Milestone 4

April 2, 2025

Project Group Number on Canvas: 'Group 41"

Name	Student ID	CS Alias	Preferred Email Address
Vincent Luong	73547515	v8c0o	vincentluong1@hotmail.com
Ahmed Khan	31684178	h6v1y	ahmeddxb400@gmail.com
Zain Ali	94391034	k9y0h	szainali284@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

CPSC 304 Introduction to Relational Databases
The University of British Columbia

1 Repository Link

Our Prison Database Management System can be found at:
`https://github.students.cs.ubc.ca/CPSC304-2024W-T2/project_h6v1y_k9y0h_v8c0o`

2 Project Summary

A brief summary about our project (2-3 sentences)

We are developing a prison database management system from the ground up. This system will store and manage essential information about a prison and its inmates while capturing and modeling the facility's internal logistics.

3 Project Schema Updates

A short description of how our final schema differed from the schema of our M2.

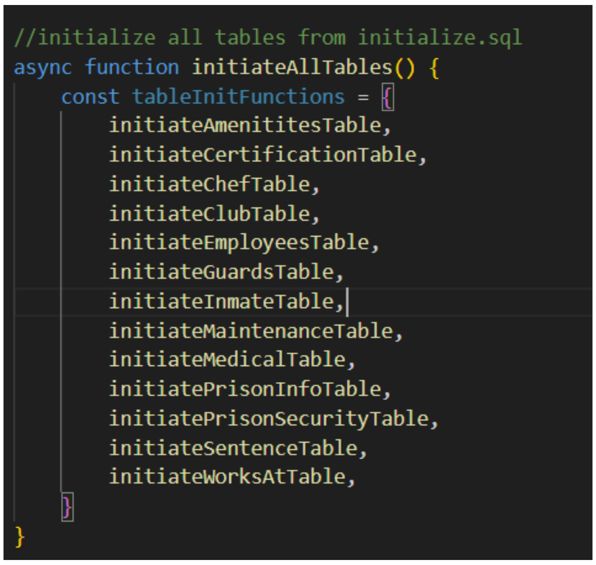
Our final schema stayed the same; the only difference occurred in the corrections in FK's and Many-to-Many constraints we needed to correct from Milestone 3.

We had also renamed our schema relationship tables Inmates1, Inmates2, to InmateInfo, and InmateCell.

Same thing can be said for our tables Prison1, and Prison2. For naming conventions, we had renamed them to PrisonInfo and PrisonSecurity respectively.

4 Copy of Screenshots that show Data Relations

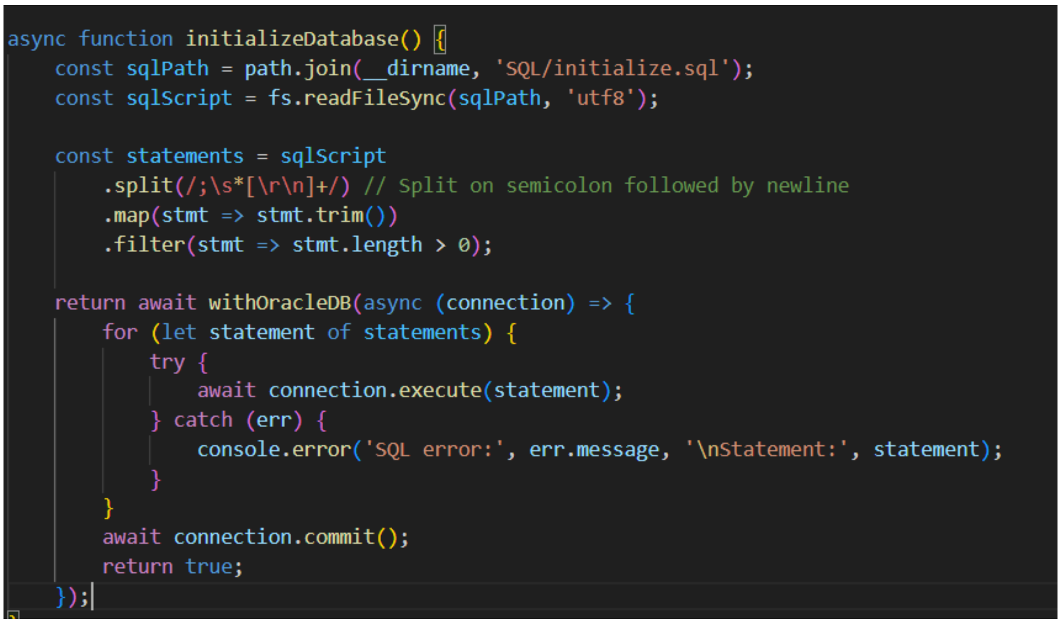
Can be found at appService.js line 405, and following data tables/types can be found above the line



```
//initialize all tables from initialize.sql
async function initiateAllTables() {
    const tableInitFunctions = [
        initiateAmenititesTable,
        initiateCertificationTable,
        initiateChefTable,
        initiateClubTable,
        initiateEmployeesTable,
        initiateGuardsTable,
        initiateInmateTable,
        initiateMaintenanceTable,
        initiateMedicalTable,
        initiatePrisonInfoTable,
        initiatePrisonSecurityTable,
        initiateSentenceTable,
        initiateWorksAtTable,
    ]
}
```

5 SQL Scripts to Create all Tables and Data

Can be found at appService.js line 492



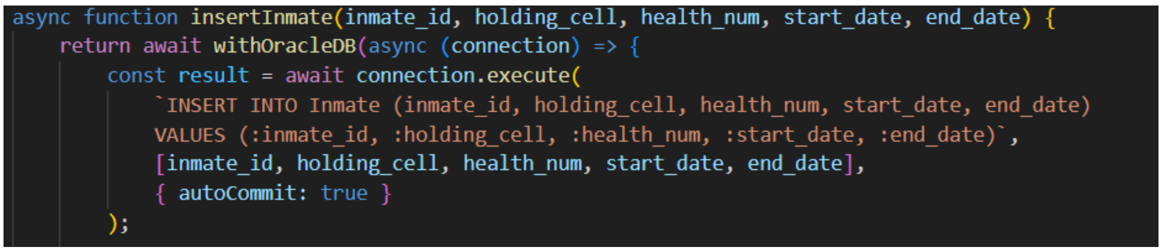
```javascript
async function initializeDatabase() {
    const sqlPath = path.join(__dirname, 'SQL/initialize.sql');
    const sqlScript = fs.readFileSync(sqlPath, 'utf8');

    const statements = sqlScript
        .split(/;\s*[\r\n]+/) // Split on semicolon followed by newline
        .map(stmt => stmt.trim())
        .filter(stmt => stmt.length > 0);

    return await withOracleDB(async (connection) => {
        for (let statement of statements) {
            try {
                await connection.execute(statement);
            } catch (err) {
                console.error('SQL error:', err.message, '\nStatement:', statement);
            }
        }
        await connection.commit();
        return true;
    });
```

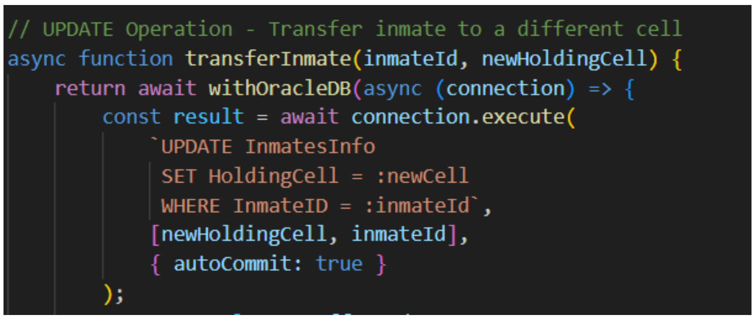
6 Queries

INSERT appService.js line 581



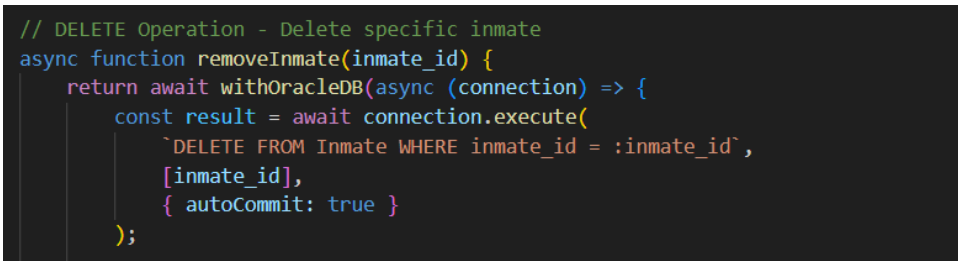
```
async function insertInmate(inmate_id, holding_cell, health_num, start_date, end_date) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `INSERT INTO Inmate (inmate_id, holding_cell, health_num, start_date, end_date)
            VALUES (:inmate_id, :holding_cell, :health_num, :start_date, :end_date)`,
            [inmate_id, holding_cell, health_num, start_date, end_date],
            { autoCommit: true }
        );
```

UPDATE appService.js line 624



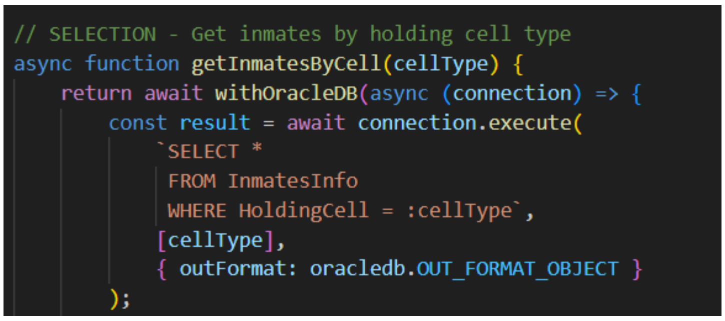
```
// UPDATE Operation - Transfer inmate to a different cell
async function transferInmate(inmateId, newHoldingCell) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `UPDATE InmatesInfo
             SET HoldingCell = :newCell
             WHERE InmateID = :inmateId`,
            [newHoldingCell, inmateId],
            { autoCommit: true }
        );
```

DELETE appService.js line 642



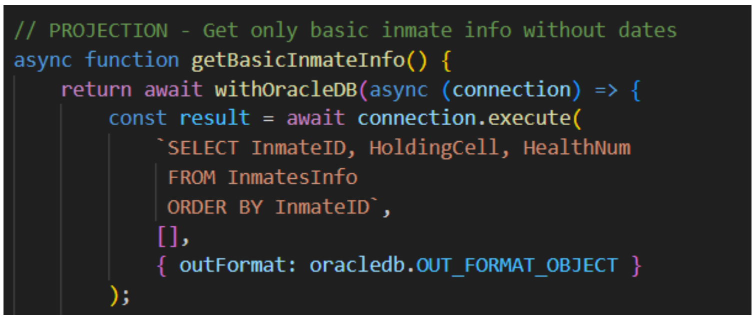
```
// DELETE Operation - Delete specific inmate
async function removeInmate(inmate_id) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `DELETE FROM Inmate WHERE inmate_id = :inmate_id`,
            [inmate_id],
            { autoCommit: true }
        );
```

SELECTION appService.js line 659



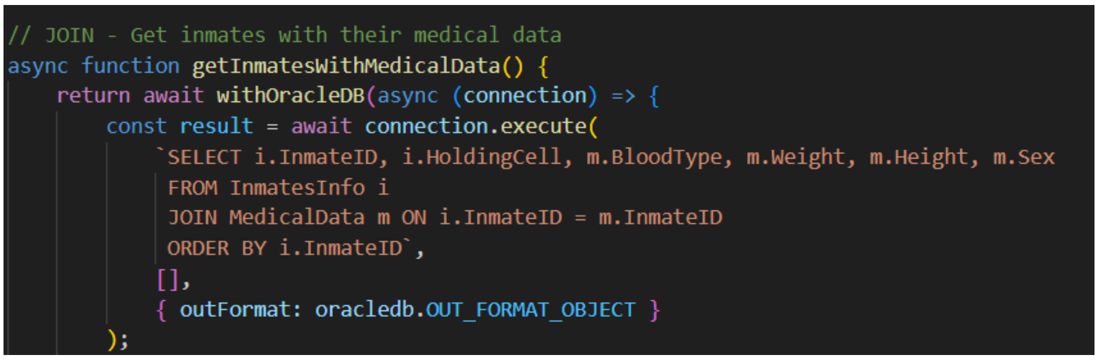
```
// SELECTION - Get inmates by holding cell type
async function getInmatesByCell(cellType) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT *
             FROM InmatesInfo
             WHERE HoldingCell = :cellType`,
            [cellType],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
```

PROJECTION appService.js line 676



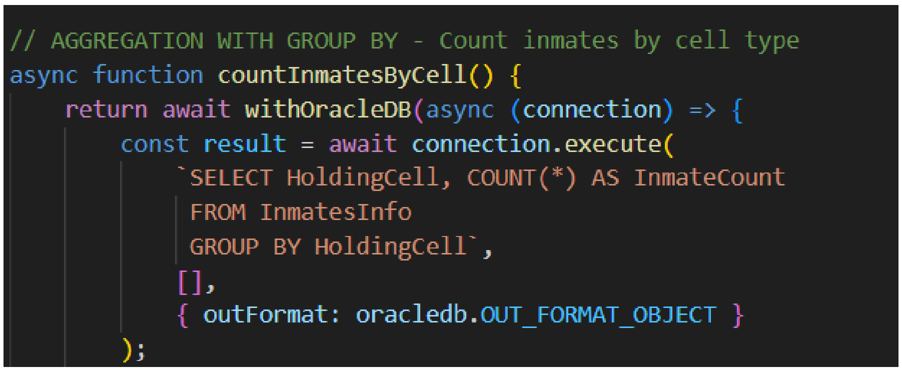
```
// PROJECTION - Get only basic inmate info without dates
async function getBasicInmateInfo() {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT InmateID, HoldingCell, HealthNum
             FROM InmatesInfo
             ORDER BY InmateID`,
            [],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
```

JOIN appService.js line 693



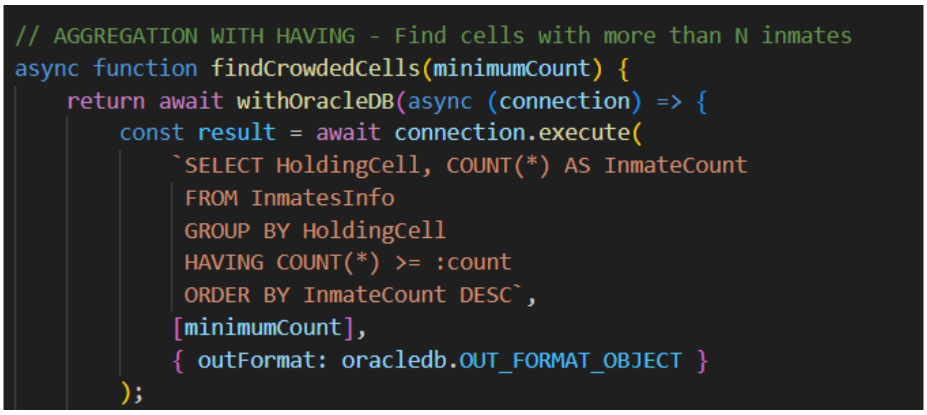
```
// JOIN - Get inmates with their medical data
async function getInmatesWithMedicalData() {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT i.InmateID, i.HoldingCell, m.BloodType, m.Weight, m.Height, m.Sex
            FROM InmatesInfo i
            JOIN MedicalData m ON i.InmateID = m.InmateID
            ORDER BY i.InmateID`,
            [],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
```

AGGREGATION WITH GROUP BY appService.js line 711



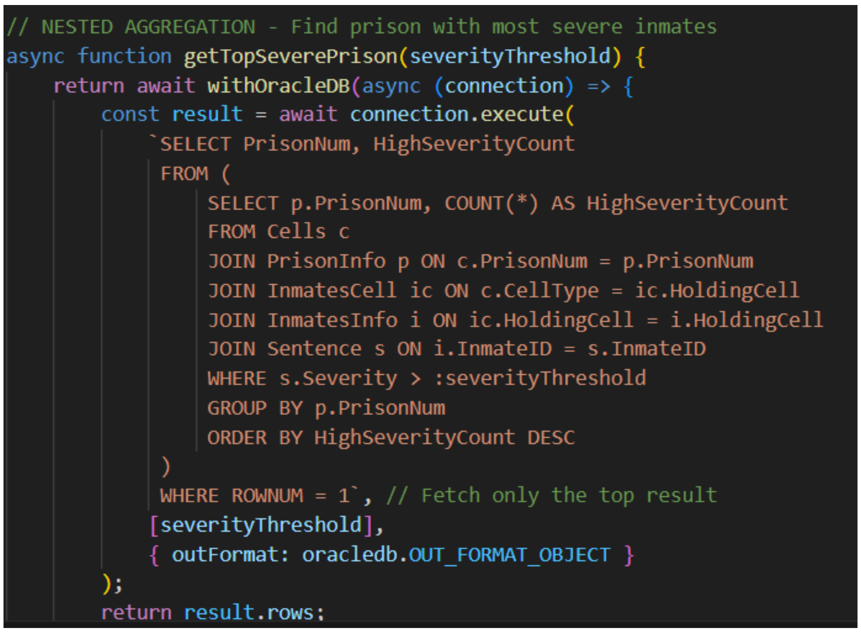
```
// AGGREGATION WITH GROUP BY - Count inmates by cell type
async function countInmatesByCell() {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT HoldingCell, COUNT(*) AS InmateCount
            FROM InmatesInfo
            GROUP BY HoldingCell`,
            [],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
```

AGGREGATION WITH HAVING appService.js line 730



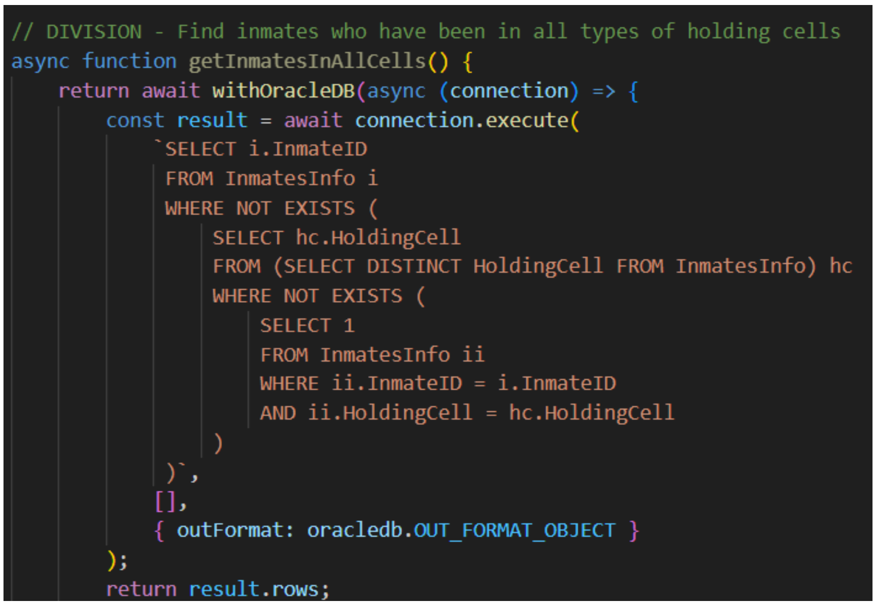
```
// AGGREGATION WITH HAVING - Find cells with more than N inmates
async function findCrowdedCells(minimumCount) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT HoldingCell, COUNT(*) AS InmateCount
            FROM InmatesInfo
            GROUP BY HoldingCell
            HAVING COUNT(*) >= :count
            ORDER BY InmateCount DESC`,
            [minimumCount],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
```

NESTED AGGREGATION WITH GROUP BY appService.js line 749



```
// NESTED AGGREGATION - Find prison with most severe inmates
async function getTopSeverePrison(severityThreshold) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT PrisonNum, HighSeverityCount
            FROM (
                SELECT p.PrisonNum, COUNT(*) AS HighSeverityCount
                FROM Cells c
                JOIN PrisonInfo p ON c.PrisonNum = p.PrisonNum
                JOIN InmatesCell ic ON c.CellType = ic.HoldingCell
                JOIN InmatesInfo i ON ic.HoldingCell = i.HoldingCell
                JOIN Sentence s ON i.InmateID = s.InmateID
                WHERE s.Severity > :severityThreshold
                GROUP BY p.PrisonNum
                ORDER BY HighSeverityCount DESC
            )
            WHERE ROWNUM = 1`, // Fetch only the top result
            [severityThreshold],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
        return result.rows;
```

DIVISION appService.js line 777



```
// DIVISION - Find inmates who have been in all types of holding cells
async function getInmatesInAllCells() {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT i.InmateID
            FROM InmatesInfo i
            WHERE NOT EXISTS (
                SELECT hc.HoldingCell
                FROM (SELECT DISTINCT HoldingCell FROM InmatesInfo) hc
                WHERE NOT EXISTS (
                    SELECT 1
                    FROM InmatesInfo ii
                    WHERE ii.InmateID = i.InmateID
                    AND ii.HoldingCell = hc.HoldingCell
                )
            )`,
            [],
            { outFormat: oracledb.OUT_FORMAT_OBJECT }
        );
        return result.rows;
```

7 READ.ME

The READ.ME file can be found in the GitHub Repo Link.