Answers

Question 17: Sequelize Models and Associations

a) Number of Database Tables (4 marks)

Answer: 5 tables will be generated:

1. **Courses** - Stores course information (courseCode, courseTitle, courseUnit)
2. **Students** - Stores student information (studentNumber, studentName)
3. **CourseOfferings** - Stores course offering information (courseOfferingId, academicYear, semester, and foreign key to Course)
4. **CourseRequires** (junction table) - Manages the self-referential many-to-many "Requires" relationship between courses (stores prerequisite relationships)
5. **Registrations** (junction table) - Manages the many-to-many relationship between Students and CourseOfferings (stores which students registered for which course offerings, including finalGrade)

b) Course Model Definition (4 marks)

```
const { DataTypes } = require("sequelize");

// Using sequelize.define approach
const Course = sequelize.define(
  "Course",
  {
    courseCode: {
      type: DataTypes.STRING,
      primaryKey: true,
      allowNull: false,
    },
    courseTitle: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    courseUnit: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        min: 4,
        max: 16,
      },
    },
  },
  {
    timestamps: false,
  }
);
```

c) CourseOffering Model Definition (4 marks)

```
const { DataTypes } = require("sequelize");

// Using Model.init approach
class CourseOffering extends Model {}

CourseOffering.init(
  {
    courseOfferingId: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
    academicYear: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    semester: {
      type: DataTypes.STRING,
      allowNull: false,
    },
  },
  {
    sequelize,
    modelName: "CourseOffering",
    timestamps: false,
  }
);
```

d) Requires and Mounted Relationships (4 marks)

```
// Requires relationship (self-referential many-to-many)
Course.belongsToMany(Course, {
  as: "Prerequisites",
  through: "CourseRequires",
  foreignKey: "dependentCourseCode",
  otherKey: "prerequisiteCourseCode",
});

Course.belongsToMany(Course, {
  as: "DependentCourses",
  through: "CourseRequires",
  foreignKey: "prerequisiteCourseCode",
  otherKey: "dependentCourseCode",
});

// Mounted relationship (one-to-many)
Course.hasMany(CourseOffering, {
  foreignKey: "courseCode",
  as: "Offerings",
});

CourseOffering.belongsTo(Course, {
  foreignKey: "courseCode",
  as: "Course",
});
```

e) finalGrade Relationship Attribute (4 marks)

Answer: Yes, it is possible to define the finalGrade relationship attribute in Sequelize.

This can be done by defining additional attributes on the junction table (through table) for the many-to-many relationship between Student and CourseOffering:

```
// Define the Registration association with finalGrade
Student.belongsToMany(CourseOffering, {
  through: {
    model: "Registration",
    attributes: ["finalGrade"],
  },
  foreignKey: "studentNumber",
  as: "RegisteredOfferings",
});

CourseOffering.belongsToMany(Student, {
  through: {
    model: "Registration",
    attributes: ["finalGrade"],
  },
  foreignKey: "courseOfferingId",
  as: "RegisteredStudents",
});

// Define the Registration model explicitly
const Registration = sequelize.define("Registration", {
  finalGrade: {
    type: DataTypes.STRING,
    allowNull: true,
  },
});
```

Question 18: Express.js Endpoints

a) GET /course Endpoint (8 marks)

```
// In course.js
const express = require("express");
const router = express.Router();
const { Course } = require("./models");
const { Op } = require("sequelize");

router.get("/", async (req, res) => {
  try {
    const { courseCode } = req.query;

    let whereClause = {};

    // If courseCode parameter is provided, filter by it
    if (courseCode) {
```

```
      whereClause.courseCode = {
        [Op.like]: `%${courseCode}%`,
      };
    }

    const courses = await Course.findAll({
      where: whereClause,
    });

    res.status(200).json(courses);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to retrieve courses" });
  }
});

module.exports = router;
```

b) Improvement Explanation (4 marks)

Why it's not useful: The current endpoint only returns basic course information but doesn't include critical details needed for course registration:

- **Prerequisites** - Students need to know which courses they must complete first
- **Course offerings** - Students need to see when/where courses are available
- **Dependent courses** - Students may want to know what courses this enables
- **Current availability** - Which offerings are available in the current semester

Improvements:

1. Include prerequisite information using Sequelize's `include` option
2. Include available course offerings for the current academic year/semester
3. Add filtering by academic year and semester parameters
4. Return aggregated data showing course + offerings together

Modified approach:

```
router.get("/", async (req, res) => {
  const { courseCode, academicYear, semester } = req.query;

  // Include Prerequisites and current CourseOfferings
  Course.findAll({
    where: courseCode ? { courseCode: { [Op.like]: `%${courseCode}%` } } : {
    include: [
      { model: Course, as: "Prerequisites" },
      {
        model: CourseOffering,
        as: "Offerings",
        where: academicYear && semester ? { academicYear, semester } : {},
        required: false,
      },
    ],
  });
});
```

c) PUT /registration Endpoint (8 marks)

```
// In registration.js
const express = require("express");
const router = express.Router();
const { Student, CourseOffering, Registration } = require("./models");

router.put("/", async (req, res) => {
  try {
    const { studentNumber, courseOfferingIds } = req.body;

    // Validate input
    if (
      !studentNumber ||
      !courseOfferingIds ||
      !Array.isArray(courseOfferingIds)
    ) {
      return res.status(400).json({
        error: "studentNumber and courseOfferingIds array are required",
      });
    }

    // Check if student exists
    const student = await Student.findByPk(studentNumber);
    if (!student) {
      return res.status(404).json({ error: "Student not found" });
    }

    // Register student for each course offering
    const registrations = [];
    for (const courseOfferingId of courseOfferingIds) {
      // Check if course offering exists
      const offering = await CourseOffering.findByPk(courseOfferingId);
      if (!offering) {
        return res.status(404).json({
          error: `Course offering ${courseOfferingId} not found`,
        });
      }

      // Create or update registration
      const [registration, created] = await Registration.findOrCreate({
        where: {
          studentNumber: studentNumber,
          courseOfferingId: courseOfferingId,
        },
        defaults: {
          finalGrade: null,
        },
      });

      registrations.push(registration);
    }

    res.status(200).json({
      message: "Registration successful",
      registrations: registrations,
    });
```

```
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to register for courses" });
  }
});

module.exports = router;
```

Question 19: React Components

a) Academic Year and Semester Selection Component (5 marks)

```
import React, { useState } from "react";

function AcademicYearSemesterSelector({ onSelectionChange }) {
  const [academicYear, setAcademicYear] = useState("AY 2024/2025");
  const [semester, setSemester] = useState("Semester 1");

  const academicYears = ["AY 2024/2025", "AY 2025/2026", "AY 2026/2027"];

  const semesters = ["Semester 1", "Semester 2"];

  const handleAcademicYearChange = (e) => {
    const newYear = e.target.value;
    setAcademicYear(newYear);
    if (onSelectionChange) {
      onSelectionChange(newYear, semester);
    }
  };

  const handleSemesterChange = (e) => {
    const newSemester = e.target.value;
    setSemester(newSemester);
    if (onSelectionChange) {
      onSelectionChange(academicYear, newSemester);
    }
  };

  return (
    <div>
      <label>
        Academic Year:
        <select value={academicYear} onChange={handleAcademicYearChange}>
          {academicYears.map((year) => (
            <option key={year} value={year}>
              {year}
            </option>
          ))}
        </select>
      </label>

      <label>
        Semester:
        <select value={semester} onChange={handleSemesterChange}>
```

```
          {semesters.map((sem) => (
            <option key={sem} value={sem}>
              {sem}
            </option>
          ))}
        </select>
      </label>
    </div>
  );
}

export default AcademicYearSemesterSelector;
```

b) Course Offerings Data Table Component (7 marks)

```
import React, { useEffect, useState } from "react";
import axios from "axios";

function CourseOfferingsTable({ academicYear, semester, onSelectionChange })
  const [offerings, setOfferings] = useState([]);
  const [selectedOfferings, setSelectedOfferings] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchCourseOfferings();
  }, [academicYear, semester]);

  const fetchCourseOfferings = async () => {
    setLoading(true);
    setError(null);
    try {
      const response = await axios.get("http://localhost:3001/course", {
        params: {
          academicYear: academicYear,
          semester: semester,
        },
      });
      setOfferings(response.data);
    } catch (err) {
      setError("Failed to load course offerings");
      console.error(err);
    } finally {
      setLoading(false);
    }
  };

  const handleCheckboxChange = (courseOfferingId) => {
    const newSelection = selectedOfferings.includes(courseOfferingId)
      ? selectedOfferings.filter((id) => id !== courseOfferingId)
      : [...selectedOfferings, courseOfferingId];

    setSelectedOfferings(newSelection);
    if (onSelectionChange) {
      onSelectionChange(newSelection);
    }
```

```
  };

  if (loading) return <div>Loading...</div>;
  if (error) return <div>{error}</div>;

  return (
    <table border="1">
      <thead>
        <tr>
          <th>Select</th>
          <th>Course Offering ID</th>
          <th>Course Code</th>
          <th>Course Title</th>
          <th>Academic Year</th>
          <th>Semester</th>
        </tr>
      </thead>
      <tbody>
        {offerings.map((offering) => (
          <tr key={offering.courseOfferingId}>
            <td>
              <input
                type="checkbox"
                checked={selectedOfferings.includes(offering.courseOfferingI
                onChange={() => handleCheckboxChange(offering.courseOfferingI
              />
            </td>
            <td>{offering.courseOfferingId}</td>
            <td>{offering.courseCode}</td>
            <td>{offering.courseTitle}</td>
            <td>{offering.academicYear}</td>
            <td>{offering.semester}</td>
          </tr>
        ))}
      </tbody>
    </table>
  );
}

export default CourseOfferingsTable;
```

c) Course Registration Page Component (8 marks)

```
import React, { useState } from "react";
import axios from "axios";
import AcademicYearSemesterSelector from "./AcademicYearSemesterSelector";
import CourseOfferingsTable from "./CourseOfferingsTable";

function CourseRegistrationPage() {
  const [academicYear, setAcademicYear] = useState("AY 2024/2025");
  const [semester, setSemester] = useState("Semester 1");
  const [selectedCourseOfferingIds, setSelectedCourseOfferingIds] = useState
    []
  );
  const [studentNumber, setStudentNumber] = useState("");
  const [message, setMessage] = useState("");
```

```
  const handleSelectionChange = (year, sem) => {
    setAcademicYear(year);
    setSemester(sem);
    setSelectedCourseOfferingIds([]); // Reset selections when period change
  };

  const handleOfferingsSelectionChange = (offeringIds) => {
    setSelectedCourseOfferingIds(offeringIds);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage("");

    if (!studentNumber) {
      setMessage("Please enter your student number");
      return;
    }

    if (selectedCourseOfferingIds.length === 0) {
      setMessage("Please select at least one course offering");
      return;
    }

    try {
      const response = await axios.put("http://localhost:3001/registration",
        studentNumber: studentNumber,
        courseOfferingIds: selectedCourseOfferingIds,
      });

      setMessage("Registration successful!");
      setSelectedCourseOfferingIds([]);
    } catch (error) {
      setMessage(
        "Registration failed: " + (error.response?.data?.error || error.messa
      );
      console.error(error);
    }
  };

  return (
    <div>
      <h1>Course Registration</h1>

      <div>
        <label>
          Student Number:
          <input
            type="text"
            value={studentNumber}
            onChange={(e) => setStudentNumber(e.target.value)}
            placeholder="Enter student number"
          />
        </label>
      </div>

      <h2>Select Academic Period</h2>
      <AcademicYearSemesterSelector onSelectionChange={handleSelectionChange
```

```
      <h2>Available Course Offerings</h2>
      <CourseOfferingsTable
        academicYear={academicYear}
        semester={semester}
        onSelectionChange={handleOfferingsSelectionChange}
      />

      <div>
        <button onClick={handleSubmit}>Register for Selected Courses</button>
      </div>

      {message && (
        <div style={{ marginTop: "10px", fontWeight: "bold" }}>{message}</div>
      )}
    </div>
  );
}

export default CourseRegistrationPage;
```