University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 4

Date: Friday, November 29th, 2024

Group Number: 32

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Martin Janzen	44630812	g4i8s	martinezjan2014@gmail.com
Armen Henry Bagdasarov	20302303	c7v1f	armenhb@student.ubc.ca
Borhan Rahmani Nejad	43553015	t3p1c	borhan200153@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Project Description:
This is PortObello. It's something.
This application helps companies, port managers, and ship captains gain information. We provide many port management options through data analysis. Users will be able to find the state of the global economy by location countries with large amounts of trade agreements, as well as view them by GDPs. Additionally, users are capable of filtering through tables to find ships that match their needs, as well as find ports with ships of a certain size, to align ships with the value of their cargo. There are many more features that we can list, but it's best for you to try it yourself, invest in PortObello, to make you say "Oh! Bello!" at the ease with which you can manage your port.

Differences in Schema:
We made four major adjustments to our initial schema:

1. Destination country instead of destination port in shipping route.
2. ISA relationship is now inclusive, rather than disjoint.
3. Added the unique constraint to the country government attribute.
4. Added the GoodValue attribute to Ship1.

The reasons are as follows:

1. Done to be able to change a port in a country without destroying a route.
2. Required for trade agreement functionality.
3. Governments are unique, this fit the real-world application best.
4. Increased

SQL Queries:

1. INSERT
   Location: in appService.js, insertHomeCountry(name, population, government, gdp, portaddress) line 674
   Implementation:

```
const result = await connection.execute(
   `INSERT INTO HOMECOUNTRY (name, population, government, gdp,
PortAddress)
    VALUES (:name, :population, :government, :gdp, :portaddress)`,
   [name, population, government, gdp, portaddress],
   { autoCommit: false }
);

const result2 = await connection.execute(
   `INSERT INTO FOREIGNCOUNTRY (name, population, government, gdp,
PortAddress, DockingFee)
    VALUES (:name, :population, :government, :gdp, :portaddress, 500.0)`,
   [name, population, government, gdp, portaddress],
   { autoCommit: false }
);
```

```
const result3 = await connection.execute(
   `UPDATE COUNTRY
    SET population = :population,
        government = :government,
        gdp = :gdp,
        portaddress = :portaddress
    WHERE name = :name
   `,
   [population, government, gdp, portaddress, name],
   { autoCommit: false }

);

await connection.commit();
```

2. UPDATE
   Location:in appService.js, updateCountry(cname, population, government, portaddress, gdp), line 245
   Implementation:

```
const checkResult = await connection.execute(
   `SELECT COUNT(*) AS COUNT1 FROM COUNTRY WHERE government =
:government`,
   [government]
);

const existingCount = checkResult.rows.length > 0 ?
checkResult.rows[0]["COUNT1"] : 0;

if (existingCount > 0) {
   // Government value already exists
   throw new Error(`Government value '${government}' already exists and
must be unique.`);
}

const result = await connection.execute(
   `UPDATE COUNTRY
       SET population=:population,
           government=:government,
           portaddress=:portaddress,
           gdp=:gdp
         WHERE name=:cname`,
   [population, government, portaddress, gdp, cname],
   { autoCommit: false }
);

const result2 = await connection.execute(
   `UPDATE HOMECOUNTRY
       SET population=:population,
```

```
            government=:government,
            portaddress=:portaddress,
            gdp=:gdp
          WHERE name=:cname`,
    [population, government, portaddress, gdp, cname],
    { autoCommit: false }
);

const result3 = await connection.execute(
    `UPDATE FOREIGNCOUNTRY
        SET population=:population,
            government=:government,
            portaddress=:portaddress,
            gdp=:gdp
          WHERE name=:cname`,
    [population, government, portaddress, gdp, cname],
    { autoCommit: false }
);

await connection.commit();
```

3. DELETE
   Location: appService.js, deletePort(addy) function. Line 1668
   Implementation:

```
await connection.execute( `
            DELETE FROM Warehouse
            WHERE PortAddress =:addy
    `,
   [addy],
    { autoCommit: true }
);
await connection.execute( `
            UPDATE Country
            SET PortAddress = 'No ports from this country are currently
monitored.'
            WHERE PortAddress =:addy
    `,
    [addy],
    { autoCommit: true }
);

await connection.execute( `
            UPDATE HomeCountry
            SET PortAddress = 'No ports from this country are currently
monitored.'
            WHERE PortAddress =:addy
    `,
    [addy],
```

```
    { autoCommit: true }
);

await connection.execute( `
            UPDATE ForeignCountry
            SET PortAddress = 'No ports from this country are currently
monitored.'
            WHERE PortAddress =:addy
    `,
    [addy],
    { autoCommit: true }
);


const deletion = await connection.execute( `
DELETE FROM Port WHERE PortAddress =:addy
`,
    [addy]
);
```

4. SELECTION
   Location: in appService.js, runDynamicShipQuery(userInput), line 2084
   Implementation:

```
const { whereClause, bindParams } = parseShipQuery(userInput);

console.log('parsed whereClause:', whereClause);


const query = `
    SELECT
        s1.Owner AS Owner,
        s1.ShipName AS ShipName,
        s1.ShipSize AS ShipSize,
        s2.Capacity AS Capacity,
        s1.ShippingRouteName AS ShippingRouteName,
        s1.DockedAtPortAddress AS DockedAtPortAddress
    FROM Ship1 s1
             LEFT JOIN Ship2 s2 ON s1.ShipSize = s2.ShipSize
    WHERE ${whereClause}`;
```

5. PROJECTION
   Location: appService.js projectShippingRoute(attributes), line 2035
   Implementation:

```
const selectClause = selectedAttributes.join(", ");
//query
return await withOracleDB(async (connection) => {
            await connection.execute(`
                SELECT ${selectClause}
```

```
                FROM ShippingRoute1
                JOIN ShippingRoute2
                ON ShippingRoute1.OriginCountryName =
ShippingRoute2.OriginCountryName
                AND ShippingRoute1.TerminalPortAddress =
ShippingRoute2.TerminalPortAddress
            `,
            { autoCommit: true }
            );
        })
```

6. JOIN
   Location: in appService.js, joinCompanyShipments(companyName, companyCEO), line 2010
   Implementation:

```
const result = await connection.execute(
   `
   SELECT sc.ShipOwner, sc.ShipName, sc.GoodType, sc.TrackingNumber,
sc.CompanyName, sc.CompanyCEO,
          c.Industry, c.YearlyRevenue
   FROM ShipmentContainer2 sc
   JOIN Company c ON sc.CompanyName = c.Name AND sc.CompanyCEO = c.CEO
   WHERE sc.CompanyName = :companyName AND sc.CompanyCEO = :companyCEO
   `,
   { companyName, companyCEO },
   { outFormat: oracledb.OUT_FORMAT_OBJECT }
);
```

7. AGGREGATION with GROUP BY
   Location: in appService.js, countCountry(), line 302
   Implementation:

```
const result = await connection.execute(
   `SELECT
      CASE
          WHEN GDP < 1 THEN '0-1'
          WHEN GDP BETWEEN 1 AND 5 THEN '1-5'
          WHEN GDP BETWEEN 5 AND 10 THEN '5-10'
          WHEN GDP BETWEEN 10 AND 15 THEN '10-15'
          ELSE '15+'
      END AS GDPRange,
      COUNT(*) AS CountryCount
    FROM COUNTRY
    GROUP BY
      CASE
          WHEN GDP < 1 THEN '0-1'
          WHEN GDP BETWEEN 1 AND 5 THEN '1-5'
          WHEN GDP BETWEEN 5 AND 10 THEN '5-10'
```

```
            WHEN GDP BETWEEN 10 AND 15 THEN '10-15'
            ELSE '15+'
        END
     ORDER BY GDPRange`,
    [],
    { outFormat: oracledb.OUT_FORMAT_OBJECT }
);
```

Description:
Throws countries into buckets depending on their gdp (in billions), and then counts the number of countries in each.

8. AGGREGATION with HAVING
   Location: in appService.js, portsNumShip(min,max), line 1849.
   Implementation:

```
const result = await connection.execute(
    `
        SELECT P.PortAddress     AS PortLocation,
               COUNT(S.ShipName) AS NumOfShips
        FROM Ship1 S
                 JOIN Port P ON S.DockedAtPortAddress = P.PortAddress
        WHERE S.ShipSize BETWEEN :min AND :max
        GROUP BY P.PortAddress
        HAVING COUNT(S.ShipName) > 0
    `,
    {min, max},
    {outFormat: oracledb.OUT_FORMAT_OBJECT}
);
```

   Description:
   This function allows companies to find how many ships of a certain size are found in ports. For example, if a supplier wanted to find how many ports contain small ships (say with sizes between 0-15), they could find out that only one port has ships that small, and that is where they should take their business.

9. NESTED AGGREGATION with GROUP BY
   Location: in appService.js, maxAvgContainer(), 1878
   Implementation:

```
const result = await connection.execute(`
    SELECT ShipName, MAX(avg_value) AS max_avg
    FROM (
             SELECT s1.ShipName, AVG(s2.GoodValue) AS avg_value
             FROM Ship1 s1
                      JOIN ShipmentContainer2 s2
                           ON s1.ShipName = s2.ShipName
             GROUP BY s1.ShipName
         )
```

```
    GROUP BY ShipName
`);
```

Description: This is a function which finds the ship with the highest average container value. This would be useful for shipping companies to know which ships are the most valuable.

10. DIVISION

    Location: in appService.js, fetchHomeCountriesWithAllTradeAgreements(), line 828

    Implementation:

```
        const query = `
            SELECT hc.Name
            FROM HomeCountry hc
            WHERE NOT EXISTS (
                SELECT fc.Name
                FROM ForeignCountry fc
                WHERE NOT EXISTS (
                    SELECT 1
                    FROM Tariff1 t
                    WHERE t.HomeName = hc.Name
                      AND t.ForeignName = fc.Name
                )
            )`;
```

    Description: This function finds all Home Countries that have agreements with every foreign country. This can be used by companies to see which countries have the most developed trade ties.

Citations:

The tab functionality on the frontend was sourced from this:
https://www.w3schools.com/howto/howto_js_tabs.asp

The button functionality on the frontend was based off this:
https://www.w3schools.com/css/css3_buttons.asp

A lot of our frontend was based off the demo project found here:
https://www.students.cs.ubc.ca/~cs-304/resources/javascript-oracle-resources/node-setup.html