Tender Management System - Sequelize ORM Backend

Executive Summary

A production-ready Node.js/Express backend using **Sequelize ORM** instead of raw SQL queries. Sequelize provides an elegant, object-oriented approach to database operations with automatic table management, built-in validations, and powerful query capabilities.

Sequelize vs Raw SQL Comparison

Raw SQL Approach (Previous)

```
// Manual query construction
const query = `
  INSERT INTO lost_domestic_leads (...)
  VALUES ($1, $2, $3, ...)
  RETURNING *;
`;
const result = await pool.query(query, values);
```

Sequelize ORM Approach (New)

```
// Object-oriented, type-safe
const lead = await LostDomesticLead.create({
  serialNumber: 'DL-001',
  tenderName: 'Test Tender',
  customer: 'ABC Corp'
});
```

Key Advantages

✅ **No SQL Strings** - Write queries using JavaScript objects
✅ **Built-in Validations** - Enforce constraints at model level
✅ **Automatic Type Casting** - Proper data type handling
✅ **Query Builder** - Chainable, readable syntax
✅ **Migrations** - Version control for database schema
✅ **Hooks** - Lifecycle methods for data transformation
✅ **Relationships** - Easy model associations
✅ **Transactions** - ACID compliance support
✅ **Less SQL Injection Risk** - Parameterized queries by default
✅ **Better Error Handling** - ValidationError with detailed messages

Installation & Setup

Quick Start

```
# 1. Create project
mkdir tender-backend-sequelize
cd tender-backend-sequelize
npm init -y

# 2. Install dependencies
npm install express sequelize pg pg-hstore dotenv cors helmet morgan uuid
npm install --save-dev sequelize-cli nodemon

# 3. Initialize Sequelize
npx sequelize-cli init

# 4. Configure environment
cp .env.example .env
# Edit .env with credentials

# 5. Create database
psql -U postgres -c "CREATE DATABASE tender_management;"

# 6. Start development
npm run dev
```

Project Structure

```
tender-backend-sequelize/
├── config/
│   ├── database.js              # Sequelize configuration
│   └── config.json              # Sequelize CLI config
├── models/
│   ├── index.js                 # Model initialization
│   ├── LostDomesticLead.js      # Model definition
│   ├── DomesticOrder.js
│   ├── BudgetaryQuotation.js
│   ├── LeadSubmitted.js
│   ├── DomesticLeadV2.js
│   ├── ExportLead.js
│   └── CRMLead.js
├── controllers/
│   ├── lostDomesticLeadsController.js
│   ├── domesticOrderController.js
│   ├── budgetaryQuotationController.js
│   ├── leadSubmittedController.js
│   ├── domesticLeadsV2Controller.js
│   ├── exportLeadsController.js
│   └── crmLeadsController.js
├── routes/
│   ├── lostDomesticLeads.js
│   ├── domesticOrder.js
│   ├── budgetaryQuotation.js
│   ├── leadSubmitted.js
│   ├── domesticLeadsV2.js
│   ├── exportLeads.js
│   └── crmLeads.js
├── migrations/
│   ├── [timestamp]-create-lost-domestic-leads.js
│   ├── [timestamp]-create-domestic-order.js
│   └── ... (one per form)
├── seeders/
│   └── [timestamp]-demo-data.js
├── middleware/
│   ├── errorHandler.js
│   └── validation.js
├── server.js
├── .env
├── .env.example
├── .sequelizerc
└── package.json
```

Model Definition Example

```
// models/LostDomesticLead.js
const { DataTypes } = require('sequelize');

module.exports = (sequelize) =&gt; {
  const LostDomesticLead = sequelize.define('LostDomesticLead', {
    id: {
      type: DataTypes.UUID,
      defaultValue: DataTypes.UUIDV4,
      primaryKey: true
    },
    serialNumber: {
      type: DataTypes.STRING(255),
      allowNull: false,
      unique: true,
      validate: {
        notEmpty: { msg: 'Serial number cannot be empty' },
        len: { args: [1, 255], msg: 'Serial number length' }
      }
    },
    tenderName: {
      type: DataTypes.STRING(255),
      allowNull: false,
      validate: {
        notEmpty: { msg: 'Tender name is required' }
      }
    },
    customer: {
      type: DataTypes.STRING(255),
      allowNull: false
    },
    valueWithoutGst: {
      type: DataTypes.DECIMAL(15, 2),
      validate: {
        isDecimal: true,
        min: 0
      }
    },
    competitors: {
      type: DataTypes.JSONB,
      defaultValue: []
    },
    // ... more fields
  }, {
    tableName: 'lost_domestic_leads',
    timestamps: true
  });

  return LostDomesticLead;
};
```

Controller Implementation

Create Operation

```
exports.create = async (req, res, next) =&gt; {
  try {
    const lead = await LostDomesticLead.create(req.body);
    res.status(201).json({
      success: true,
      message: 'Lead created',
      data: lead
    });
  } catch (error) {
    if (error instanceof ValidationError) {
      return res.status(400).json({
        success: false,
        errors: error.errors.map(e =&gt; ({
          field: e.path,
          message: e.message
        }))
      });
    }
    next(error);
  }
};
```

Read Operations

```
// Get all with pagination
const { count, rows } = await LostDomesticLead.findAndCountAll({
  limit: 20,
  offset: 0,
  order: [['createdAt', 'DESC']]
});

// Find by primary key
```

```
const lead = await LostDomesticLead.findByPk(id);

// Find by custom condition
const lead = await LostDomesticLead.findOne({
  where: { serialNumber: 'DL-001' }
});

// Complex queries
const leads = await LostDomesticLead.findAll({
  where: {
    customer: 'ABC Corp',
    year: 2025
  },
  limit: 10,
  order: [['submittedAt', 'DESC']]
});
```

Update Operation

```
// Method 1: Update instance and save
const lead = await LostDomesticLead.findByPk(id);
lead.tenderName = 'New Name';
await lead.save();

// Method 2: Direct update
await LostDomesticLead.update(
  { tenderName: 'New Name' },
  { where: { id } }
);
```

Delete Operation

```
await LostDomesticLead.destroy({
  where: { id }
});
```

Sequelize Query Methods

Method	Purpose	Example
`create()`	Insert single record	`Model.create(data)`
`bulkCreate()`	Insert multiple	`Model.bulkCreate(arrayData)`
`findAll()`	Get multiple records	`Model.findAll({ limit: 10 })`
`findOne()`	Get single record	`Model.findOne({ where: {...} })`
`findByPk()`	Get by ID	`Model.findByPk(id)`
`findAndCountAll()`	Get with count	`Model.findAndCountAll()`
`update()`	Modify records	`Model.update(data, { where })`

Method	Purpose	Example
`destroy()`	Delete records	`Model.destroy({ where })`
`count()`	Get count	`Model.count()`
`sum()`	Sum values	`Model.sum('value')`
`increment()`	Increment value	`Model.increment('count')`
`decrement()`	Decrement value	`Model.decrement('count')`

Migrations: Version Control for Database

Create Migration

```
npx sequelize-cli migration:create --name create-lost-domestic-leads
```

Migration File

```
module.exports = {
  up: async (queryInterface, Sequelize) =&gt; {
    await queryInterface.createTable('lost_domestic_leads', {
      id: {
        type: Sequelize.UUID,
        primaryKey: true
      },
      serial_number: {
        type: Sequelize.STRING,
        unique: true
      },
      // ... all columns
    });
  },
  down: async (queryInterface) =&gt; {
    await queryInterface.dropTable('lost_domestic_leads');
  }
};
```

Run Migrations

```
npm run migrate           # Run all pending
npm run migrate:undo     # Undo last migration
npm run migrate:undo:all # Undo all migrations
```

Validations: Built-in Data Integrity

Model-level Validations

```
email: {
  type: DataTypes.STRING,
  validate: {
    isEmail: { msg: 'Must be valid email' },
    len: { args: [1, 255] }
  }
},
value: {
  type: DataTypes.DECIMAL(15, 2),
  validate: {
    min: { args: [0], msg: 'Must be positive' },
    max: { args: [999999999.99] }
  }
},
status: {
  type: DataTypes.ENUM('active', 'inactive'),
  validate: {
    isIn: {
      args: [['active', 'inactive']],
      msg: 'Invalid status'
    }
  }
}
```

Hooks: Lifecycle Methods

Auto-execute functions at key points

```
// Before create
LostDomesticLead.beforeCreate((lead) =&gt; {
  lead.createdAt = new Date();
});

// After create
LostDomesticLead.afterCreate((lead) =&gt; {
  console.log('Lead created:', lead.id);
});

// Before update
LostDomesticLead.beforeUpdate((lead) =&gt; {
  lead.updatedAt = new Date();
});

// Before destroy
LostDomesticLead.beforeDestroy(async (lead) =&gt; {
  // Log deletion
  console.log('Deleting lead:', lead.id);
});
```

Scopes: Reusable Query Filters

Define scopes

```
LostDomesticLead.addScope('recent', {
  order: [['createdAt', 'DESC']],
  limit: 50
});

LostDomesticLead.addScope('byCustomer', (customer) =&gt; ({
  where: { customer }
}));
```

Use scopes

```
// Get 50 most recent
const leads = await LostDomesticLead.scope('recent').findAll();

// Get for specific customer
const customerLeads = await LostDomesticLead
  .scope(['recent', { method: ['byCustomer', 'ABC Corp'] }])
  .findAll();
```

Transactions: ACID Compliance

Multi-step operations with rollback

```
const t = await sequelize.transaction();
try {
  await LostDomesticLead.create(data1, { transaction: t });
  await DomesticOrder.create(data2, { transaction: t });
  await t.commit();
} catch (error) {
  await t.rollback();
  throw error;
}
```

API Endpoints (Enhanced)

Lost Domestic Leads

```
POST   /api/lost-domestic-leads           # Create
GET    /api/lost-domestic-leads           # Get all (paginated)
GET    /api/lost-domestic-leads/:id       # Get by ID
GET    /api/lost-domestic-leads/serial/:serialNumber  # Get by serial
GET    /api/lost-domestic-leads/search    # Search by customer
GET    /api/lost-domestic-leads/stats     # Get statistics
PUT    /api/lost-domestic-leads/:id       # Update
DELETE /api/lost-domestic-leads/:id       # Delete
```

Plus 6 more similar endpoint groups (one per form type).

Response Format

Success

```
{
  "success": true,
  "message": "Operation successful",
  "data": { /* record data */ },
  "pagination": {
    "total": 100,
    "page": 1,
    "limit": 20,
    "pages": 5
  },
  "timestamp": "2025-11-27T22:54:00Z"
}
```

Validation Error

```
{
  "success": false,
  "message": "Validation error",
  "errors": [
    {
      "field": "serialNumber",
      "message": "Serial number cannot be empty"
    }
  ],
  "timestamp": "2025-11-27T22:54:00Z"
}
```

Development Commands

```
npm run dev              # Start with auto-reload
npm start               # Production start
npm run migrate         # Run migrations
npm run migrate:create  # Create new migration
npm run seed           # Run seeders
npm run seed:create    # Create new seeder
npm run test           # Run tests
```

Testing Example

```
const request = require('supertest');
const app = require('../server');

describe('Lost Domestic Leads API', () =&gt; {
  test('POST creates lead with valid data', async () =&gt; {
    const res = await request(app)
      .post('/api/lost-domestic-leads')
      .send({
        serialNumber: 'TEST-001',
        tenderName: 'Test Tender',
        customer: 'Test Company'
      });

    expect(res.status).toBe(201);
    expect(res.body.success).toBe(true);
    expect(res.body.data.id).toBeDefined();
  });

  test('POST fails with missing required fields', async () =&gt; {
    const res = await request(app)
      .post('/api/lost-domestic-leads')
      .send({ serialNumber: 'TEST-002' });

    expect(res.status).toBe(400);
    expect(res.body.success).toBe(false);
  });
});
```

Performance Optimization

- **Connection Pooling** - Configured in database.js
- **Indexes** - Defined in migrations
- **Scopes** - Avoid N+1 queries
- **Raw Queries** - For complex queries when needed
- **Pagination** - Always paginate large datasets

Production Deployment

Pre-deployment

1. Run all migrations
2. Test all endpoints
3. Set NODE_ENV=production
4. Configure connection pooling
5. Enable logging
6. Set up monitoring

Environment

```
NODE_ENV=production
PORT=5000
DB_HOST=production-db.example.com
DB_NAME=tender_production
DB_USER=prod_user
DB_PASSWORD=strong_password_here
```

Troubleshooting

Issue	Solution
Connection fails	Check .env credentials, verify PostgreSQL running
Migration error	Check timestamps in migration files, run undo
Validation fails silently	Add `validate: { msg: 'Custom message' }`
N+1 query problem	Use eager loading or scopes
Transaction errors	Ensure transaction parameter passed to all operations

Files Provided

1. sequelize-setup-guide.md - Complete Sequelize guide
2. **config-sequelize-database-js.txt** - Database configuration
3. **models-lostdomesticlead-sequelize-js.txt** - Model example
4. **models-index-js.txt** - Model initialization
5. **controllers-lostdomesticlead-sequelize-js.txt** - Controller example
6. **routes-lostdomesticlead-sequelize-js.txt** - Routes example
7. **sequelize-server-js.txt** - Express server with Sequelize
8. **migration-example-js.txt** - Migration template
9. **sequelize-package-json.txt** - Dependencies

10. **This PDF** - Complete documentation

Advantages Over Raw SQL

Aspect	Raw SQL	Sequelize
Queries	String-based	Object-based
Validations	Manual	Built-in
Type Safety	Low	High
Migrations	Manual	Version controlled
Error Handling	Generic	Specific errors
Learning Curve	Easier	Moderate
Scalability	Manual indexing	Built-in options
Code Reuse	Low	High (scopes, hooks)
SQL Injection	Possible	Protected by default

Next Steps

1. ✅ Install Sequelize and dependencies
2. ✅ Configure database connection
3. ✅ Define all 7 models
4. ✅ Create migrations for each form
5. ✅ Implement controllers with ORM
6. ✅ Setup routes
7. ✅ Add validations and hooks
8. ✅ Create seeders for test data
9. ✅ Write comprehensive tests
10. ✅ Deploy to production

Resources

- **Sequelize Documentation:** https://sequelize.org/
- **PostgreSQL Documentation:** https://www.postgresql.org/docs/
- **Express Documentation:** https://expressjs.com/
- **Node.js Best Practices:** https://github.com/goldbergyoni/nodebestpractices

Status: Production-Ready ✅
Version: 1.0.0-sequelize
Architecture: Node.js + Express + Sequelize ORM + PostgreSQL
Date: November 27, 2025