DBMS Lab Assignment-3

Database Connectivity

Harshit Jain [22CS10030]

SQL Program

```sql
SELECT c.name
FROM citizens AS c
JOIN land_records AS l ON c.citizen_id = l.citizen_id
WHERE l.land_area > 1.00;

SELECT c1.name
FROM citizens AS c1
JOIN households AS h ON c1.household_id = h.household_id
WHERE c1.gender = 'Female'
AND c1.is_student = TRUE
AND
( SELECT SUM(c2.income)
FROM citizens AS c2
WHERE c2.household_id = h.household_id
)
< 100000;

SELECT SUM(land_area)
FROM land_records
WHERE crop_type ILIKE 'rice';

SELECT COUNT(*)
FROM citizens
WHERE dob > '2000-01-01'
AND education ILIKE '10th';

SELECT c.name
FROM citizens AS c
JOIN panchayat_employees AS p ON c.citizen_id = p.citizen_id
JOIN land_records AS l ON p.citizen_id = l.citizen_id
WHERE l.land_area > 1.00;
```

```
SELECT c1.name
FROM citizens AS c1
JOIN households AS h ON h.household_id = c1.household_id
JOIN citizens AS c2 ON c2.household_id = h.household_id
JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id
WHERE p.role ILIKE 'Pradhan';

SELECT COUNT(*)
FROM assets
WHERE type ILIKE 'Street Light'
AND location ILIKE 'Phulera'
AND EXTRACT(YEAR FROM installation_date) = 2024;

SELECT COUNT(*)
FROM vaccinations AS v
JOIN citizens as c1 ON v.citizen_id = c1.citizen_id
JOIN citizens AS c2 ON c1.parent_id = c2.citizen_id
WHERE EXTRACT(YEAR FROM v.date_administered) = 2024
AND c2.education ILIKE '10th';

SELECT COUNT(*) from citizens
WHERE gender = 'Male'
AND EXTRACT(YEAR FROM dob) = 2024;
SELECT COUNT(DISTINCT c1.citizen_id)
FROM citizens AS c1
JOIN households AS h ON h.household_id = c1.household_id
JOIN citizens AS c2 ON c2.household_id = h.household_id
JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id;
```

High-level language programs

Python (**psycopg2**)

Setup:

Modules:

```
pip install psycopg2-binary python-dotenv
```

Environment variables (.env):
DB_NAME=your_database_name
DB_USER=your_username
DB_PASSWORD=your_password
DB_HOST=your_host
DB_PORT=your_port

Code:

```
import os
import psycopg2
from dotenv import load_dotenv
from datetime import datetime

# Load environment variables
load_dotenv()

# Database connection parameters from environment variables
DB_PARAMS = {
   'dbname': os.getenv('DB_NAME'),
   'user': os.getenv('DB_USER'),
   'password': os.getenv('DB_PASSWORD'),
   'host': os.getenv('DB_HOST', 'localhost'),
   'port': os.getenv('DB_PORT', '5432')
}

def execute_queries():
   # List of queries with descriptive labels
   queries = [
       {
           'label': 'Citizens with land area > 1.00',
           'query': """
               SELECT c.name
```

```
            FROM citizens AS c
            JOIN land_records AS l ON c.citizen_id = l.citizen_id
            WHERE l.land_area > 1.00;
        """
    },
    {
        'label': 'Female students from low-income households',
        'query': """
            SELECT c1.name
            FROM citizens AS c1
            JOIN households AS h ON c1.household_id = h.household_id
            WHERE c1.gender = 'Female'
            AND c1.is_student = TRUE
            AND
            ( SELECT SUM(c2.income)
            FROM citizens AS c2
            WHERE c2.household_id = h.household_id
            )
            < 100000;
        """
    },
    {
        'label': 'Total rice cultivation area',
        'query': """
            SELECT SUM(land_area)
            FROM land_records
            WHERE crop_type ILIKE 'rice';
        """
    },
    {
        'label': 'Young citizens with 10th education',
        'query': """
            SELECT COUNT(*)
            FROM citizens
            WHERE dob > '2000-01-01'
            AND education ILIKE '10th';
        """
    },
    {
        'label': 'Panchayat employees with land > 1.00',
        'query': """
            SELECT c.name
            FROM citizens AS c
            JOIN panchayat_employees AS p ON c.citizen_id = p.citizen_id
            JOIN land_records AS l ON p.citizen_id = l.citizen_id
            WHERE l.land_area > 1.00;
        """
    },
    {
        'label': 'Household members of Pradhans',
        'query': """
            SELECT c1.name
            FROM citizens AS c1
            JOIN households AS h ON h.household_id = c1.household_id
            JOIN citizens AS c2 ON c2.household_id = h.household_id
            JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id
            WHERE p.role ILIKE 'Pradhan';
        """
    },
    {
        'label': 'Street lights in Phulera installed in 2024',
        'query': """
            SELECT COUNT(*)
            FROM assets
            WHERE type ILIKE 'Street Light'
            AND location ILIKE 'Phulera'
            AND EXTRACT(YEAR FROM installation_date) = 2024;
        """
    },
    {
        'label': 'Vaccinations in 2024 for children of 10th pass parents',
        'query': """
            SELECT COUNT(*)
            FROM vaccinations AS v
            JOIN citizens as c1 ON v.citizen_id = c1.citizen_id
            JOIN citizens AS c2 ON c1.parent_id = c2.citizen_id
            WHERE EXTRACT(YEAR FROM v.date_administered) = 2024
            AND c2.education ILIKE '10th';
        """
    },
    {
        'label': 'Male births in 2024',
        'query': """
            SELECT COUNT(*) from citizens
            WHERE gender = 'Male'
            AND EXTRACT(YEAR FROM dob) = 2024;
        """
    },
    {
        'label': 'Unique citizens in households with panchayat employees',
        'query': """
            SELECT COUNT(DISTINCT c1.citizen_id)
            FROM citizens AS c1
            JOIN households AS h ON h.household_id = c1.household_id
            JOIN citizens AS c2 ON c2.household_id = h.household_id
            JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id;
        """
    }
]

try:
    # Establish database connection
    conn = psycopg2.connect(**DB_PARAMS)
    cur = conn.cursor()

    # Execute each query and print results
    for query_info in queries:
        print(f"\n=== {query_info['label']} ===")
        cur.execute(query_info['query'])
        results = cur.fetchall()

        # Print results in a formatted way
        if results:
            if len(results[0]) == 1:  # Single column result
                if len(results) == 1:  # Single row
                    print(f"Result: {results[0][0]}")
                else:
                    for row in results:
                        print(row[0])
            else:  # Multiple columns
                for row in results:
                    print(row)
        else:
            print("No results found")

except psycopg2.Error as e:
    print(f"Database error: {e}")
except Exception as e:
    print(f"Error: {e}")
finally:
    if 'cur' in locals():
        cur.close()
        if 'conn' in locals():
            conn.close()

if __name__ == "__main__":
    execute_queries()
```

Run:

```
python3 queries.py
```

C++ (**ODBC**)

Setup:

Packages:

```
sudo apt-get install unixodbc unixodbc-dev
sudo apt-get install odbc-postgresql
odbcinst -q -d
```

Config file (db_config.txt):

```
# Database configuration
Driver=PostgreSQL Unicode
Server=localhost
Port=5432
Database=your_database
Username=your_username
Password=your_password
```

Code:

```cpp
#include <sql.h>
#include <sqlext.h>
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <cstring>

// Function to read connection string from config file
std::string readConnectionString(const std::string& filename) {
   std::ifstream file(filename);
   if (!file.is_open()) {
       throw std::runtime_error("Unable to open config file: " + filename);
   }

   std::stringstream buffer;
   std::string line;
   std::string connStr;

   // Read file and construct connection string
   while (std::getline(file, line)) {
       // Skip empty lines and comments
```

```
        if (line.empty() || line[0] == '#') {
            continue;
        }

        size_t pos = line.find('=');
        if (pos != std::string::npos) {
            std::string key = line.substr(0, pos);
            std::string value = line.substr(pos + 1);

            // Trim whitespace
            key.erase(0, key.find_first_not_of(" \t"));
            key.erase(key.find_last_not_of(" \t") + 1);
            value.erase(0, value.find_first_not_of(" \t"));
            value.erase(value.find_last_not_of(" \t") + 1);

            buffer << key << "=" << value << ";";
        }
    }

    return buffer.str();
}

struct QueryInfo {
    std::string label;
    std::string query;
};

class DatabaseConnection {
private:
    SQLHENV env;
    SQLHDBC dbc;
    SQLHSTMT stmt;

    void checkError(SQLHANDLE handle, SQLSMALLINT type, RETCODE ret) {
        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO)
            return;

        SQLSMALLINT i = 0;
        SQLINTEGER native;
        SQLCHAR state[7];
        SQLCHAR text[256];
        SQLSMALLINT len;

        while (SQLGetDiagRec(type, handle, ++i, state, &native, text,
                            sizeof(text), &len) == SQL_SUCCESS) {
            std::cerr << "ODBC Error: " << text << std::endl;
        }
    }

public:
    DatabaseConnection() : env(NULL), dbc(NULL), stmt(NULL) {}

    bool connect(const std::string& connStr) {
        // Allocate environment handle
        if (SQL_SUCCESS != SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env)) {
            std::cerr << "Failed to allocate environment handle\n";
            return false;
        }

        // Set ODBC version
        if (SQL_SUCCESS != SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION,
(void*)SQL_OV_ODBC3, 0)) {
            std::cerr << "Failed to set ODBC version\n";
            return false;
        }

        // Allocate connection handle
        if (SQL_SUCCESS != SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc)) {
            std::cerr << "Failed to allocate connection handle\n";
            return false;
        }

        // Set connection timeout
        SQLSetConnectAttr(dbc, SQL_ATTR_CONNECTION_TIMEOUT, (SQLPOINTER)5, 0);

        // Connect to database
        SQLCHAR outstr[1024];
        SQLSMALLINT outstrlen;
        RETCODE ret = SQLDriverConnect(dbc, NULL,
                                     (SQLCHAR*)connStr.c_str(), SQL_NTS,
                                     outstr, sizeof(outstr),
                                     &outstrlen, SQL_DRIVER_NOPROMPT);

        if (SQL_SUCCESS != ret && SQL_SUCCESS_WITH_INFO != ret) {
            checkError(dbc, SQL_HANDLE_DBC, ret);
            return false;
        }
```

```cpp
        // Allocate statement handle
        if (SQL_SUCCESS != SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt)) {
            std::cerr << "Failed to allocate statement handle\n";
            return false;
        }

        return true;
    }

    void executeQuery(const QueryInfo& queryInfo) {
        std::cout << "\n=== " << queryInfo.label << " ===\n";

        RETCODE ret = SQLExecDirect(stmt, (SQLCHAR*)queryInfo.query.c_str(), SQL_NTS);
        if (SQL_SUCCESS != ret && SQL_SUCCESS_WITH_INFO != ret) {
            checkError(stmt, SQL_HANDLE_STMT, ret);
            return;
        }

        // Get number of columns
        SQLSMALLINT columns;
        SQLNumResultCols(stmt, &columns);

        // Fetch and print results
        while (SQL_SUCCESS == SQLFetch(stmt)) {
            for (SQLSMALLINT i = 1; i <= columns; i++) {
                SQLCHAR buffer[512];
                SQLLEN indicator;

                ret = SQLGetData(stmt, i, SQL_C_CHAR, buffer, sizeof(buffer),
&indicator);
                if (SQL_SUCCESS == ret || SQL_SUCCESS_WITH_INFO == ret) {
                    if (indicator != SQL_NULL_DATA)
                        std::cout << (i > 1 ? "\t" : "") << buffer;
                    else
                        std::cout << (i > 1 ? "\t" : "") << "NULL";
                }
            }
            std::cout << std::endl;
        }

        // Reset statement
        SQLCloseCursor(stmt);
    }
```

```
   ~DatabaseConnection() {
       if (stmt) SQLFreeHandle(SQL_HANDLE_STMT, stmt);
       if (dbc) {
           SQLDisconnect(dbc);
           SQLFreeHandle(SQL_HANDLE_DBC, dbc);
       }
       if (env) SQLFreeHandle(SQL_HANDLE_ENV, env);
   }
};

int main(int argc, char* argv[]) {
   if (argc < 2) {
       std::cerr << "Usage: " << argv[0] << " <config_file_path>\n";
       return 1;
   }

   std::string configFile = argv[1];
   std::string connectionString;

   try {
       connectionString = readConnectionString(configFile);
   } catch (const std::exception& e) {
       std::cerr << "Error reading config file: " << e.what() << std::endl;
       return 1;
   }

   std::vector<QueryInfo> queries = {
       {
           "Citizens with land area > 1.00",
           "SELECT c.name "
           "FROM citizens AS c "
           "JOIN land_records AS l ON c.citizen_id = l.citizen_id "
           "WHERE l.land_area > 1.00;"
       },
       {
           "Female students from low-income households",
           "SELECT c1.name "
           "FROM citizens AS c1 "
           "JOIN households AS h ON c1.household_id = h.household_id "
           "WHERE c1.gender = 'Female' "
           "AND c1.is_student = TRUE "
           "AND (SELECT SUM(c2.income) "
           "     FROM citizens AS c2 "
           "     WHERE c2.household_id = h.household_id) "
```

```
            "< 100000;"
        },
        {
            "Total rice cultivation area",
            "SELECT SUM(land_area) "
            "FROM land_records "
            "WHERE crop_type ILIKE 'rice';"
        },
        {
            "Young citizens with 10th education",
            "SELECT COUNT(*) "
            "FROM citizens "
            "WHERE dob > '2000-01-01' "
            "AND education ILIKE '10th';"
        },
        {
            "Panchayat employees with land > 1.00",
            "SELECT c.name "
            "FROM citizens AS c "
            "JOIN panchayat_employees AS p ON c.citizen_id = p.citizen_id "
            "JOIN land_records AS l ON p.citizen_id = l.citizen_id "
            "WHERE l.land_area > 1.00;"
        },
        {
            "Household members of Pradhans",
            "SELECT c1.name "
            "FROM citizens AS c1 "
            "JOIN households AS h ON h.household_id = c1.household_id "
            "JOIN citizens AS c2 ON c2.household_id = h.household_id "
            "JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id "
            "WHERE p.role ILIKE 'Pradhan';"
        },
        {
            "Street lights in Phulera installed in 2024",
            "SELECT COUNT(*) "
            "FROM assets "
            "WHERE type ILIKE 'Street Light' "
            "AND location ILIKE 'Phulera' "
            "AND EXTRACT(YEAR FROM installation_date) = 2024;"
        },
        {
            "Vaccinations in 2024 for children of 10th pass parents",
            "SELECT COUNT(*) "
            "FROM vaccinations AS v "
            "JOIN citizens as c1 ON v.citizen_id = c1.citizen_id "
            "JOIN citizens AS c2 ON c1.parent_id = c2.citizen_id "
            "WHERE EXTRACT(YEAR FROM v.date_administered) = 2024 "
            "AND c2.education ILIKE '10th';"
        },
        {
            "Male births in 2024",
            "SELECT COUNT(*) from citizens "
            "WHERE gender = 'Male' "
            "AND EXTRACT(YEAR FROM dob) = 2024;"
        },
        {
            "Unique citizens in households with panchayat employees",
            "SELECT COUNT(DISTINCT c1.citizen_id) "
            "FROM citizens AS c1 "
            "JOIN households AS h ON h.household_id = c1.household_id "
            "JOIN citizens AS c2 ON c2.household_id = h.household_id "
            "JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id;"
        }
    };

    DatabaseConnection db;
    if (!db.connect(connectionString)) {
        std::cerr << "Failed to connect to database\n";
        return 1;
    }

    for (const auto& query : queries) {
        db.executeQuery(query);
    }

    return 0;
}
```

Compile and Run:

```
g++ -o db_query queries.cpp -lodbc
./db_query db_config.txt
```

Java (**JDBC**)

<u>Setup:</u>

Drivers:

```
wget https://jdbc.postgresql.org/download/postgresql-42.7.2.jar
```

DB properties file (db.properties):

host=localhost

port=5432

database=your_database

username=your_username

password=your_password

<u>Code:</u>

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

class QueryInfo {
   String label;
   String query;

   public QueryInfo(String label, String query) {
       this.label = label;
       this.query = query;
   }
}

public class DatabaseQueries {
   private static Properties loadConfig(String filename) throws IOException {
       Properties props = new Properties();
       try (FileInputStream fis = new FileInputStream(filename)) {
           props.load(fis);
```

```java
        }
        return props;
    }

    private static String buildConnectionString(Properties props) {
        return String.format("jdbc:postgresql://%s:%s/%s",
            props.getProperty("host", "localhost"),
            props.getProperty("port", "5432"),
            props.getProperty("database")
        );
    }

    private static void executeQuery(Connection conn, QueryInfo queryInfo) throws
SQLException {
        System.out.println("\n=== " + queryInfo.label + " ===");

        try (PreparedStatement stmt = conn.prepareStatement(queryInfo.query);
             ResultSet rs = stmt.executeQuery()) {

            ResultSetMetaData metaData = rs.getMetaData();
            int columnCount = metaData.getColumnCount();

            while (rs.next()) {
                StringBuilder row = new StringBuilder();
                for (int i = 1; i <= columnCount; i++) {
                    if (i > 1) row.append("\t");
                    String value = rs.getString(i);
                    row.append(value == null ? "NULL" : value);
                }
                System.out.println(row);
            }
        }
    }

    public static void main(String[] args) {
        if (args.length < 1) {
            System.err.println("Usage: java DatabaseQueries <config_file>");
            System.exit(1);
        }

        List<QueryInfo> queries = new ArrayList<>();
        queries.add(new QueryInfo(
            "Citizens with land area > 1.00",
            "SELECT c.name " +
            "FROM citizens AS c " +
            "JOIN land_records AS l ON c.citizen_id = l.citizen_id " +
            "WHERE l.land_area > 1.00;"
        ));

        queries.add(new QueryInfo(
            "Female students from low-income households",
            "SELECT c1.name " +
            "FROM citizens AS c1 " +
            "JOIN households AS h ON c1.household_id = h.household_id " +
            "WHERE c1.gender = 'Female' " +
            "AND c1.is_student = TRUE " +
            "AND (SELECT SUM(c2.income) " +
            "     FROM citizens AS c2 " +
            "     WHERE c2.household_id = h.household_id) " +
            "< 100000;"
        ));

        queries.add(new QueryInfo(
            "Total rice cultivation area",
            "SELECT SUM(land_area) " +
            "FROM land_records " +
            "WHERE crop_type ILIKE 'rice';"
        ));

        queries.add(new QueryInfo(
            "Young citizens with 10th education",
            "SELECT COUNT(*) " +
            "FROM citizens " +
            "WHERE dob > '2000-01-01' " +
            "AND education ILIKE '10th';"
        ));

        queries.add(new QueryInfo(
            "Panchayat employees with land > 1.00",
            "SELECT c.name " +
            "FROM citizens AS c " +
            "JOIN panchayat_employees AS p ON c.citizen_id = p.citizen_id " +
            "JOIN land_records AS l ON p.citizen_id = l.citizen_id " +
            "WHERE l.land_area > 1.00;"
        ));

        queries.add(new QueryInfo(
            "Household members of Pradhans",
            "SELECT c1.name " +
            "FROM citizens AS c1 " +
            "JOIN households AS h ON h.household_id = c1.household_id " +
            "JOIN citizens AS c2 ON c2.household_id = h.household_id " +
            "JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id " +
            "WHERE p.role ILIKE 'Pradhan';"
        ));

        queries.add(new QueryInfo(
            "Street lights in Phulera installed in 2024",
            "SELECT COUNT(*) " +
            "FROM assets " +
            "WHERE type ILIKE 'Street Light' " +
            "AND location ILIKE 'Phulera' " +
            "AND EXTRACT(YEAR FROM installation_date) = 2024;"
        ));

        queries.add(new QueryInfo(
            "Vaccinations in 2024 for children of 10th pass parents",
            "SELECT COUNT(*) " +
            "FROM vaccinations AS v " +
            "JOIN citizens as c1 ON v.citizen_id = c1.citizen_id " +
            "JOIN citizens AS c2 ON c1.parent_id = c2.citizen_id " +
            "WHERE EXTRACT(YEAR FROM v.date_administered) = 2024 " +
            "AND c2.education ILIKE '10th';"
        ));

        queries.add(new QueryInfo(
            "Male births in 2024",
            "SELECT COUNT(*) from citizens " +
            "WHERE gender = 'Male' " +
            "AND EXTRACT(YEAR FROM dob) = 2024;"
        ));

        queries.add(new QueryInfo(
            "Unique citizens in households with panchayat employees",
            "SELECT COUNT(DISTINCT c1.citizen_id) " +
            "FROM citizens AS c1 " +
            "JOIN households AS h ON h.household_id = c1.household_id " +
            "JOIN citizens AS c2 ON c2.household_id = h.household_id " +
            "JOIN panchayat_employees AS p ON p.citizen_id = c2.citizen_id;"
        ));

        try {
            // Load database configuration
            Properties config = loadConfig(args[0]);
            String url = buildConnectionString(config);

            // Connect to database and execute queries
            try (Connection conn = DriverManager.getConnection(
                    url,
                    config.getProperty("username"),
                    config.getProperty("password"))) {

                for (QueryInfo query : queries) {
                    try {
                        executeQuery(conn, query);
                    } catch (SQLException e) {
                        System.err.println("Error executing query '" + query.label + "':
" + e.getMessage());
                    }
                }
            }
        } catch (IOException e) {
            System.err.println("Error reading configuration file: " + e.getMessage());
        } catch (SQLException e) {
            System.err.println("Database error: " + e.getMessage());
        }
    }
}
```

Compile and Run:

```
javac -cp postgresql-42.7.2.jar DatabaseQueries.java
java -cp .:postgresql-42.7.2.jar DatabaseQueries db.properties
```