```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <occi.h>
#include <cstring>
#include <iomanip>

using oracle::occi::Environment;
using oracle::occi::Connection;
using namespace oracle::occi;
using namespace std;
struct Employee {
	int employeeNumber;
	char lastName[50];
	char firstName[50];
	char email[100];
	char phone[50];
	char extension[10];
	char reportsTo[100];
	char jobTitle[50];
	char city[10];
};

//Function prototypes
int menu(void);
int getInt();
int getIntRange(int min, int max);
int findEmployee(Connection* conn, int employeeNumber, Employee* emp);
void insertEmployee(Connection* conn, Employee* emp);
void displayAllEmployees(Connection* conn);
void displayEmployee(Connection* conn, Employee emp);
void updateEmployee(Connection* conn, int employeeNumber);
void deleteEmployee(Connection* con, int employeeNumber);
//Function Definitions
int getInt() { // get selection from user
	int value = 0;
	bool badEntry;
	char nextChar;
	do {
		badEntry = false;
		cin >> value;
		if (cin.fail()) {
			cout << "Bad integer value, try again: ";
			cin.clear();
			cin.ignore(3000, '\n');
			badEntry = true;
		}
		else {
			nextChar = cin.get();
			if (nextChar != '\n') {
				cout << "Only enter an integer, try again: ";
				cin.ignore(3000, '\n');
				badEntry = true;
			}
		}
	} while (badEntry);
	return value;
}

int getIntRange(int min, int max)//check the input value that user enter if it
is in the range
{
	int val = getInt();// the getInt function was given and assigning integer
valut to func
```

```
	while (val < min || val > max) {// checks the range of values
		cout << "Invalid value enterd, retry[0 <= value <= 6]: ";
		val = getIntRange(1, 6);// and sets the val range
	}
	return val;

}
int menu(void)///prints menu and set user input

{

	cout << "********************* HR Menu *********************" << endl;
	cout << "1) Find Employee" << endl;
	cout << "2) Employees Report" << endl;
	cout << "3) Add Employee" << endl;
	cout << "4) Update Employee" << endl;
	cout << "5) Remove Employee" << endl;
	cout << "6) Exit" << endl;
	cout << "Please enter an option: ";

	int menuOption = -1;


		string selection;

		cin >> selection;

		while (menuOption ==-1) {
			if (selection == "1")
			{
				menuOption = 1;
			}

			else if (selection == "2")

			{

				menuOption = 2;

			}

			else if (selection == "3")

			{

				menuOption = 3;

			}

			else if (selection == "4")

			{

				menuOption = 4;

			}

			else if (selection == "5")

			{

				menuOption = 5;

			}
```

```
                else if (selection == "6")
                {
                        menuOption = 0;
                }
                else
                {
                        cout << "Please enter a valid option from the list: ";
                        menuOption = -1;
                }
                cin.clear();
                cin.ignore(2000, '\n');


        }

        return menuOption;
}
int main() {
        Environment* env = nullptr;
        Connection* conn = nullptr;
        string str;
        string usr = "";
        string pass = "4525838";
        string srv = "myoracle12c.";

        Employee* emp = nullptr;
        emp = new Employee;


        try {
                env = Environment::createEnvironment(Environment::DEFAULT);
                conn = env->createConnection(usr, pass, srv);
                cout << "Connection is Successful!" << endl << endl;
                int selection = 0;
                int flag = 0;
                int employeeNumber = 0;
                int check = 0;

                while (!flag)
                {
                        selection = menu();
                        switch (selection)
                        {
                        case 1:
                                cout << "Enter Employee Number: ";
```

```
                        cin >> employeeNumber;

                        check = findEmployee(conn, employeeNumber, emp);

                        if (check == 1)
                        {
                              displayEmployee(conn, *emp);
                        }
                        else
                        {
                              cout << "Employee " << employeeNumber << " does
not exist." << endl;
                        }


                        break;

                  case 2:
                        displayAllEmployees(conn);
                        break;

                  case 3:
                        insertEmployee(conn, emp);
                        break;

                  case 4:
                        cout << "Enter Employee Number: ";
                        cin >> employeeNumber;
                        check = findEmployee(conn, employeeNumber, emp);
                        if (check == 1)
                        {

                              updateEmployee(conn, employeeNumber);
                        }
                        else
                        {
                              cout << "Employee " << employeeNumber << " does
not exist." << endl;
                        }

                        break;

                  case 5:

                        cout << "Employee Number: ";
                        cin >> employeeNumber;

                        check = findEmployee(conn, employeeNumber, emp);

                        if (check == 1)
                        {

                              deleteEmployee(conn, employeeNumber);

                        }
                        else
                        {
                              cout << "Employee " << employeeNumber << " does
not exist." << endl;
                        }

                        break;

                  case 6:
```

```
                        env->terminateConnection(conn);
                        Environment::terminateEnvironment(env);
                        cout << "Exiting";
                      flag= 1;
                  }
            }
      }
      catch (SQLException& sqlExcp) {
            cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
      }
      return 0;
}



int findEmployee(Connection* conn, int employeeNumber, Employee* emp) {
      int flag= 0;
      try
      {
            Statement* stmt = conn->createStatement("SELECT * FROM employees
WHERE employeenumber = :1");
            stmt->setInt(1, employeeNumber);
            ResultSet* rs = stmt->executeQuery();

            if (!rs->next())
            {
                  flag = 0;
            }
            else
            {
                  flag = 1;

                  emp->employeeNumber = rs->getInt(1);
                  strcpy(emp->lastName, rs->getString(2).c_str());
                  strcpy(emp->firstName, rs->getString(3).c_str());
                  strcpy(emp->email, rs->getString(4).c_str());
                  strcpy(emp->extension, rs->getString(5).c_str());
                  strcpy(emp->phone, rs->getString(6).c_str());
                  strcpy(emp->reportsTo, rs->getString(7).c_str());
                  strcpy(emp->jobTitle, rs->getString(8).c_str());
                  strcpy(emp->city, rs->getString(9).c_str());

            }
      }
      catch (SQLException& sqlExcp)
      {
            cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();

      }

      return flag;


}
void displayEmployee(Connection* conn, Employee emp) {



      cout << "-------------- Employee Information -------------" << endl;
      cout << "Employee Number: " << emp.employeeNumber << endl;
      cout << "Last Name: " << emp.lastName << endl;
      cout << "First Name: " << emp.firstName << endl;
      cout << "Email: " << emp.email << endl;
      cout << "Phone: " << emp.phone << endl;
```

```
	cout << "Extension: " << emp.extension << endl;
	cout << "Reporsto: " << emp.reportsTo << endl;
	cout << "Job Title: " << emp.jobTitle << endl;
	cout << "City: " << emp.city << endl;

}

void displayAllEmployees(Connection* conn) {
	try {
		Statement* stmt = conn->createStatement();
		ResultSet* rs = stmt->executeQuery("SELECT *FROM (SELECT DISTINCT
emp.EMPLOYEENUMBER, emp.FIRSTNAME || ' ' || emp.LASTNAME AS \"Employee Name\",
emp.Email, x.PHONE, emp.EXTENSION ,b.FIRSTNAME || ' ' || b.LASTNAME AS \"Manager
Name\" FROM EMPLOYEES emp FULL OUTER JOIN EMPLOYEES b ON
emp.REPORTSTO=b.EMPLOYEENUMBER FULL OUTER JOIN OFFICES x ON emp.CITY= x.CITY
WHERE emp.EMPLOYEENUMBER IS NOT NULL ORDER BY emp.EMPLOYEENUMBER)");
		int numberofemployee = 0;
		while (rs->next())
		{
			numberofemployee = rs->getInt(1);

			break;
		}
		if (numberofemployee > 0) {
			ResultSet* rs = stmt->executeQuery("SELECT DISTINCT
emp.EMPLOYEENUMBER, emp.FIRSTNAME || ' ' || emp.LASTNAME AS \"Employee Name\",
emp.Email, x.PHONE, emp.EXTENSION ,b.FIRSTNAME || ' ' || b.LASTNAME AS \"Manager
Name\" FROM EMPLOYEES emp FULL OUTER JOIN EMPLOYEES b ON
emp.REPORTSTO=b.EMPLOYEENUMBER FULL OUTER JOIN OFFICES x ON emp.CITY = x.CITY
WHERE emp.EMPLOYEENUMBER IS NOT NULL ORDER BY emp.EMPLOYEENUMBER");


			cout << left << setw(15) << "E" << setw(20) << "Employee Name"
<< setw(35) << "Email" << setw(25) << "Phone" << setw(15) << "Extension" <<
setw(25) << "Manager" << endl;


			while (rs->next()) {
				int employeeID = rs->getInt(1);
				string empName = rs->getString(2);
				string email = rs->getString(3);
				string phone = rs->getString(4);
				string extension = rs->getString(5);
				string manName = rs->getString(6);
				cout << left << setw(17) << employeeID << setw(15) <<
empName << setw(25) << email << setw(25) << phone << setw(15) << extension <<
setw(25) << manName << endl;
			}
		}
		else {
			cout << "There is no employees' information " << endl;
		}
	}
	catch (SQLException& sqlExcp) {
		cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
	}


}
void insertEmployee(Connection* conn, Employee* emp) {
	try {
		Statement* stmt = conn->createStatement();
		stmt->setSQL("INSERT INTO EMPLOYEES VALUES
(:1,:2,:3,:4,:5,:6,:7,:8)");
```

```
            stmt->setInt(1, emp->employeeNumber);
            stmt->setString(2, emp->lastName);
            stmt->setString(3, emp->firstName);
            stmt->setString(4, emp->extension);
            stmt->setString(5, emp->email);
            stmt->setString(6, emp->city);
            stmt->setString(7, emp->reportsTo);
            stmt->setString(8, emp->jobTitle);
            stmt->executeUpdate();
            cout << "\nThe new employee is added successfully." << endl;

      }
      catch (SQLException& sqlExcp)
      {
            cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
      }

}

void updateEmployee(Connection* conn, int employeeNumber)//Final part
{
      try
      {
            string extension = "x0000";
            cout << "Extension: ";
            cin >> extension;

            Statement* stmt = conn->createStatement("UPDATE employees SET
extension = :1 WHERE employeenumber = :2");
            stmt->setString(1, extension);
            stmt->setInt(2, employeeNumber);
            stmt->executeUpdate();

            cout << "The employee's extension is updated succesfully." << endl;

      }
      catch (SQLException& sqlExcp)
      {
            cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
      }
}
```