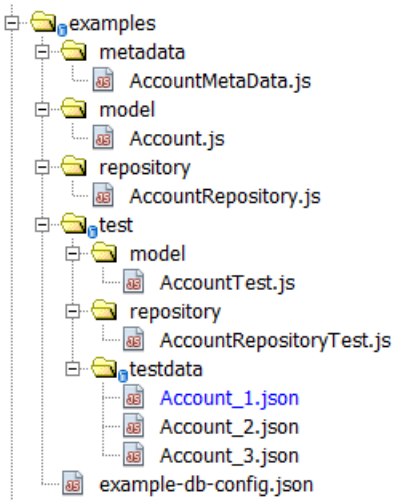
testconfig.json:

```
{
    "stopTestsOnFailure" : false,
    "testDbConfiguration" : "path-to-test-dbconfig-json",
    "maxRowsForGetAll" : 100,
    "fieldsToIgnoreForUpdate" : [
        "fieldName1",
        "fieldName2"
    ]
}
```

name	description
stopTestsOnFailure	If set to true testing process will stop when error is encountered
testDbConfiguration	Path to test database configuration
maxRowsFoGetAll	Limit on number of rows returned from getAll() calls
fieldsToIgnoreForUpdate	The test logic looks at database tables under test and attempts to find fields that can safely be updated for update tests. By default, primary keys, foreign keys and non-nullable fields are ignored. If there are other fields that should be ignored they can be entered here.

Example Code

Example code is found under the “examples” directory of the project.



MetaData Details

The metadata definitions are crucial to the operation of the ORM. Database queries are generated as joins based on the metadata definitions. The depth of the resulting object hierarchy – and the complexity of the required join to get this depth is driven by the maxDefaultJoinDepth setting in appconfig.json. A MetaData object consists of field and reference definitions. There are 3 types of reference definitions supported:

- One to Many
- One to One
- Many to One

All MetaData objects must extend the base object shown below:

class AccountMetaData extends MetaData {

```
  constructor() {
```

Required definitions are passed to the base class in the following order:

1. objectName – name of model object, for example 'Account'
2. module – relative path to Node.js module, for example 'model/Account.js'
3. tableName associated database table name
4. fields – JSON array of field definitions
5. oneToOneDefinitions – JSON array of one to one reference definitions (empty array if none)
6. oneToManyDefinitions - JSON array of one to many reference definitions (empty array if none)
7. manyToOneDefinitions - JSON array of many to one reference definitions (empty array if none)

Many to Many reference definitions are supported under the One to Many definitions with a link table definition. Order of the field definition array entries is important. It is required that the primary key fields appear first and in the correct order. Generated select clauses will be in the order of the field entries.

Field Definition Details

Below is a field definition from the example AccountMetaData.js:

```
{ // 0
  fieldName: "finCoaCd",
  type: "VARCHAR2",
  length: 2,
  columnName: "FIN_COA_CD",
  required: true,
  primaryKey: true
},
```

Available field definitions are listed below:

name	required	description
fieldName	Y	Name that is associated with the database column data
type	Y	Database type name
length	N	Maximum field length for string type fields
columnName	Y	Database column name
required	N	If true, the field is not nullable
primaryKey	N	If true, then this field is part of the primary key
autoIncrementGenerator	N	In oracle, this would be the sequence name
defaultValue	N	If defined and the field is null, the defaultValue will be used in inserts and updates
converter	N	If defined, the named converter will be used to convert values when retrieving/saving values to/from the database.
lob	N	If true, this indicates a LOB field
lazyLoad	N	If true, lazy load will be performed when getField() called
decimalDigits	N	Precision for numeric fields

Join Strategy

To implement database references the ORM builds SQL joins to pull data as a result set then populates the object graph from this result set. For efficiency reasons the following join strategy is used:

1. One-to-Many joins are created to the depth specified in the appconfig.json value "`defaultMaxJoinDepth`"
2. One-to-One and Many-to-One joins are only created for the top-level object of the hierarchy.
3. All other defined references will be populated via lazy load if/when the associated Model.getField() method is called.

Reference Field Definitions

Join definitions define parent-child relationships between tables. 3 types are supported one-to-one and one-to-many and many-to-one. Many-to-many joins are supported via a one-to-many definition with an associated join table defined. Below are some example definitions:

One-to-One Definition

```
    { // 1 - from example AccountMetaData
    fieldName: "organization",
    type: 1,
    targetModelName: "Organization",
    targetModule: "../model/ca/Organization.js",
    targetTableName: "CA_ORG_T",
    status: "enabled",
    joinColumns : {
        sourceColumns : "FIN_COA_CD,ORG_CD",
        targetColumns : "FIN_COA_CD,ORG_CD"
    }
},
```

One-to-Many Definition

```
    { // 0 - from example AccountMetaData
    fieldName: "subAccounts",
    type: 2,
    targetModelName: "SubAccount",
    targetModule: "../model/ca/SubAccount.js",
    targetTableName: "CA_SUB_ACCT_T",
    cascadeUpdate: true,
    cascadeDelete: true,
    status: "enabled",
    joinColumns : {
        sourceColumns : "FIN_COA_CD,ACCOUNT_NBR",
        targetColumns : "FIN_COA_CD,ACCOUNT_NBR"
    }
},
```

Many-to-One

```
{ // 0 - from child of account such as SubAccountMetaData
    fieldName: "account",
    type: 3,
    targetModelName: "Account",
    targetModule: "../model/Account.js",
    targetTableName: "CA_ACCOUNT_T",
    status: "enabled",
    joinColumns : {
        sourceColumns : "FIN_COA_CD,ACCOUNT_NBR",
        targetColumns : "FIN_COA_CD,ACCOUNT_NBR"
    }
}
```

One-to-Many (with many-to-many configuration)

```
{ // 0
   fieldName: "prerequisites",
   type: 2,
   targetModelName: "TermSpecification",
   targetModule: "../model/krms/TermSpecification.js",
   targetTableName: "KRMS_TERM_SPEC_T",
   joinTableName: "KRMS_TERM_RSLVR_INPUT_SPEC_T",
   status: "enabled",
   joinColumns : {
       sourceColumns : "TERM_RSLVR_ID",
       targetColumns : "TERM_RSLVR_ID",
       inverseSourceColumns : "TERM_SPEC_ID",
       inverseTargetColumns : "TERM_SPEC_ID"
   }
},
```

The logic in these definitions differs from the JPA annotation logic in that the terms target and source are always based on the current model object regardless of the relationship. If I am in the AccountMetaData.js module, source refers to the Account and target refers to the related object. fieldname, targetModelName, targetModule, targetTableName, type, status and joinColumns are required. Below are the available field descriptions:

name	required	description
fieldname	Y	The field name used for access in the Model object
targetModelName	Y	Model name associated with the linked database table
targetModel	Y	Relative path to the node.js module for the target model
targetTableName	Y	Linked database table name
type	Y	1, 2 or 3 where 1=one-to-one, 2=one-to-many and 3=many-to-one
cascadeUpdate	N	If true, parent update/inserts will cascade down to children
cascadeDelete	N	If true, parent deletes will cascade down to children
Status	Y	Enabled or disabled. You can turn off defined relationships if desired
Required	N	If true, an inner join will be created for this reference, otherwise an outer join will be created.

joinColumns	Y	Comma delimited list of source table to target table columns
		joinColumns : {
		sourceColumns : "ID1,ID2",
		targetColumns : "TID1,TID2"
		}

joinTableName	N	Used to support many-to-many references, define the linking table. If this is defined, then the join columns must contain the associated inverse column definitions, for example:
		joinColumns : {
		sourceColumns : "TERM_RSLVR_ID",
		targetColumns : "TERM_RSLVR_ID",
		inverseSourceColumns : "TERM_SPEC_ID",
		inverseTargetColumns : "TERM_SPEC_ID"
		}

Repository Details

The bulk of the work in the ORM goes on in the Repository.js base class. All custom repository objects must extend Respository:

```
"use strict";

const orm = require('../../orm.js');
const poolAlias = 'mydb';
const Repository = require('../../main/Repository.js');

class AccountRepository extends Repository {
    constructor(metaData) {
        super(poolAlias, metaData);
    };

    loadNamedDbOperations() {
        // define named database operations here - the convention is as follows
        // namedDbOperations.set('functionName', 'objectQuery')
        // example: select Account o from Account where o.finCoaCd
        //    = :finCoaCd and o.accountNbr := accountNbr
    };
};
```

```
module.exports = function(metaData) {

  return new AccountRepository(metaData);

};
```

When running queries SQL is generated as one select with joins to related tables as defined by the one-to-one and one-to-many and many-to-one definitions in the associated metadata. The "defaultMaxJoinDepth" entry in appconfig.json defines how deep the repository will traverse down the parent/child hierarchy and create joins – the deeper you go, the bigger the select statement and the poorer the performance. The top-level class always designated with alias “t0” will be the only table to define joins on one-to-one and many-to-one relationships. Other relationship definitions will be executed via lazy-load logic. The repository object supports running a query with a join depth specified at any level from 0 to the defaultMaxJoinDepth. A join depth of 0 is a special case where only the column data for the top-level object is pulled– no joins are created. If you need a high-performance query on a large table and you do not need related information you can run the query with joinDepth = 0. If no specific joinDepth is specified for a query the defaultMaxJoinDepth will be used.

Below is an example of the SQL generated findOne with defaultMaxJoinDepth = 4.

```
SELECT
  t0.FIN_COA_CD,
  t0.ACCOUNT_NBR,
  t0.SUB_ACCT_NBR,
  t0.FIN_RPT_CD,
  t0.FIN_RPT_CHRT_CD,
  t0.FIN_RPT_ORG_CD,
  t0.LAST_UPDT_TS,
  t0.OBJ_ID,
  t0.SUB_ACCT_ACTV_CD,
  t0.SUB_ACCT_NM,
  t0.VER_NBR,
  t0_t407_0.FIN_COA_CD,
```

```
  t0_t407_0.ACCOUNT_NBR,
  t0_t407_0.AC_CSTM_ICREXCL_CD,
  t0_t407_0.ACCOUNT_NM,
  t0_t407_0.ACCT_CITY_NM,
  t0_t407_0.ACCT_CLOSED_IND,
  t0_t407_0.ACCT_CREATE_DT,
  t0_t407_0.ACCT_EFFECT_DT,
  t0_t407_0.ACCT_EXPIRATION_DT,
  t0_t407_0.ACCT_FRNG_BNFT_CD,
  t0_t407_0.ACCT_FSC_OFC_UID,
  t0_t407_0.ACCT_ICR_TYP_CD,
  t0_t407_0.ACCT_IN_FP_CD,
  t0_t407_0.ACCT_MGR_UNVL_ID,
  t0_t407_0.ACCT_OFF_CMP_IND,
  t0_t407_0.ACCT_PHYS_CMP_CD,
  t0_t407_0.ACCT_PND_SF_CD,
  t0_t407_0.ACCT_RSTRC_STAT_CD,
  t0_t407_0.ACCT_RSTRC_STAT_DT,
  t0_t407_0.ACCT_SF_CD,
  t0_t407_0.ACCT_SPVSR_UNVL_ID,
  t0_t407_0.ACCT_STATE_CD,
  t0_t407_0.ACCT_STREET_ADDR,
  t0_t407_0.ACCT_TYP_CD,
  t0_t407_0.ACCT_ZIP_CD,
  t0_t407_0.BDGT_REC_LVL_CD,
  t0_t407_0.CG_ACCT_RESP_ID,
  t0_t407_0.CG_CFDA_NBR,
  t0_t407_0.CONT_ACCOUNT_NBR,
  t0_t407_0.CONT_FIN_COA_CD,
  t0_t407_0.CONTR_CTRL_FCOA_CD,
  t0_t407_0.CONTR_CTRLACCT_NBR,
  t0_t407_0.ENDOW_ACCOUNT_NBR,
  t0_t407_0.ENDOW_FIN_COA_CD,
  t0_t407_0.FIN_EXT_ENC_SF_CD,
  t0_t407_0.FIN_HGH_ED_FUNC_CD,
  t0_t407_0.FIN_INT_ENC_SF_CD,
  t0_t407_0.FIN_OBJ_PRSCTRL_CD,
  t0_t407_0.FIN_PRE_ENC_SF_CD,
  t0_t407_0.FIN_SERIES_ID,
  t0_t407_0.FUNDS_TYPE_CD,
  t0_t407_0.ICR_ACCOUNT_NBR,
  t0_t407_0.ICR_FIN_COA_CD,
  t0_t407_0.INCOME_ACCOUNT_NBR,
  t0_t407_0.INCOME_FIN_COA_CD,
  t0_t407_0.LAST_UPDT_TS,
  t0_t407_0.LBR_BEN_RT_CAT_CD,
  t0_t407_0.OBJ_ID,
  t0_t407_0.ORG_CD,
  t0_t407_0.RPTS_TO_ACCT_NBR,
  t0_t407_0.RPTS_TO_FIN_COA_CD,
  t0_t407_0.SUB_FUND_GRP_CD,
  t0_t407_0.VER_NBR,
  t0_t407_0_t252_1.FIN_COA_CD,
  t0_t407_0_t252_1.ACCOUNT_NBR,
  t0_t407_0_t252_1.SUB_ACCT_NBR,
  t0_t407_0_t252_1.FIN_RPT_CD,
  t0_t407_0_t252_1.FIN_RPT_CHRT_CD,
  t0_t407_0_t252_1.FIN_RPT_ORG_CD,
  t0_t407_0_t252_1.LAST_UPDT_TS,
  t0_t407_0_t252_1.OBJ_ID,
  t0_t407_0_t252_1.SUB_ACCT_ACTV_CD,
  t0_t407_0_t252_1.SUB_ACCT_NM,
  t0_t407_0_t252_1.VER_NBR,
  t0_t407_0_t253_1.CA_ICR_ACCT_GNRTD_ID,
  t0_t407_0_t253_1.ACCOUNT_NBR,
  t0_t407_0_t253_1.ACLN_PCT,
  t0_t407_0_t253_1.DOBJ_MAINT_CD_ACTV_IND,
  t0_t407_0_t253_1.FIN_COA_CD,
  t0_t407_0_t253_1.ICR_FIN_ACCT_NBR,
```

```
  t0_t407_0_t253_1.ICR_FIN_COA_CD,
  t0_t407_0_t253_1.OBJ_ID,
  t0_t407_0_t253_1.VER_NBR,
  t0_t402_0.FIN_COA_CD,
  t0_t402_0.EXPBDGT_ELIMOBJ_CD,
  t0_t402_0.FIN_AP_OBJ_CD,
  t0_t402_0.FIN_AR_OBJ_CD,
  t0_t402_0.FIN_CASH_OBJ_CD,
  t0_t402_0.FIN_COA_ACTIVE_CD,
  t0_t402_0.FIN_COA_DESC,
  t0_t402_0.FIN_EXT_ENC_OBJ_CD,
  t0_t402_0.FIN_INT_ENC_OBJ_CD,
  t0_t402_0.FIN_PRE_ENC_OBJ_CD,
  t0_t402_0.FND_BAL_OBJ_CD,
  t0_t402_0.ICR_EXP_FIN_OBJ_CD,
  t0_t402_0.ICR_INC_FIN_OBJ_CD,
  t0_t402_0.INCBDGT_ELIMOBJ_CD,
  t0_t402_0.LAST_UPDT_TS,
  t0_t402_0.OBJ_ID,
  t0_t402_0.RPTS_TO_FIN_COA_CD,
  t0_t402_0.VER_NBR,
  t0_t403_0.FIN_COA_CD,
  t0_t403_0.ORG_CD,
  t0_t403_0.CMP_PLNT_ACCT_NBR,
  t0_t403_0.CMP_PLNT_COA_CD,
  t0_t403_0.LAST_UPDT_TS,
  t0_t403_0.OBJ_ID,
  t0_t403_0.ORG_ACTIVE_CD,
  t0_t403_0.ORG_BEGIN_DT,
  t0_t403_0.ORG_CITY_NM,
  t0_t403_0.ORG_CNTRY_CD,
  t0_t403_0.ORG_DFLT_ACCT_NBR,
  t0_t403_0.ORG_END_DT,
  t0_t403_0.ORG_IN_FP_CD,
  t0_t403_0.ORG_LN1_ADDR,
  t0_t403_0.ORG_LN2_ADDR,
  t0_t403_0.ORG_MGR_UNVL_ID,
  t0_t403_0.ORG_NM,
  t0_t403_0.ORG_PHYS_CMP_CD,
  t0_t403_0.ORG_PLNT_ACCT_NBR,
  t0_t403_0.ORG_PLNT_COA_CD,
  t0_t403_0.ORG_STATE_CD,
  t0_t403_0.ORG_TYP_CD,
  t0_t403_0.ORG_ZIP_CD,
  t0_t403_0.RC_CD,
  t0_t403_0.RPTS_TO_FIN_COA_CD,
  t0_t403_0.RPTS_TO_ORG_CD,
  t0_t403_0.VER_NBR,
  t0_t404_0.FIN_COA_CD,
  t0_t404_0.EXPBDGT_ELIMOBJ_CD,
  t0_t404_0.FIN_AP_OBJ_CD,
  t0_t404_0.FIN_AR_OBJ_CD,
  t0_t404_0.FIN_CASH_OBJ_CD,
  t0_t404_0.FIN_COA_ACTIVE_CD,
  t0_t404_0.FIN_COA_DESC,
  t0_t404_0.FIN_EXT_ENC_OBJ_CD,
  t0_t404_0.FIN_INT_ENC_OBJ_CD,
  t0_t404_0.FIN_PRE_ENC_OBJ_CD,
  t0_t404_0.FND_BAL_OBJ_CD,
  t0_t404_0.ICR_EXP_FIN_OBJ_CD,
  t0_t404_0.ICR_INC_FIN_OBJ_CD,
  t0_t404_0.INCBDGT_ELIMOBJ_CD,
  t0_t404_0.LAST_UPDT_TS,
  t0_t404_0.OBJ_ID,
  t0_t404_0.RPTS_TO_FIN_COA_CD,
  t0_t404_0.VER_NBR,
  t0_t405_0.FIN_RPT_CHRT_CD,
  t0_t405_0.FIN_RPT_ORG_CD,
  t0_t405_0.FIN_RPT_CD,
```

```
  t0_t405_0.FIN_REP_CD_MGR_ID,
  t0_t405_0.FIN_RPT_CD_DESC,
  t0_t405_0.FIN_RPTS_TO_RPT_CD,
  t0_t405_0.LAST_UPDT_TS,
  t0_t405_0.OBJ_ID,
  t0_t405_0.ROW_ACTV_IND,
  t0_t405_0.VER_NBR,
  t0_t406_0.FIN_COA_CD,
  t0_t406_0.ACCOUNT_NBR,
  t0_t406_0.SUB_ACCT_NBR,
  t0_t406_0.CST_SHR_COA_CD,
  t0_t406_0.CST_SHRSRCACCT_NBR,
  t0_t406_0.CST_SRCSUBACCT_NBR,
  t0_t406_0.FIN_SERIES_ID,
  t0_t406_0.ICR_ACCOUNT_NBR,
  t0_t406_0.ICR_FIN_COA_CD,
  t0_t406_0.ICR_TYP_CD,
  t0_t406_0.OBJ_ID,
  t0_t406_0.OFF_CMP_CD,
  t0_t406_0.SUB_ACCT_TYP_CD,
  t0_t406_0.VER_NBR,
  t0_t406_0_t230_1.CA_A21_ICR_ACCT_GNRTD_ID,
  t0_t406_0_t230_1.ACCOUNT_NBR,
  t0_t406_0_t230_1.ACLN_PCT,
  t0_t406_0_t230_1.DOBJ_MAINT_CD_ACTV_IND,
  t0_t406_0_t230_1.FIN_COA_CD,
  t0_t406_0_t230_1.ICR_FIN_ACCT_NBR,
  t0_t406_0_t230_1.ICR_FIN_COA_CD,
  t0_t406_0_t230_1.OBJ_ID,
  t0_t406_0_t230_1.SUB_ACCT_NBR,
  t0_t406_0_t230_1.VER_NBR
FROM CA_SUB_ACCT_T t0
JOIN CA_ACCOUNT_T t0_t407_0
  ON (t0_t407_0.FIN_COA_CD = t0.FIN_COA_CD
  AND t0_t407_0.ACCOUNT_NBR = t0.ACCOUNT_NBR)
LEFT OUTER JOIN CA_SUB_ACCT_T t0_t407_0_t252_1
  ON (t0_t407_0_t252_1.FIN_COA_CD = t0_t407_0.FIN_COA_CD
  AND t0_t407_0_t252_1.ACCOUNT_NBR = t0_t407_0.ACCOUNT_NBR)
LEFT OUTER JOIN CA_ICR_ACCT_T t0_t407_0_t253_1
  ON (t0_t407_0_t253_1.FIN_COA_CD = t0_t407_0.FIN_COA_CD
  AND t0_t407_0_t253_1.ACCOUNT_NBR = t0_t407_0.ACCOUNT_NBR)
LEFT OUTER JOIN CA_CHART_T t0_t402_0
  ON (t0_t402_0.FIN_COA_CD = t0.FIN_COA_CD)
LEFT OUTER JOIN CA_ORG_T t0_t403_0
  ON (t0_t403_0.FIN_COA_CD = t0.FIN_RPT_CHRT_CD
  AND t0_t403_0.ORG_CD = t0.FIN_RPT_ORG_CD)
LEFT OUTER JOIN CA_CHART_T t0_t404_0
  ON (t0_t404_0.FIN_COA_CD = t0.FIN_RPT_CHRT_CD)
LEFT OUTER JOIN FP_RPT_CD_T t0_t405_0
  ON (t0_t405_0.FIN_RPT_CHRT_CD = t0.FIN_RPT_CHRT_CD
  AND t0_t405_0.FIN_RPT_ORG_CD = t0.FIN_RPT_ORG_CD
  AND t0_t405_0.FIN_RPT_CD = t0.FIN_RPT_CD)
LEFT OUTER JOIN CA_A21_SUB_ACCT_T t0_t406_0
  ON (t0_t406_0.FIN_COA_CD = t0.FIN_COA_CD
  AND t0_t406_0.ACCOUNT_NBR = t0.ACCOUNT_NBR
  AND t0_t406_0.SUB_ACCT_NBR = t0.SUB_ACCT_NBR)
LEFT OUTER JOIN CA_A21_ICR_ACCT_T t0_t406_0_t230_1
  ON (t0_t406_0_t230_1.FIN_COA_CD = t0_t406_0.FIN_COA_CD
  AND t0_t406_0_t230_1.ACCOUNT_NBR = t0_t406_0.ACCOUNT_NBR
  AND t0_t406_0_t230_1.SUB_ACCT_NBR = t0_t406_0.SUB_ACCT_NBR)
WHERE t0.FIN_COA_CD = :finCoaCd
AND t0.ACCOUNT_NBR = :accountNbr
AND t0.SUB_ACCT_NBR = :subAcctNbr
```

And here is the JSONresults from a findOne() call the call above:

```
{
   "__model__":"SubAccount",
   "modified":false,
   "newModel":false,
   "constraintsEnabled":false,
   "data":{
      "chart":{
         "__model__":"Chart",
         "modified":false,
         "newModel":false,
         "constraintsEnabled":false,
         "data":{
            "finCoaCd":"EA",
            "expbdgtElimobjCd":"1209",
            "finApObjCd":"9041",
            "finArObjCd":"8118",
            "finCashObjCd":"8000",
            "finCoaActiveCd":true,
            "finCoaDesc":"EAST",
            "finExtEncObjCd":"9892",
            "finIntEncObjCd":"9891",
            "finPreEncObjCd":"9890",
            "fndBalObjCd":"9899",
            "icrExpFinObjCd":"5500",
            "icrIncFinObjCd":"1803",
            "incbdgtElimobjCd":"1209",
            "lastUpdtTs":"2009-07-01T04:00:00.000Z",
            "objId":"014F3DAF748BA448E043814FD28EA448",
            "rptsToFinCoaCd":"IU",
            "verNbr":1
         }
      },
      "org":null,
      "financialReportChart":null,
      "reportingCode":null,
      "a21SubAccount":null,
      "account":{
         "__model__":"Account",
         "modified":false,
         "newModel":false,
         "constraintsEnabled":false,
         "data":{
            "subAccounts":[
               {
                  "__model__":"SubAccount",
                  "modified":false,
                  "newModel":false,
                  "constraintsEnabled":false,
                  "data":{
                     "finCoaCd":"EA",
                     "accountNbr":"4769235",
                     "subAcctNbr":"OH",
                     "lastUpdtTs":"2009-07-01T04:00:00.000Z",
                     "objId":"35A5641EB28D91B6E043814FD88191B6",
                     "subAcctActvCd":true,
                     "subAcctNm":"OLDER HOOSIER",
                     "verNbr":1
                  }
               },
               {
                  "__model__":"SubAccount",
                  "modified":false,
                  "newModel":false,
                  "constraintsEnabled":false,
                  "data":{
                     "finCoaCd":"EA",
                     "accountNbr":"4769235",
                     "subAcctNbr":"PI",
                     "lastUpdtTs":"2009-07-01T04:00:00.000Z",
```

```
                  "objId":"35A5641EB26D91B6E043814FD88191B6",
                  "subAcctActvCd":true,
                  "subAcctNm":"PROJECT INCOME",
                  "verNbr":1
               }
            },
            {
               "__model__":"SubAccount",
               "modified":false,
               "newModel":false,
               "constraintsEnabled":false,
               "data":{
                  "finCoaCd":"EA",
                  "accountNbr":"4769235",
                  "subAcctNbr":"PP",
                  "lastUpdtTs":"2009-07-01T04:00:00.000Z",
                  "objId":"35A5641EB28B91B6E043814FD88191B6",
                  "subAcctActvCd":true,
                  "subAcctNm":"PRIVATE PAY",
                  "verNbr":1
               }
            },
            {
               "__model__":"SubAccount",
               "modified":false,
               "newModel":false,
               "constraintsEnabled":false,
               "data":{
                  "finCoaCd":"EA",
                  "accountNbr":"4769235",
                  "subAcctNbr":"TIII",
                  "lastUpdtTs":"2009-07-01T04:00:00.000Z",
                  "objId":"35A5641EB28C91B6E043814FD88191B6",
                  "subAcctActvCd":true,
                  "subAcctNm":"TITLE III",
                  "verNbr":1
               }
            }
         ],
         "indirectCostRecoveryAccounts":[
            {
               "__model__":"IndirectCostRecoveryAccount",
               "modified":false,
               "newModel":false,
               "constraintsEnabled":false,
               "data":{
                  "caIcrAcctGnrtdId":11977,
                  "accountNbr":"4769235",
                  "aclnPct":"100",
                  "dobjMaintCdActvInd":true,
                  "finCoaCd":"EA",
                  "icrFinAcctNbr":"0368187",
                  "icrFinCoaCd":"EA",
                  "objId":"AF204E1BF9989BC4E040007F0100096E",
                  "verNbr":1
               }
            },
            {
               "__model__":"IndirectCostRecoveryAccount",
               "modified":false,
               "newModel":false,
               "constraintsEnabled":false,
               "data":{
                  "caIcrAcctGnrtdId":11977,
                  "accountNbr":"4769235",
                  "aclnPct":"100",
                  "dobjMaintCdActvInd":true,
                  "finCoaCd":"EA",
                  "icrFinAcctNbr":"0368187",
```

```
                    "icrFinCoaCd":"EA",
                    "objId":"AF204E1BF9989BC4E040007F0100096E",
                    "verNbr":1
                 }
              },
              {
                 "__model__":"IndirectCostRecoveryAccount",
                 "modified":false,
                 "newModel":false,
                 "constraintsEnabled":false,
                 "data":{
                    "caIcrAcctGnrtdId":11977,
                    "accountNbr":"4769235",
                    "aclnPct":"100",
                    "dobjMaintCdActvInd":true,
                    "finCoaCd":"EA",
                    "icrFinAcctNbr":"0368187",
                    "icrFinCoaCd":"EA",
                    "objId":"AF204E1BF9989BC4E040007F0100096E",
                    "verNbr":1
                 }
              },
              {
                 "__model__":"IndirectCostRecoveryAccount",
                 "modified":false,
                 "newModel":false,
                 "constraintsEnabled":false,
                 "data":{
                    "caIcrAcctGnrtdId":11977,
                    "accountNbr":"4769235",
                    "aclnPct":"100",
                    "dobjMaintCdActvInd":true,
                    "finCoaCd":"EA",
                    "icrFinAcctNbr":"0368187",
                    "icrFinCoaCd":"EA",
                    "objId":"AF204E1BF9989BC4E040007F0100096E",
                    "verNbr":1
                 }
              }
           ],
           "finCoaCd":"EA",
           "accountNbr":"4769235",
           "accountNm":"NON-GENERAL FUND ACCOUNT",
           "acctCityNm":"RICHMOND",
           "acctClosedInd":false,
           "acctCreateDt":"2005-06-07T19:43:48.000Z",
           "acctEffectDt":"2006-07-01T04:00:00.000Z",
           "acctExpirationDt":"2007-06-30T04:00:00.000Z",
           "acctFrngBnftCd":true,
           "acctFscOfcUid":"4160100413",
           "acctIcrTypCd":"10",
           "acctMgrUnvlId":"6274603810",
           "acctOffCmpInd":false,
           "acctPhysCmpCd":"EA",
           "acctPndSfCd":false,
           "acctRstrcStatCd":"R",
           "acctSfCd":"N",
           "acctSpvsrUnvlId":"3474206166",
           "acctStateCd":"IN",
           "acctStreetAddr":"2325 CHESTER BLVD",
           "acctTypCd":"NA",
           "acctZipCd":"47374",
           "bdgtRecLvlCd":"A",
           "cgCfdaNbr":"93.045",
           "contAccountNbr":"4769249",
           "contFinCoaCd":"EA",
           "finExtEncSfCd":true,
           "finHghEdFuncCd":"CSSR",
           "finIntEncSfCd":true,
```

```
            "finObjPrsctrlCd":false,
            "finPreEncSfCd":true,
            "finSeriesId":"000",
            "icrAccountNbr":"0368187",
            "icrFinCoaCd":"EA",
            "incomeAccountNbr":"4769235",
            "incomeFinCoaCd":"EA",
            "lastUpdtTs":"2009-07-01T04:00:00.000Z",
            "lbrBenRtCatCd":"--",
            "objId":"35A5641EACBF91B6E043814FD88191B6",
            "orgCd":"AAOA",
            "subFundGrpCd":"STATEJ",
            "verNbr":2
         }
      },
      "finCoaCd":"EA",
      "accountNbr":"4769235",
      "subAcctNbr":"TIII",
      "lastUpdtTs":"2009-07-01T04:00:00.000Z",
      "objId":"35A5641EB28C91B6E043814FD88191B6",
      "subAcctActvCd":true,
      "subAcctNm":"TITLE III",
      "verNbr":1
   }
}
```


Object Based Database Operations

Repository data access is designed to use object-based queries. An example of a typical query is shown below:

```
select Account o where from Account
where o.acctClosedInd = 'N'
order by o.finCoaCd, o.accountNbr
```

The "o." prefix shown above is significant and is required in the OQL. Field names can be specified using dot notation to designate child objects, for example:

```
select Account o where from Account
where o.acctClosedInd = 'N' and o.subAccounts.subAcctNm = 'someName'
order by o.finCoaCd, o.accountNbr
```

Predefined SQL operations can be added and used by the repository by overriding the Repository.loadNamedDbOperations() method and adding the custom operations as shown below:

```
loadNamedDbOperations() {
        getNamedDbOperations().add("findByName",
                  "select Account o from Account where o.accountNm = :accountNm');
 }
```

You can now execute the query using the method Repository. executeNamedDbOperation() as shown below:

```
let params = new Array();
params.push('myAccountName')
let repo =  orm.getRepository('Account');
let result = await repo.executeNamedDbOperation('findByName', params);
```

Provided Repository Object Operations

The base class Repository.js provides the database operations described below out of the box. In all cases, if an error occurs a JSON object of the form {error: <error_information> } will be returned.

findOne

The findOne method returns a model object by primary key – the signature is shown below:

```
async findOne(primaryKey, options)
```

- primaryKey – required array of the primary key values for bind parameters. The order is important. Values should be in the order of the field definitions in the metadata.
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

find

Returns an array of model objects matching the criteria passed to the method

```
async find(whereComparisons, orderByEntries, options)
```

- whereComparisons – required array of WhereComparison.js objects defining the where clause
- orderByEntries – optional array of OrderByEntry.js defining the order by clause, if not included the query will be order by primary key values
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

exists

Returns true if input parameter exists in database, false otherwise

```
async exists(modelInstance, options)
```

- modelInstance – required - model to check
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

count

Returns count of objects matching input criteria. If no input where clause return table row count

```
async count(whereComparisons, options)
```

- whereComparisons – optional array of WhereComparison.js objects defining the where clause
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

save

Updates or inserts new model records to the database. The entire object graph will be persisted based on the metadata relationship configuration (cascadeUpdate).

```
async save(modelInstances, options)
```

- modelInstances – required – one or more model instances to save
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

delete

Deletes data from the database matching model instances passed in the input. The entire object graph will be processed based on the metadata relationship configuration (cascadeDelete).

```
async delete(modelInstances, options)
```

- modelInstances – required – one or more model instances to delete
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

Provided Repository SQL Operations

The base class Repository.js supports standard SQL operations described below. In all cases, if an error occurs a JSON object of the form {error: <error_information>} will be returned.

executeSqlQuery

Executes standard SQL query and returns result in row/column format

```
async executeSqlQuery(sql, parameters, options)
```

- sql – required SQL select statement
- parameters – optional array of bind parameter values
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

executeSql

Executes a non-select SQL statement

```
async executeSql(sql, parameters, options)
```

- sql – required SQL statement
- parameters – optional array of bind parameter values
- options – optional parameter that specifies additional information – see the next section for more information. If no options parameter is passed a default will be created and used.

Repository Operation options Parameter

As described above, all base SQL operations can take an “options” argument. This parameter is a basic JSON object that adds additional parameters to the standard oracledb options. The oracledb options are listed below and you can find a detailed description here https://github.com/oracle/node-oracledb/blob/master/doc/api.md#executeoptions

- autoCommit – defaults to false
- extendedMetaData – defaults to false

- fetchArraySize
- fetchInfo
- maxRows
- outFormat
- prefetchRows
- resultSet

The custom options are all optional and are described below:

- distinct – true/false – if set to true on a query will add “distinct” to the select clause
- conn – by default all SQL operations pull a connection from the configured connection pool which will rollback on close if no commit is executed. To handle multi-operation transactional processing, you can pull a connection using the orm.getConnection(poolAlias) method and pass this connection to all operations via the options parameter. If a connection exists in the incoming options parameter, the methods described above will use this connection and will not issue a close().
- joinDepth – by default all the SQL operations construct SQL based on the maxDefaultJoinDepth value from the appconfig.json. If you want to override this setting for an individual call you can set the desired joinDepth in the options parameter.
- returnValues – true/false – by default the save method only returns a “rowsAffected” count. If you would like to have the actual updated records returned set this to true. The results will be returned in result.updatedValues.

WhereComparison.js

The Where Comparison module defines where clause comparison information. An array of Where Comparison objects will be passed to repository methods to generate the desired where clause. A portion of the source is shown below:

```
class WhereComparison {
    constructor(fieldName, comparisonValue, comparisonOperator, logicalOperator, useBindParams) {
        this.fieldName = fieldName;
        this.comparisonValue = comparisonValue;
        this.comparisonOperator = comparisonOperator;
        this.openParen = '';
        this.closeParen = '';
        if (util.isDefined(logicalOperator)) {
            this.logicalOperator = logicalOperator;
        } else {
            this.logicalOperator = util.AND;
        }

        if (util.isDefined(useBindParams)) {
            this.useBindParams = useBindParams;
        } else {
            this.useBindParams = true;
        }

        if (this.isUnaryOperator()) {
            this.useBindParams = false;
            comparisonValue = '';
        }

        if (util.IN === comparisonOperator) {
            this.useBindParams = false;
        }
    }
```

Below are the field descriptions:

- fieldName – object field name of the field that this comparison applies to. Dot notation can be used for child objects, for example on the Account objects subAccounts.subAcctNm.
- comparisonValue – this is the value to compare. By default, the generated where clause will use bind parameters and pass the comparison values in a parameter list
- comparisonOperator – supports standard SQL comparisons: =, >=, <= <>, is null, is not null, like and in. The util module includes constants for these values:
  const EQUAL_TO = '=';
  const GREATER_THAN = '>';
  const LESS_THAN = '<';
  const LEES_THAN_OR_EQ = '<=';
  const GREATER_THAN_OR_EQ = '>';
  const NOT_EQUAL = '<>';
  const LIKE = 'like';
  const IN = 'in';
  const NOT_NULL = 'is not null';
  const NULL = 'is null';
- logicalOperator – and/or – defaults to and – the util module includes constants for these values
  const AND = 'and';
  const OR = 'or';
- openParen and closeParen – these allow one or more parenthesis to be added at the beginning or end of the generated comparison.
- useBindParams – true/false – defaults to true except in the case where the comparisonOperator is “in”. The in clause is handled in special manner. It is expected that the comparison value will be an array of values and the bind parameter flag will be ignored for the in clause.

OrderByEntry.js

The OrderByEntry module is used to generate the order by clause. An array of these values will be passed to the repository method to generate the order by clause. A portion of the source is shown below:

```
class OrderByEntry {
    constructor(fieldName, descending) {
        this.fieldName = fieldName;
        this.descending = descending;
    }

    getFieldName() {
        return this.fieldName;
    }

    isDescending() {
        return this.descending;
    }
}
```

Below are the field descriptions

- fieldName – object field name of the field that this comparison applies to. Dot notation can be used for child objects, for example on the Account objects subAccounts.subAcctNm.
- descending – true/false – defaults to false

Database Configuration

Datasource and connection pool initialization logic is provided in the class dbConfiguration.js.

```
"use strict";
const oracledb = require('oracledb');
const util = require("../main/util.js");
const fs = require('fs');

module.exports = function(poolCreatedEmitter, appConfiguration) {
    if (testMode) {
        initPool(appConfiguration.testDbConfiguration,
            poolCreatedEmitter);
    } else {
        initPool(appConfiguration.dbConfiguration,\
            poolCreatedEmitter);
    }
};

async function initPool(securityPath, poolCreatedEmitter) {
    util.logInfo("creating connection pools...");
    let pdefs = JSON.parse(fs.readFileSync(securityPath));

    for (let i = 0; i < pdefs.pools.length; ++i) {
        await oracledb.createPool(pdefs.pools[i]).then(function(pool) {
            util.logInfo("    " + pool.poolAlias
                + " connection pool created");
        });
    }

    // tell orm init that pools are created
    poolCreatedEmitter.emit("poolscreated");

}
```

Multiple connection pools can be created and used by the application. Each connection pool is defined by a pool alias and each repository object has an assigned pool alias. The database connection parameters are found outside the application in a JSON file as described in the Application Configuration section of this document.

Getting Started

Below are the steps to setup Node.js for using simplenodeorm:

1. Ensure Node.js version 10.5 or greater installed
2. Ensure the node-oracledb driver is installed and setup correctly – see https://github.com/oracle/node-oracledb/blob/master/INSTALL.md
3. Install simplenodeorm via npm: npm -install simplenodeorm
4. Create database connection json files ad described in this document
5. Create required custom model, metadata and repository objects for the desired database.
6. Create appconfig.json and set the application properties discussed in this document.

Some Example Code

The code below shows some example calls using a model with name “Account”:

findOne

```
let repo = orm.getRepositry('Account');
let params = new Array();
params.push('MS');
params.push('MYACCOUNTNBR');

let res = repo.findOne(params);
if (util.isDefined(res.error)) {
        // handle error
} else {
        return res.result;
}
```

find

```
let repo = orm.getRepositry('Account');
let whereComparisons = new Array();

whereComparisons.push(require('../main/WhereComparison.js')('finsCoaCd', 'MS', util.EQUAL_TO));

whereComparisons.push(require('../main/WhereComparison.js')('acctClosedInd', 'Y',
util.EQUAL_TO));

let orderByEntries = new Array();

orderByEntries.push(require('../main/OrderByEntry.js')('accountNbr');



let res = repo.find(whereComparions, orderByEntries);
if (util.isDefined(res.error)) {
        // handle error
} else {
        return res.result;
}
```

count

```
let repo = orm.getRepositry('Account');
let whereComparisons = new Array();

whereComparisons.push(require('../main/WhereComparison.js')('finsCoaCd', 'MS', util.EQUAL_TO));

whereComparisons.push(require('../main/WhereComparison.js')('acctClosedInd', 'Y',
util.EQUAL_TO));


let res = repo.count(whereComparions);
if (util.isDefined(res.error)) {
        // handle error
} else {
        return res.result;
}
```

exists

```
let repo = orm.getRepositry('Account');

let model = orm.newModelInstance(orm.getMetaData('Account');
model.setFinCoaCd('MS');
model.setAccountNbr('1234567');

let res = repo.exists(model);
```

```
if (util.isDefined(res.error)) {
        // handle error
} else {
        return res.result;
}
```

getAll

let res = orm.getRepositry('Account').getAll();

```
if (util.isDefined(res.error)) {
        // handle error
} else {
        return res.result;
}
```

save

```
let repo = orm.getRepositry('Account');
let conn = orm.getConnection(repo.getAlias());


let params = new Array();
params.push('MS');
params.push('MYACCOUNTNBR');


let model = repo.findOne(params, {conn: conn});

model.setAcctClosedInd('Y');

let res = repo.save(model, {conn: conn});

if (util.isDefined(res.error)) {
    conn.rollback();
        // handle error
} else {
    conn.commit():
    return res.result;
}
```

conn.close();

delete

```
let repo = orm.getRepositry('Account');
let conn = orm.getConnection(repo.getAlias());

let params = new Array();
params.push('MS');
params.push('MYACCOUNTNBR');


let model = repo.findOne(params, {conn: conn});

model.setAcctClosedInd('Y');

let res = repo.delete(model, {conn: conn});

if (util.isDefined(res.error)) {
    conn.rollback();
        // handle error
} else {
    conn.commit():
    return res.result;
```

```
}

conn.close();
```

executeSqlQuery

```
let repo = orm.getRepositry('Account');
repo.executeSqlQuery('select ACCOUNT_NM from CA_ACCOUNT_T where ACCT_CLOSED_IND = 'Y' order by
1');

if (util.isDefined(res.error)) {
        // handle error
} else {
        return res.result;
}
```

executeSql

```
let repo = orm.getRepositry('Account');
let conn = orm.getConnection(repo.getAlias());
repo.executeSql('update CA_ACCOUNT_T set ACCT_CLOSED_IND = 'N' where ACCOUNT_CLOSED_IND = 'Y', [],
{conn: conn});
if (util.isDefined(res.error)) {
    conn.rollback();
    // handle error

} else {
    conn.commit();
    return res.result;
}
conn.close();
```

REST Access

Basic REST access is available with the application using the Node express package. To run rest server on startup set the startRestServer=true in appconfig.json. The appconfig.json contains some required information that must be populated to use the REST server:

```
"restUrlBase" : "/orm",
"restPort" : 8888,
```

You can see the startup in orm.js below:

```
var poolCreatedEmitter = new events.EventEmitter();
poolCreatedEmitter.on("poolscreated", async function() {
    loadOrm();
    if (appConfiguration.testMode) {
        let suite = require("./test/testSuite.js");
        await suite.run();
    }

    if (appConfiguration.startRestServer) {
        startRest();
    }
});
```

A code snippet of the server code is shown below:

```
server.get(REST_URL_BASE + '/:module/:method', async function(req, res) {
```

```
    let repo = repositoryMap.get(req.params.module);
    let md = repo.getMetaData(req.params.module);
    if (util.isUndefined(repo)) {
        // support for using an alias for long module names
        if (util.isDefined(appConfiguration.aliases[req.params.module])) {
            repo = repositoryMap.get(appConfiguration.aliases[req.params.module]);
            md = repo.getMetaData(appConfiguration.aliases[req.params.module]);
        }
    }

    let params = new Array();
    let pk = md.getPrimaryKeyFields();
    let fields = md.getFields();

    if (util.isUndefined(repo) || util.isUndefined(md)) {
        res.status(400).send('invalid module \'' + req.params.module + '\' specified');
    } else {
        var result;
        switch(req.params.method.toLowerCase()) {
            case util.FIND_ONE.toLowerCase():
                for (let i = 0; i < pk.length; ++i) {
                    params.push(req.query[pk[i].fieldName]);
                }
                result = await repo.findOne(params);
                break;
            case util.FIND.toLowerCase():
                for (let i = 0; i < fields.length; ++i) {
                    if (util.isDefined(req.query[fields[i].fieldName])) {
                        params.push(require('./main/WhereComparison.js')(fields[i].fieldName,
                            req.query[fields[i].fieldName], util.EQUAL_TO));
                    }
                }
                result = await repo.find(params);
                break;
            case util.COUNT.toLowerCase():
                for (let i = 0; i < fields.length; ++i) {
                    if (util.isDefined(req.query[fields[i].fieldName])) {
                        params.push(require('./main/WhereComparison.js')(fields[i].fieldName,
                            req.query[fields[i].fieldName], util.EQUAL_TO));
                    }
                }
                result = await repo.count(params);
                break;
            case util.EXISTS.toLowerCase():
                for (let i = 0; i < pk.length; ++i) {
                    params.push(req.query[pk[i].fieldName]);
                }
                result = await repo.exists(params);
                break;
            default:
                res.status(400).send('invalid method \'' + req.params.method + '\' specified');
                break;
        }

        if (util.isUndefined(result)) {
            res.status(404).send('not found');
        } else if (util.isDefined(result.error)) {
            res.status(500).send(util.toString(result.error));
        } else if (util.isDefined(result.result)) {
            res.status(200).send(util.toDataTransferString(result.result));
        } else {
            res.status(200).send(result);
        }
    }

    res.end();
});

server.post(REST_URL_BASE + '/:module/:method', async function(req, res) {
```

```
    let repo = repositoryMap.get(req.params.module);
    let md = repo.getMetaData(req.params.module);
    if (util.isUndefined(repo)) {
        // support for using an alias for long module names
        if (util.isDefined(appConfiguration.aliases[req.params.module])) {
            repo = repositoryMap.get(appConfiguration.aliases[req.params.module]);
            md = repo.getMetaData(appConfiguration.aliases[req.params.module]);
        }
    }

    if (util.isUndefined(repo) || util.isUndefined(md)) {
        res.status(400).send('invalid module \'' + req.params.module + '\' specified');
    } else {
        let result;

        switch(req.params.method.toLowerCase()) {
            case util.FIND_ONE.toLowerCase():
                result = await repo.findOne(req.body.primaryKeyValues);
                break;
            case util.FIND.toLowerCase():
                result = await repo.find(populateWhereFromRequestInput(req.body.whereComparisons),

               populateOrderByFromRequestInput(req.body.orderByEntries),
                   populateOptionsFromRequestInput(req.body.options));
               break;
            case util.SAVE.toLowerCase():
                result = repo.save(populateModelObjectsFromRequestInput(req.body.modelInstances),
                   req.body.options);
                break;
            default:
                res.status(400).send('invalid method \'' + req.params.method + '\' specified');
                break;
        }

        if (util.isUndefined(result)) {
            res.status(404).send('not found');
        } else if (util.isDefined(result.error)) {
            res.status(500).send(util.toString(result.error));
        } else if (util.isDefined(result.result)) {
            res.status(200).send(util.toDataTransferString(result.result));
        } else if (util.isDefined(result.updatedValues)) {
            res.status(200).send(util.toDataTransferString(result.updatedValues));
        } else if (util.isDefined(result.rowsAffected)) {
            res.status(200).send(util.toDataTransferString(result));
        } else {
            res.status(200).send(result);
        }
    }

    res.end();
});

server.put(REST_URL_BASE + '/:module/:method', function(req, res) {
    let repo = repositoryMap.get(req.params.module);
    let md = repo.getMetaData(req.params.module);
    if (util.isUndefined(repo)) {
        // support for using an alias for long module names
        if (util.isDefined(appConfiguration.aliases[req.params.module])) {
            repo = repositoryMap.get(appConfiguration.aliases[req.params.module]);
            md = repo.getMetaData(appConfiguration.aliases[req.params.module]);
        }
    }

    if (util.isUndefined(repo) || util.isUndefined(md)) {
        res.status(400).send('invalid module \'' + req.params.module + '\' specified');
    } else {
        let f;
        switch(req.params.method.toLowerCase()) {
            case util.SAVE.toLowerCase():
```

```
                result = repo.save(populateModelObjectsFromRequestInput(req.body.modelInstances),
                    populateOptionsFromRequestInput(req.body.options));
                break;
            default:
                res.status(400).send('invalid method \'' + req.params.method + '\' specified');
                break;
        }

        if (util.isUndefined(result)) {
            res.status(404).send('not found');
        } else if (util.isDefined(result.error)) {
            res.status(500).send(util.toString(result.error));
        } else if (util.isDefined(result.updatedValues)) {
            res.status(200).send(util.toDataTransferString(result.updatedValues));
        } else if (util.isDefined(result.rowsAffected)) {
            res.status(200).send(util.toDataTransferString(result));
        } else {
            res.status(200).send(result);
        }
    }

    res.end();
});

server.delete(REST_URL_BASE + '/:module/:method', function(req, res) {
    let repo = repositoryMap.get(req.params.module);
    let md = repo.getMetaData(req.params.module);
    if (util.isUndefined(repo)) {
        // support for using an alias for long module names
        if (util.isDefined(appConfiguration.aliases[req.params.module])) {
            repo = repositoryMap.get(appConfiguration.aliases[req.params.module]);
            md = repo.getMetaData(appConfiguration.aliases[req.params.module]);
        }
    }

    if (util.isUndefined(repo) || util.isUndefined(md)) {
        res.status(400).send('invalid module \'' + req.params.module + '\' specified');
    } else {
        switch(req.params.method.toLowerCase()) {
            case util.DELETE.toLowerCase():
                result =
                    repo.delete(populateModelObjectsFromRequestInput(req.body.modelInstances),
                        populateOptionsFromRequestInput(req.body.options));
                break;
            default:
                res.status(400).send('invalid method \'' + req.params.method + '\' specified');
                break;
        }

        if (util.isDefined(result.error)) {
            res.status(500).send(util.toString(result.error));
        } else if (util.isDefined(result.rowsAffected)) {
            res.status(200).send(util.toDataTransferString(result));
        } else {
            res.status(200).send(result);
        }
    }

    res.end();
});
}
```

As you can see this API supports the base calls supported by the repository – findOne, find, exists, count, save and delete. The URL used is something like:

http://localhost:<restPort>/<restUrlBase>/<model-name-

lowercase>/<operation>?param1=MS&param2=RA123456

for example:
http://localhost:8888/orm/account/findOne?finCoaCd=XX&accountNbr=12345678

where “account” is the ORM object name (all lowercase) and “findOne” is the method name (not case sensitive). Simple GET calls for findOne, find, exists and count are supported as shown. It is assumed that the where clause will be all ands. More complex queries as well as save and delete are supported by POST, DELETE and PUT. In the POST query you can pass and array of WhereComparison objects and OrderByEntry objects for a more complex query definition.