Day 5 — NestJS + Sequelize ORM Integration

Siddhesh Prabhugaonkar
Microsoft Certified Trainer
siddheshpg@azureauthority.in

1. Learning goals

- **Previous Day Recap:**
    - Creating a NestJS module and controller for Employee API
    - Implementing in-memory CRUD operations (GET, POST, PUT, DELETE)
    - Using DTOs (Data Transfer Objects) for request validation
    - Exception handling with built-in filters
    - Folder structure best practices for scalable NestJS apps
    - Bruno testing of CRUD endpoints.
- **Today**
    - Introduction to Sequelize ORM
    - Key Sequelize concepts: models, migrations, associations
    - Connecting Sequelize with Oracle database
    - Setting up Sequelize in a NestJS project
    - Implementing CRUD with NestJS + Sequelize + Oracle DB
    - Testing and verifying API endpoints.

2. Core concepts

2.1 What is Sequelize?

Sequelize is a promise-based Node.js ORM for SQL databases that maps JavaScript/TypeScript classes (models) to database tables. It supports transactions, associations (relations), eager/lazy loading, and migrations. Recent releases include an Oracle dialect that uses the `node-oracledb` driver.

2.2 Key concepts

- **Model** — JS/TS class that maps to a DB table (columns defined as attributes).
- **Migration** — a versioned script to create/alter DB schema (keeps schema reproducible).
- **Association** — relations between models (belongsTo, hasMany, etc.).

- **Dialect/Driver** — the DB backend adapter; for Oracle Sequelize uses `node-oracledb` (you must install it and the Oracle Instant Client on the host).

---

3. Architecture diagram (simple ASCII)

```
[Client] --HTTP--> [NestJS Controllers] --> [NestJS Services] -->
[Sequelize Models]
                                                          \
                                                           -> node-
oracledb -> Oracle DB
```

---

4. Before you start — system prerequisites

- Node.js 18+ (or LTS you use).
- Oracle Database accessible (XE or cloud ADB). If using Oracle Cloud Autonomous DB, prepare Wallet or connection details per Oracle docs. oracle.com
- Oracle Instant Client installed on machine (node-oracledb needs it). oracle.com

---

5. Setup steps (commands + packages)

Open a project folder and initialize:

```
mkdir nest-sequelize-oracle
cd nest-sequelize-oracle
npm init -y
# If you're using TypeScript (recommended), add tsconfig and types:
npm install --save-dev typescript ts-node @types/node
```

Install Nest + Sequelize + Oracle driver + helper libs:

```
# Nest core (you can use Nest CLI if you prefer)
npm install --save @nestjs/common @nestjs/core @nestjs/platform-express
reflect-metadata rxjs

# Sequelize + Nest bridge + types
npm install --save sequelize sequelize-typescript @nestjs/sequelize
npm install --save oracledb         # Oracle driver used by Sequelize for
Oracle
npm install --save-dev @types/sequelize
```

Notes:

- `oracledb` requires Oracle Instant Client and sometimes environment variables (e.g., `LD_LIBRARY_PATH` on Linux / `PATH` on Windows). See Oracle node-oracledb docs.

- Sequelize will use dialect `'oracle'` when configured for Oracle. The Sequelize docs indicate `node-oracledb` is the connector for Oracle. sequelize.org

---

6. NestJS + Sequelize bootstrap (ESM / TypeScript)

Create `src/main.ts`:

```
// src/main.ts (ESM / TS)
import 'reflect-metadata';
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module.js';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.setGlobalPrefix('api');
  await app.listen(3000);
  console.log('Listening on http://localhost:3000/api');
}
bootstrap();
```

Create `src/app.module.ts` — configure Sequelize for Oracle:

```
// src/app.module.ts
import { Module } from '@nestjs/common';
import { SequelizeModule } from '@nestjs/sequelize';
import { EmployeeModule } from './employee/employee.module.js';

@Module({
  imports: [
    SequelizeModule.forRoot({
      dialect: 'oracle',           // important: use 'oracle'
      username: process.env.DB_USER || 'HR',
      password: process.env.DB_PASS || 'hrpass',
      host: process.env.DB_HOST || 'localhost',
      port: Number(process.env.DB_PORT || 1521),
      database: process.env.DB_NAME || 'ORCLCDB', // or your service name
      models: [],                  // we'll add models in their modules
      dialectOptions: {
        // For Oracle Cloud Wallet you may need connectString or TNS_ADMIN
env var
      },
      logging: false,
    }),
    EmployeeModule,
  ],
})
export class AppModule {}
```

Caveat: for Oracle Autonomous DB you may need to set `dialectOptions.connectString` or rely on `TNS_ADMIN` environment variable that points to the wallet. See community discussion and Oracle docs. Stack Overflow+1

7. Employee model (Sequelize + sequelize-typescript, ESM)

Create `src/employee/employee.model.ts`:

```
// src/employee/employee.model.ts
import { Table, Column, Model, DataType, PrimaryKey, AutoIncrement,
AllowNull, ForeignKey, BelongsTo } from 'sequelize-typescript';
import { Department } from '../department/department.model.js';

@Table({
  tableName: 'EMPLOYEES',
  timestamps: true,
})
export class Employee extends Model {
  @PrimaryKey
  @AutoIncrement
  @Column({ type: DataType.INTEGER })
  id;

  @AllowNull(false)
  @Column({ type: DataType.STRING(100) })
  name;

  @Column({ type: DataType.STRING(100) })
  email;

  @Column({ type: DataType.STRING(20) })
  phone;

  @ForeignKey(() => Department)
  @Column({ type: DataType.INTEGER, field: 'DEPARTMENT_ID' })
  departmentId;
}
```

And a minimal `Department` model (we'll use it in the practice activity):

```
// src/department/department.model.ts
import { Table, Column, Model, DataType, PrimaryKey, AutoIncrement,
AllowNull, HasMany } from 'sequelize-typescript';
import { Employee } from '../employee/employee.model.js';

@Table({ tableName: 'DEPARTMENTS' })
export class Department extends Model {
  @PrimaryKey
  @AutoIncrement
  @Column({ type: DataType.INTEGER })
  id;

  @AllowNull(false)
  @Column({ type: DataType.STRING(100) })
  name;

  @HasMany(() => Employee)
  employees;
}
```

Register models with SequelizeModule in modules (see next section).

---

8. Employee module, service, controller (CRUD)

`employee.module.ts`:

```
// src/employee/employee.module.ts
import { Module } from '@nestjs/common';
import { SequelizeModule } from '@nestjs/sequelize';
import { Employee } from './employee.model.js';
import { Department } from '../department/department.model.js';
import { EmployeeService } from './employee.service.js';
import { EmployeeController } from './employee.controller.js';

@Module({
  imports: [SequelizeModule.forFeature([Employee, Department])],
  providers: [EmployeeService],
  controllers: [EmployeeController],
})
export class EmployeeModule {}
```

`employee.service.ts`:

```
// src/employee/employee.service.ts
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import { Employee } from './employee.model.js';

@Injectable()
export class EmployeeService {
  constructor(@InjectModel(Employee) private employeeModel: typeof
Employee) {}

  create(payload) {
    return this.employeeModel.create(payload);
  }

  findAll() {
    return this.employeeModel.findAll({ include: { all: true } });
  }

  async findOne(id) {
    const emp = await this.employeeModel.findByPk(id, { include: { all:
true } });
    if (!emp) throw new NotFoundException('Employee not found');
    return emp;
  }

  async update(id, changes) {
    const emp = await this.findOne(id);
    return emp.update(changes);
  }

  async remove(id) {
    const emp = await this.findOne(id);
    await emp.destroy();
    return { deleted: true };
  }
}
```

employee.controller.ts:

```
// src/employee/employee.controller.ts
import { Controller, Get, Post, Put, Delete, Param, Body } from
'@nestjs/common';
import { EmployeeService } from './employee.service.js';

@Controller('employees')
export class EmployeeController {
  constructor(private svc: EmployeeService) {}

  @Post()
  create(@Body() body) {
    return this.svc.create(body);
  }

  @Get()
  findAll() {
    return this.svc.findAll();
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.svc.findOne(Number(id));
  }

  @Put(':id')
  update(@Param('id') id: string, @Body() body) {
    return this.svc.update(Number(id), body);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.svc.remove(Number(id));
  }
}
```

Notes:

- DTOs and validation (class-validator + class-transformer) are recommended for production — see practice section where we include a sample DTO.

---

9. Migrations (Sequelize CLI) — basic flow

Install CLI:

```
npm install --save-dev sequelize-cli
```

Create `.sequelizerc` in project root to point to folders:

```
// .sequelizerc (ESM or CommonJS fine)
const path = require('path');
module.exports = {
  'config': path.resolve('src', 'config', 'sequelize.config.js'),
  'models-path': path.resolve('src', 'models'),
  'seeders-path': path.resolve('src', 'seeders'),
  'migrations-path': path.resolve('src', 'migrations'),
};
```

Sample migration to create `EMPLOYEES`:

```
// src/migrations/20251110-create-employees.js
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('EMPLOYEES', {
      ID: { type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true
},
      NAME: { type: Sequelize.STRING(100), allowNull: false },
      EMAIL: { type: Sequelize.STRING(100) },
      PHONE: { type: Sequelize.STRING(20) },
      DEPARTMENT_ID: { type: Sequelize.INTEGER, allowNull: true },
      CREATED_AT: { type: Sequelize.DATE },
      UPDATED_AT: { type: Sequelize.DATE },
    });
  },
  down: async (queryInterface) => {
    await queryInterface.dropTable('EMPLOYEES');
  },
};
```

Run migrations:

```
npx sequelize-cli db:migrate
```

Caveat for Oracle: Sequelize's migration layer maps to Oracle via the Oracle QueryInterface. Some migration features (e.g., autoIncrement) and naming/case rules differ in Oracle; test and adapt column definitions (use sequences and triggers if needed). Community and Sequelize docs discuss Oracle specifics. sequelize.org+1

---

10. DTO + Validation example (recommended)

Install packages:

```
npm install --save class-validator class-transformer
```

src/employee/dto/create-employee.dto.ts:

```
// src/employee/dto/create-employee.dto.ts
import { IsString, IsOptional, IsInt } from 'class-validator';

export class CreateEmployeeDto {
  @IsString()
  name;

  @IsOptional()
  @IsString()
  email;

  @IsOptional()
  @IsString()
  phone;

  @IsOptional()
  @IsInt()
  departmentId;
}
```

Then enable global validation in `main.ts` (Nest built-in):

```
import { ValidationPipe } from '@nestjs/common';
// inside bootstrap():
app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true
}));
```

---

11. Testing & verifying API endpoints

Start the Nest app:

```
node --experimental-specifier-resolution=node -r ts-node/register
src/main.ts
# or use ts-node directly in dev
```

Basic curl examples:

- Create employee:

```
curl -X POST http://localhost:3000/api/employees \
  -H "Content-Type: application/json" \
  -d '{"name":"Alice", "email":"alice@example.com", "departmentId": 1}'
```

- Get all:

```
curl http://localhost:3000/api/employees
```

- Get one:

```
curl http://localhost:3000/api/employees/1
```

For testing with Oracle you can also run raw SQL via `node-oracledb` or use SQL Developer to inspect tables. Oracle-specific connection details (wallets, service names) can require `connectString` in `dialectOptions` or setting `TNS_ADMIN`. See the community Q&A and Oracle docs for wallet steps. Stack Overflow+1

---

12. Oracle-specific gotchas & best practices

- **Instant Client required**: `node-oracledb` needs the Oracle Instant Client library installed on the host. oracle.com
- **Uppercase vs quoted identifiers**: Oracle defaults table/column names to uppercase unless quoted — Sequelize may generate quoted lowercase names; you may need to set `quoteIdentifiers: false` or specify `field` explicitly. See community threads. Oracle Forums
- **Auto-increment**: Oracle doesn't use `AUTO_INCREMENT`; use sequences — Sequelize's Oracle dialect handles ID generation, but verify behavior and migrations. sequelize.org
- **Autonomous DB / Wallet**: If connecting to Oracle Cloud ADB, you may need to use the wallet and `TNS_ADMIN` environment variable or `connectString` in `dialectOptions`. Stack Overflow

---

13. Hands-on practice activity

Scenario

You already have Employee CRUD. Extend the app to add a `Department` CRUD API and update Employee CRUD so that employees store `departmentId` and endpoints return department data when fetching employees (i.e., include association). Add validation and a migration.

Tasks

1. **Create Department module & model**
   - Add `src/department/department.module.ts` and `src/department/department.model.ts` (model shown earlier).
   - Register Department module in `AppModule` if separate (or rely on `EmployeeModule` importing model).
2. **Add association in models**
   - In `Employee` model add `@BelongsTo(() => Department) department;`
   - In `Department` model add `@HasMany(() => Employee) employees;`
3. **Create Department service & controller** (CRUD code pattern mirrors Employee — implement `create`, `findAll`, `findOne`, `update`, `remove`).
4. **Migration to create DEPARTMENTS and adjust EMPLOYEES**
   - Create migrations:

    - `create-departments` migration that creates table `DEPARTMENTS`.
    - `alter-employees-add-department` migration that adds `DEPARTMENT_ID` column and a foreign key (Oracle-specific foreign key DDL).

Example migration to add column (simplified):

```
// src/migrations/20251110-add-department-id-to-employees.js
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.addColumn('EMPLOYEES', 'DEPARTMENT_ID', {
      type: Sequelize.INTEGER,
      allowNull: true,
    });
    // Oracle foreign key DDL might require explicit constraint names:
    await queryInterface.addConstraint('EMPLOYEES', {
      fields: ['DEPARTMENT_ID'],
      type: 'foreign key',
      name: 'FK_EMP_DEPT',
      references: { table: 'DEPARTMENTS', field: 'ID' },
      onDelete: 'SET NULL',
      onUpdate: 'CASCADE',
    });
  },
  down: async (queryInterface) => {
    await queryInterface.removeConstraint('EMPLOYEES', 'FK_EMP_DEPT');
    await queryInterface.removeColumn('EMPLOYEES', 'DEPARTMENT_ID');
  },
};
```

5. **Update service to include association when fetching employees** (already shown: `findAll({ include: { all: true } })`).
6. **Test the flow**
    - Run migrations: `npx sequelize-cli db:migrate`
    - Create department:

```
curl -X POST http://localhost:3000/api/departments -H "Content-Type:
application/json" \
  -d '{"name":"Engineering"}'
```

- Create employee referencing department ID:

```
curl -X POST http://localhost:3000/api/employees \
  -H "Content-Type: application/json" \
  -d '{"name":"Bob","email":"bob@example.com","departmentId":1}'
```

- Fetch employee and verify department included.

14. Further reading & references

- Sequelize docs — dialect-specific notes & installing drivers (Oracle uses `node-oracledb`). sequelize.org+1
- Oracle node-oracledb quickstart and installation guide (Instant Client). oracle.com
- NestJS Sequelize recipe (how to wire Sequelize in Nest). docs.nestjs.com
- Community blog posts about Sequelize + Oracle (useful examples). Medium+1

---

16. Classroom tips

- Demonstrate installing Oracle Instant Client first (so `oracledb` install succeeds). oracle.com
- Show how to set `TNS_ADMIN` and test a raw `oracledb` connection before starting Sequelize issues. Stack Overflow
- Emphasize differences between MySQL/Postgres habits and Oracle (uppercasing, sequences vs auto-increment). Oracle Forums

---

17. Quick checklist for trainees (before demo)

- Oracle DB accessible and Instant Client installed. oracle.com
- Environment variables set: `DB_USER`, `DB_PASS`, `DB_HOST`, `DB_PORT`, `DB_NAME`.
- Migrations ran: `npx sequelize-cli db:migrate`.
- App running and endpoints tested with curl/Postman.

Summary

- Introduction to Sequelize ORM
- Key Sequelize concepts: models, migrations, associations
- Connecting Sequelize with Oracle database
- Setting up Sequelize in a NestJS project
- Implementing CRUD with NestJS + Sequelize + Oracle DB
- Testing and verifying API endpoints