

# University Management System



Object Oriented Programing  
Semester V Group Project

Submitted to  
Dr. Mukesh Kumar Giluka

## MEMBERS

AVADHESH KUMAR	20/11/EC/007
CHIRAG BENIWAL	20/11/EC/008
KSHAMA MEENA	20/11/EC/055
NEERAJ NIKHIL ROY	20/11/EC/053

CSE 2020, School of Engineering, Jawaharlal Nehru University  
New Delhi 110067  
February 2023

## ABSTRACT

This report includes a development presentation of an information system for managing the student data within a small company or organization. The system as such as it has been developed is called University Management System. It consists of functionally related GUI (application program). The choice of the programming tools is individual and particular.

## Keywords

University Management System (ums), UMS, Graphics User Interface, GUI, Firemonkey libraries, fmx file types, C++, oops, class, objects, linking, database, UI, UX, elements, navigation, cpp, h, fmx, sql.

**This application is also platform independent so it can be directly shipped in all of the following operating systems with some minor changes.**

**Windows 64 bit  
Windows 32 bit  
Apple IOS  
Apple mac OS  
Android 8.0 or up**

## Contents

- Introduction
- Software Used
- Though Process
- Code
- Brief application walkthrough
- Environment Setup
- Conclusions

## Applications Used

- Visual Studio Community 2022
- Rad Studio C++ Builder
- InterBase Database (SQL)
- Visual Studio Code



## Introduction

A University Management System is a software application designed to manage the day-to-day operations of a university. The system helps university administrators to manage student admissions, enrollments, class schedules, academic records, and other related administrative tasks. In this report, we will discuss the implementation of a University Management System project in C++ using RAD Studio.

## Object-Oriented Programming:

Object-Oriented Programming (OOP) is a programming paradigm that is widely used in software development. It focuses on creating objects that encapsulate data and functionality, allowing for modular and reusable code. OOP has several key concepts, including inheritance, polymorphism, and encapsulation, which can be used to develop complex software systems.

## **C++ Programming Language:**

C++ is an object-oriented programming language that is widely used in software development. It is a general-purpose language that supports multiple paradigms, including procedural, object-oriented, and generic programming. C++ is a compiled language, which means that the source code is compiled into machine code before it is executed.

## **RAD Studio:**

RAD Studio is a software development environment that is used to develop Windows and Mac applications. It is based on the Delphi programming language and provides a visual programming environment that allows developers to create user interfaces using drag and drop tools. RAD Studio also includes a range of libraries and components that can be used to develop complex software systems.

## University Management System Project:

The University Management System project is designed to manage the day-to-day operations of a university. The system consists of several modules, including student admissions, enrollments, class schedules, academic records, and other related administrative tasks. The system is designed using OOP concepts and implemented in C++ using RAD Studio. The system consists of several classes, including a student class, a course class, and a schedule class. Each class has its own set of data members and member functions that are used to manage the data and functionality of the system.

The student class is used to manage student admissions and academic records. It has data members that store the student's name, address, phone number, and academic records, including grades and course credits. The class also has member functions that are used to add, update, and delete student records.

The course class is used to manage the courses that are offered by the university. It has data members that store the course name, course code, and course credits. The class also has member functions that are used to add, update, and delete course records. The schedule class is used to manage the class schedules for each course. It has data members that store the course name, class time, class location, and instructor. The class also has member functions that are used to add, update, and delete class schedules.

Moreover this application first was made / coded in such a way that all the student data was too be stored in text files or in csv files, but later on stages with further development we have added SQL database. I know right pretty great so yeah; The SQL database Interbase implementation over the localhost is done which is interconnected by UMS application.

## Why choose us? How is our product better than others?

There are three primary reasons why educational establishments should build university data management software.

**Efficiency** Manual entries, filling up papers, distributing pamphlets, putting announcements on physical bulletin boards are all inefficient. When all aspects of a university student administration process are online and automated, students have better leverage, control, and clarity. It saves time, cost, and confusion. Even teachers find that managing their activities, tasks, and scheduling is easier with a UMS. Teachers, faculty, and university administration personnel can focus on more creative tasks instead of wasting time on mundane repetitive tasks.

### **Flexibility**

A UMS provides the flexibility to define workflows as per changing university requirements. Hard-wired software-based workflows prevent the university from applying case-based workflows. Instead, a flexible and highly-configurable workflow-system can support any number of use cases. It also lets the university expand its capabilities and offerings without worrying about a technological upgrade or overhaul.

### **Relevancy**

A UMS is a bare necessity because education is becoming digital and online. Traditional classroom-based education is being replaced by virtual classrooms. The latter provides the same level of experience as the former. Added to that, virtual classrooms are a more powerful tool that enables a university to impart education to a wider and broader student base using the university ERP.

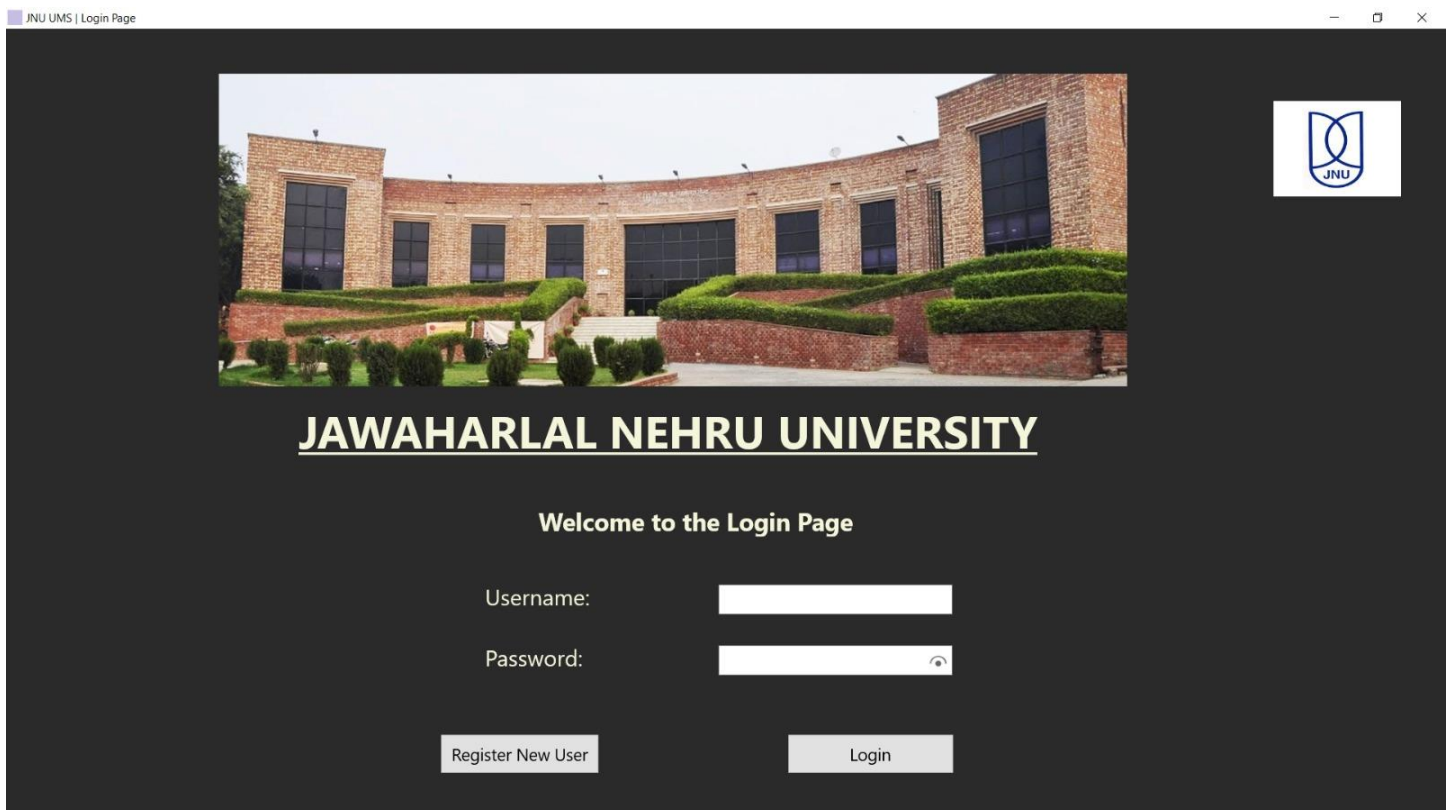
- Eliminates data corruption either through willful misappropriation or by manual errors.
- Data/information security with strong encryption.
- Ability for customizations and extensibilities.

## **Who needs a university management system?**

Every university needs a university management system to stay competitive, in the reckoning, to win favor with students, and impart quality education. Without a UMS, a university will operate inefficiently, and will be unable to create any value for the student community and teachers.

## Login Page

This right here is the welcome screen. It is a basic login page which welcomes the user / administrator. Here there are two text inputs boxes and two buttons where these boxes take the user input of username and password which are registered. To go to homepage, click login button which check the user input. If the password and username matches that with the previously entered / registered details then it successfully loges you in to the homepage. The add new user button shows the new administrator registration window where the user can register the new admin that can work on the UMS application. These forms are made in such a way that after the process of successful login or even unsuccessful one it clears the text data.



The code for the following backend service where login pages algorithm for checking of password from registered users text file is below and some UI elements.

### Header File

```
// -----
#ifndef LoginFormH
```



```

#define LoginFormH
// -----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Objects.hpp>
#include <FMX.Types.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Edit.hpp>
#include <FMX.ExtCtrls.hpp>
#include <FMX.Layouts.hpp>

// -----
class TLoginPage : public TForm {
__published: // IDE-managed Components
    TLabel *Banner;
    TLabel *Username;
    TEdit *UsernameBox;
    TLabel *Password;
    TEdit *PasswordBox;
    TButton *RegisterNewUserNavigationButton;
    TButton *LoginButton;
    TImage *BannerImage;
    TLabel *messageLabel;
    TPasswordEditButton *PasswordEditButton1;
    TLabel *JNU;
    TImage *Logo;

    void __fastcall RegisterNewUserNavigationButtonClick(TObject *Sender);
    void __fastcall LoginButtonClick(TObject *Sender);

private: // User declarations
public: // User declarations
    __fastcall TLoginPage(TComponent* Owner);
};

// -----
extern PACKAGE TLoginPage *LoginPage;
// -----
#endif

```

## CPP file

```
// -----
#include <fmx.h>
#pragma hdrstop

#include<iostream>
#include<fstream>
#include<string>
#include<vector>
#include<sstream>
#include <windows.h>

#include "LoginForm.h"
#include "HomePage.h"
#include "AdminRegistrationForm.h"
// -----
#pragma package(smart_init)
#pragma resource "*.fmx"

using namespace std;

TLoginPage *LoginPage;

// -----
__fastcall TLoginPage::TLoginPage(TComponent* Owner) : TForm(Owner) {
}
// -----

void __fastcall TLoginPage::RegisterNewUserNavigationButtonClick
(TObject *Sender)

{
    UserRegistrationForm->Show();
}

// -----

vector<string>parseCommaDelimitedString(string line) {
    vector<string>result;
    stringstream s_stream(line);
    while (s_stream.good()) {
        string substr;
        getline(s_stream, substr, ',');
        result.push_back(substr);
    }
    return result;
}
```

```

}

void __fastcall TLoginPage::LoginButtonClick(TObject *Sender) {
    messageLabel->Text = "";
    fstream myFile;
    myFile.open("RegisteredUsers.txt", ios::in);

    if (myFile.is_open()) {
        string line;

        while (getline(myFile, line)) {
            vector<string>parsedLine = parseCommaDelimitedString(line);
            const char* username = parsedLine.at(0).c_str();

            AnsiString editUsername = UsernameBox->Text;

            // if(editUsername == "" || " " || " ")
            // {
            //     break
            // }

            const char* usernameString = editUsername.c_str();

            if (strcmp(username, usernameString) == 0) {
                const char* password = parsedLine.at(1).c_str();

                AnsiString editPassword = PasswordBox->Text;
                const char* passwordString = editPassword.c_str();

                if (strcmp(password, passwordString) == 0) {
                    messageLabel->Text = "Success";
                    Sleep(5);
                    HomePageForm->Show();
                    // LoginPage->Close();
                    // this->Close();
                    messageLabel->Text = "";
                }
                else
                    messageLabel->Text = "Wrong password try again";
            }
            else
                messageLabel->Text = "Wrong username try again";
            // this->Close();
            // LoginPage->Close();
        }
    }
}

```

```

}

UsernameBox->Text = "";
PasswordBox->Text = "";
// messageLabel->Text="";

}

// -----

```

## Admin Registration Page

Here in admin registration page, there are 5 input text fields box where the user enters the data which is then stored in a text file. The five input that user must give is username, password, name, email, mobile no. Except this there are two more UI elements that are back button at bottom right corner and register button. And as the name suggest back button makes the application go / return to previous screen and the register button submits / records the data and closes these active window and return to the login page window. Here after pressing the register button it clears the user input from the text box.

JNU UMS | Admin Registration

**New Admin Registration**

Username:

Password:

Name:

Email:

Mobile No:

The code for the backend above that serves the application is below with some UI Elements code too.

## Header File

```
// -----
#ifndef AdminRegistrationFormH
#define AdminRegistrationFormH
// -----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Objects.hpp>
#include <FMX.Types.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Edit.hpp>
#include <FMX.ExtCtrls.hpp>
#include <FMX.Layouts.hpp>

// -----
class TUserRegistrationForm : public TForm {
__published: // IDE-managed Components
    TLabel *Name;
    TEdit *NameBox;
    TLabel *Email;
    TEdit *EmailBox;
    TLabel *Username;
    TEdit *UsernameBox;
    TLabel *Password;
    TEdit *PasswordBox;
    TLabel *MobileNo;
    TEdit *MobileNoBox;
    TLabel *Banner;
    TButton *RegisterButton;
    TImage *Logo;
    TButton *Back;

    void __fastcall RegisterButtonClick(TObject *Sender);
    void __fastcall BackClick(TObject *Sender);

private: // User declarations
public: // User declarations
    __fastcall TUserRegistrationForm(TComponent* Owner);
};

// -----
```

```
extern PACKAGE TUserRegistrationForm *UserRegistrationForm;
// -----
#endif
```

## C++ file

```
// -----

#include <fmx.h>
#include <fstream>
#pragma hdrstop

#include "AdminRegistrationForm.h"
#include "LoginForm.h"

// -----
#pragma package(smart_init)
#pragma resource "*.fmx"
TUserRegistrationForm *UserRegistrationForm;

// -----
__fastcall TUserRegistrationForm::TUserRegistrationForm(TComponent* Owner)
: TForm(Owner) {
}

// -----
void __fastcall TUserRegistrationForm::RegisterButtonClick(TObject *Sender) {
    fstream myFile;
    myFile.open("RegisteredUsers.txt", ios::app);
    if (myFile.is_open) {

        AnsiString username = UsernameBox->Text;
        AnsiString password = PasswordBox->Text;
        AnsiString name = NameBox->Text;
        AnsiString email = EmailBox->Text;
        AnsiString mobile = MobileNoBox->Text;

        myFile << username << "," << password << "," << name << "," << email <<
            "," << mobile << "\n";

        myFile.close();
        this->Close();
    }
}

// -----
```

```
void __fastcall TUserRegistrationForm::BackClick(TObject *Sender) {  
    UsernameBox->Text = "";  
    PasswordBox->Text = "";  
    NameBox->Text = "";  
    EmailBox->Text = "";  
    MobileNoBox->Text = "";  
  
    this->Close();  
    LoginPage->Show();  
}  
// -----
```

## Home Page

After logging in from the login page this is the first window that shows up and here are there 4 UI elements; all of them are buttons with some specified functions to them First one in the left is add student button it opens the new student registration window. The second button displays search students window, here the user can search for any student data recorded from the new students registration page which was eventually registered. The third button is show all students where the user can see the records of the all the students. And not just see the records but even edit the student data on the go. And last but not the least the Log out button logs out the registered user out of the system and goes back to / displays the login page again.



## Students

[Log out](#)
[Add New Students](#)
[Search Students](#)
[Show All Students](#)

The code for the following UI elements in the backed is also shown in here.

### Header File

```
// -----
#ifndef HomePageH
#define HomePageH
// -----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <FMX.Objects.hpp>

// -----
class THomePageForm : public TForm {
__published: // IDE-managed Components
    TLabel *Banner;
    TButton *AddStudents;
    TButton *SearchStudent;
    TButton *ShowStudents;
    TButton *LogOut;
```



```

TImage *Logo;
TImage *BannerImage;

void __fastcall AddStudentsClick(TObject *Sender);
void __fastcall SearchStudentClick(TObject *Sender);
void __fastcall ShowStudentsClick(TObject *Sender);
void __fastcall LogOutClick(TObject *Sender);

private: // User declarations
public: // User declarations
    __fastcall THomePageForm(TComponent* Owner);
};

// -----
extern PACKAGE THomePageForm *HomePageForm;
// -----
#endif

```

## CPP File

```

// //-----

#include <fmX.h>
#pragma hdrstop

#include "HomePage.h"
#include "StudentRegistration.h"
#include "SearchStudent.h"
#include "ShowStudents.h"
#include "LoginForm.h"

// -----
#pragma package(smart_init)
#pragma resource "*.fmx"
THomePageForm *HomePageForm;

// -----
__fastcall THomePageForm::THomePageForm(TComponent* Owner) : TForm(Owner) {
}

// -----
void __fastcall THomePageForm::AddStudentsClick(TObject *Sender) {
    StudentRegistrationForm->Show();
}

// -----
void __fastcall THomePageForm::SearchStudentClick(TObject *Sender) {

```

```

SearchStudentForm->Show();
}
// -----

void __fastcall THomePageForm::ShowStudentsClick(TObject *Sender) {
    // ShowStudentsForm->Show();
}
// -----

void __fastcall THomePageForm::LogoutClick(TObject *Sender) {
    this->Close();
    // clearabc();
}
// -----

```

## **Add New Students / New Student Registration Page**

Here in this window, there are whole lot of UI elements as the User can see 8 text input data boxes which takes the input of student's data such as name, email fathers name, mothers name, mobile number, email address registration number and year of admission. There are 3 checkboxes under the label gender which let user select the gender data for the student where they have 3 options with the check boxes as follows male, female and other. There are two more circle radio buttons under the label branch CSE and ECE which collects the branch data. As of now there are just two branches to demonstrate in this demo application. Which can be significantly increased later by just doing some minor tweaks and changes in the code. Last input data section is the course for the students which under this label there are three item checkboxes which when selected is converted then into the course data in the backend. The Two last UI elements back and submit and register button do as what is described on the each. Back button clears the current form and closes this window and return to the previous homepage window and the submit and register button submits the student data into the database and then registers them, and finally clears the form and closes the current window and goes back to the previous window namely the home page.

About the storing of data and database system here in this application. In the previous version or the earlier makes of the application the data used to be store in a text file or a csv file (Coma separated file) which is a legacy MS Excel format file type which is still heavily used tis days. Doing this and implementing filesystem storage of the data ensured support for older systems with lesser resources to spend and giving high thorough output performance. Due to lack of good hardware and system this is one of he great application or method that could have been used.

But keeping in order of high scalability we have implemented a SQL database with our application. This ensures huge amount of data to be stored and better performance on bigger level. Here the SQL database is implemented over the localhost and interbase SQL server provided by the embarcadero the same parent organization as the rad studio. Here the SQL table stored all the student data with a primary key and these data can be used with other applications too of the university as the data is stored in database files and SQL servers.

Below is the screenshot of the UI GUI Window of the Register new student window.

Below is the code for the following window. Note that these codes can't be directly run without the perfectly setting up of the environment.

Header file

```
// -----
#ifndef StudentRegistrationH
#define StudentRegistrationH
// -----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.Edit.hpp>
#include <FMX.Objects.hpp>
```

```

#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
//#include <ibase.h>
#include <FMX.Controls.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.Edit.hpp>
#include <FMX.Objects.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.Edit.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <FMX.Objects.hpp>
#include <Data.DB.hpp>
#include <Data.DBXInterBase.hpp>
#include <Data.SqlExpr.hpp>
#include <Dataspn.DBClient.hpp>
#include <Data.FMTBcd.hpp>
#include <IBX.IBDatabase.hpp>
//#include <Data.DB.hpp>
//#include <Data.DBXInterBase.hpp>
//#include <Data.FMTBcd.hpp>
//#include <Data.SqlExpr.hpp>

// -----
class TStudentRegistrationForm : public TForm {
__published: // IDE-managed Components
    TLabel *banner;
    TLabel *Name;
    TEdit *NameBox;
    TLabel *Email;
    TEdit *EmailBox;
    TLabel *FathersName;
    TEdit *FatherNameBox;
    TLabel *MothersName;
    TEdit *MotherNameBox;
    TLabel *MobileNo;
    TEdit *MobileNumberBox;
    TLabel *Address;
    TEdit *AddresssBox;
    TRadioButton *ECE;
    TButton *Submit;
    TLabel *Registration;
    TEdit *RegistrationBox;
    TCheckBox *Male;

```

```

TCheckBox *Female;
TCheckBox *Other;
TLabel *Label1;
TLabel *Label2;
TButton *Back;
TImage *Logo;
TLabel *Label3;
TEdit *YearBox;
TLabel *Label4;
TCheckBox *mtech;
TCheckBox *phd;
TCheckBox *btech;
TSQLConnection *SQLConnection1;
TSQLQuery *SQLQuery1;
TRadioButton *CSE;

void __fastcall SubmitClick(TObject *Sender);
void __fastcall CSEChange(TObject *Sender);
void __fastcall ECEChange(TObject *Sender);
void __fastcall MaleChange(TObject *Sender);
void __fastcall FemaleChange(TObject *Sender);
void __fastcall OtherChange(TObject *Sender);
void __fastcall BackClick(TObject *Sender);
void __fastcall mtechChange(TObject *Sender);
void __fastcall phdChange(TObject *Sender);
void __fastcall btechChange(TObject *Sender);

private: // User declarations
public: // User declarations
    __fastcall TStudentRegistrationForm(TComponent* Owner);
};

// -----
extern PACKAGE TStudentRegistrationForm *StudentRegistrationForm;
// -----
#endif

```

## CPP file

```

// #include <c:/program files/embarcadero/interbase/sdk/include/ibase.h>
#include <iostream>
#include <string>
// #include <ibase.h>

```

```

#include <fmx.h>
#pragma hdrstop
#include <vector>
#include <fstream>
#include <sstream>
#include <istream>
#include "StudentRegistration.h"
// -----
#pragma package(smart_init)
#pragma resource "*.fmx"
#pragma resource ("*.Windows.fmx", _PLAT_MSWINDOWS)
#pragma resource ("*.Surface.fmx", _PLAT_MSWINDOWS)

using namespace std;

#define MAX_LENGTH 300

AnsiString DATABASE = "localhost:C:\\DeleteMe\\STUDENTS.ib";
AnsiString USERNAME = "sysdba";
AnsiString PASSWORD = "masterkey";

TStudentRegistrationForm *StudentRegistrationForm;

// -----
__fastcall TStudentRegistrationForm::TStudentRegistrationForm(TComponent* Owner)
: TForm(Owner) {
}
// -----

//TSQLConnection SQLConnection1.Open;
//class Student {
//public:
//  AnsiString name, phone, email, mother, father, address, course, branch,
//  registration, gender;
//  int year_admission;
//
//  Student() {
//    name = "name";
//    phone = "6969696969";
//    email = "name@xyz.com";
//    mother = "mother";
//    father = "father";
//    address = "address";
//    course = "b.tech";
//    branch = "CSE";
//    gender = "non-binary";
//    registration = 000000;

```

```

//      year_admission = 2069;
//  }
//
//  Student(AnsiString name, AnsiString phone, AnsiString email,
//      AnsiString mother, AnsiString father, AnsiString address,
//      AnsiString course, AnsiString gender, AnsiString branch,
//      int registration, int year) {
//      name = name;
//      phone = phone;
//      email = email;
//      mother = mother;
//      father = father;
//      address = address;
//      course = course;
//      branch = branch;
//      gender = gender;
//      registration = registration;
//      year_admission = year;
//
//  }
//
//  void getdata() {
//      // comment;
//  }
//
//  void setdata() {
//      // comment
//  }
//
//};

// class File{
// public:
// vector<Student> student;
// ifstream file;
//
// File(){
// file.open("RegisteredStudent.txt", ios::in);
// int i = 0;
// while (file.good())
// {
// string input;
// file>>input;
// stringstream ss(input);
// vector<string> v;
//
// while(ss.good()){
// string substr;

```

```

// getline(ss, substr, ',');
// v.push_back(substr);
// }
//
// student[i].name = v[0];
// student[i].phone = v[1];
// student[i].email = v[2];
// student[i].mother = v[3];
// student[i].father = v[4];
// student[i].address = v[5];
// student[i].course = v[6];
// student[i].branch = v[7];
// student[i].gender = v[8];
// student[i].registration = v[9];
// student[i].year_admission = stoi(v[10]);
//
// i++;
// }
// }
//
// // #todo functions for exchange of data
//
// };
//
//
//
//

```

```

AnsiString branch, gender, course;

```

```

void __fastcall TStudentRegistrationForm::CSEChange(TObject *Sender) {
    branch = "CSE";

```

```

}
// -----

```

```

void __fastcall TStudentRegistrationForm::ECEChange(TObject *Sender) {
    branch = "ECE";
}

```

```

// -----

```

```

void __fastcall TStudentRegistrationForm::MaleChange(TObject *Sender) {
    gender = "Male";

```

```

}
// -----

```



```

void __fastcall TStudentRegistrationForm::FemaleChange(TObject *Sender) {
    gender = "Female";
}
// -----

void __fastcall TStudentRegistrationForm::OtherChange(TObject *Sender) {
    gender = "Other";
}
// -----

void __fastcall TStudentRegistrationForm::mtechChange(TObject *Sender) {
    course = "M.Tech";
}
// -----

void __fastcall TStudentRegistrationForm::phdChange(TObject *Sender) {
    course = "Ph.D";
}
// -----

void __fastcall TStudentRegistrationForm::btechChange(TObject *Sender) {
    course = "B.Tech";
}
// -----

//TSQLConnection *SQLConnection1 = new TSQLConnection();
//
//  SQLConnection1->DriverName = "InterBase";
//  SQLConnection1->Params->Values["Database"] = "C:\\DeleteMe\\STUDENTS.ib";
//
//  SQLConnection1->Open();

void __fastcall TStudentRegistrationForm::SubmitClick(TObject *Sender) {
    //  TSQLQuery *SQLQuery1 = new TSQLQuery(this);
    //
    //  SQLQuery1->SQLConnection = SQLConnection1;

    //  isc_db_handle database = 0;
    //  isc_tr_handle transaction = 0;
    //
    //  char status[20];
    //  isc_attach_database(status, 0, DATABASE, &database, 0, NULL);
    //
    //  if (status[0] == 1) {
    //      std::cout << "Failed to attach to the database" << std::endl;
    //  }
}

```

```

//
//     isc_start_transaction(status, &transaction, 1, &database, 0, NULL);
//
//     if (status[0] == 1) {
//         std::cout << "Failed to start a transaction" << std::endl;
//     }
//
//     std::cout << "Connected to the database" << std::endl;

AnsiString db = SQLConnection1->Params->Values["Database"];
SQLConnection1->Params->Values["Database"] = DATABASE;
db = SQLConnection1->Params->Values["Database"];
TTransactionDesc * trans = new TTransactionDesc;

SQLConnection1->Open() ;
SQLConnection1->BeginTransaction();
SQLQuery1->Open();
fstream myFile;
myFile.open("RegisteredStudents.txt", ios::app);

if (myFile.is_open) {

    AnsiString name = NameBox->Text;
    AnsiString email = EmailBox->Text;
    AnsiString father = FatherNameBox->Text;
    AnsiString mother = MotherNameBox->Text;
    AnsiString mobile = MobileNumberBox->Text;
    AnsiString address = AddresssBox->Text;
    AnsiString registration = RegistrationBox->Text;
    AnsiString year_admission = YearBox->Text;

    myFile << name << "," << email << "," << father << "," << mother <<
        "," << mobile << "," << address << "," << branch << "," << gender <<
        "," << registration << "," << course << "," <<
        year_admission << "\n";

    AnsiString insert;
    stringstream ss;

//
//     SQLQuery1->SQL->Text = "GRANT SELECT, INSERT, UPDATE, DELETE ON STUDENTS TO sysdba";
//     SQLQuery1->ExecSQL();
//
    ss<<"INSERT INTO STUDENTS(name, email, phone, address, mother, father, gender,
branch, course, registration, year_admission) ";

```

```

        ss<<"VALUES ('<< name << '", '<<email<<'", '<<mobile<<'", '<<address<<'",
'<<mother<<'", '<<father<<'", '<<gender<<'", '<<branch<<'", '<<course<<'",
'<<registration<<'", '<<year_admission<<');";
        ss>> insert;
        ShowMessage(insert);
        SQLQuery1->SQL->Text = insert;
        ShowMessage("all good");
        SQLQuery1->Prepared;
        SQLQuery1->ExecSQL();

        bool commit;
        SQLConnection1->Commit(*trans);
        SQLQuery1->Close();
        SQLConnection1->Close();
//        isc_dsql_execute_immediate(status, &database, &transaction, 0, insert, 1, NULL);
//
//        if (status[0] == 1) {
//            std::cout << "Failed to execute SQL statement" << std::endl;
//        }
//
//        std::cout << "Data added to the database" << std::endl;
//
//        isc_commit_transaction(status, &transaction);
//        isc_detach_database(status, &database);
//
//        if (status[0] == 1) {
//            std::cout << "Failed to close the database connection" << std::endl;
//        }
//
//        std::cout << "Disconnected from the database" << std::endl;

//after done being sql and add student data
//clearing thr input fieldds so it can work for another registration
        NameBox->Text = "";
        EmailBox->Text = "";
        FatherNameBox->Text = "";
        MotherNameBox->Text = "";
        MobileNumberBox->Text = "";
        AddresssBox->Text = "";
        RegistrationBox->Text = "";

        CSE->IsChecked = false;
        ECE->IsChecked = false;
        Male->IsChecked = false;
        Female->IsChecked = false;
        Other->IsChecked = false;
        btech->IsChecked = false;

```

```

    mtech->IsChecked = false;
    phd->IsChecked = false;

//      // Clear all TCheckBox components on the form
//      for (int i = 0; i < ComponentCount; i++)
//      {
//          if (dynamic_cast<TCheckBox *>(Components[i]))
//          {
//              ((TCheckBox *)Components[i])->Checked = false;
//          }
//      }
//
//      // Clear all TRadioButton components on the form
//      for (int i = 0; i < ComponentCount; i++)
//      {
//          if (dynamic_cast<TRadioButton *>(Components[i]))
//          {
//              ((TRadioButton *)Components[i])->Checked = false;
//          }
//      }

    myFile.close();
    this->Close();
}
}
// -----

void __fastcall TStudentRegistrationForm::BackClick(TObject *Sender) {
    NameBox->Text = "";
    EmailBox->Text = "";
    FatherNameBox->Text = "";
    MotherNameBox->Text = "";
    MobileNumberBox->Text = "";
    AddresssBox->Text = "";
    RegistrationBox->Text = "";

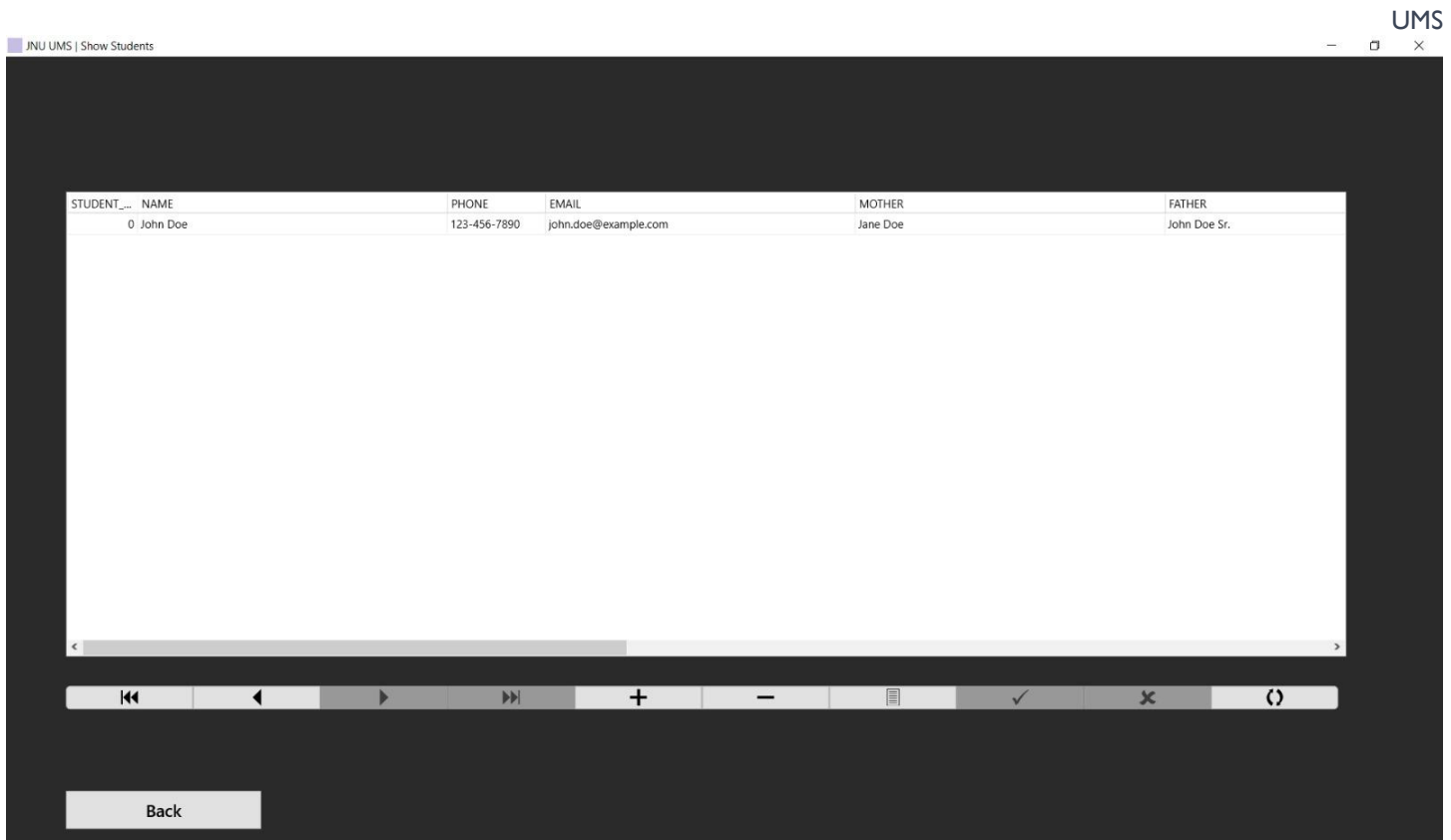
    CSE->IsChecked = false;
    ECE->IsChecked = false;
    Male->IsChecked = false;
    Female->IsChecked = false;
    Other->IsChecked = false;
    btech->IsChecked = false;
    mtech->IsChecked = false;
    phd->IsChecked = false;
}

```

```
this->Close();  
}  
// -----
```

## Show Students

Show Students or Show all Students is the window where you can see the data for the all registered students in the system. Here in table the whole database is connected where we can see the all member and every data that is registered in the database. Here we can on the go change / edit the data of the students and even delete the data. There is also a UI element, a back button and a s the name says, it closes the current window and returns to the previous screen that is homepage. There is also a bar just below the table that shows entries. This bar has some set of icons and buttons that helps the user add edit and delete the student data on the go.



The code for the following window and the backend is below:

Header file

```
//-----
#ifndef ShowStudentsH
#define ShowStudentsH
//-----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <Data.Bind.Components.hpp>
#include <Data.Bind.DBScope.hpp>
#include <Data.Bind.EngExt.hpp>
#include <Data.Bind.Grid.hpp>
#include <Data.DB.hpp>
#include <FireDAC.Comp.Client.hpp>
#include <FireDAC.Comp.DataSet.hpp>
#include <FireDAC.DApt.hpp>
```

```

#include <FireDAC.DApt.Intf.hpp>
#include <FireDAC.DatS.hpp>
#include <FireDAC.FMXUI.Wait.hpp>
#include <FireDAC.Phys.hpp>
#include <FireDAC.Phys.IB.hpp>
#include <FireDAC.Phys.IBDef.hpp>
#include <FireDAC.Phys.Intf.hpp>
#include <FireDAC.Stan.Async.hpp>
#include <FireDAC.Stan.Def.hpp>
#include <FireDAC.Stan.Error.hpp>
#include <FireDAC.Stan.Intf.hpp>
#include <FireDAC.Stan.Option.hpp>
#include <FireDAC.Stan.Param.hpp>
#include <FireDAC.Stan.Pool.hpp>
#include <FireDAC.UI.Intf.hpp>
#include <Fmx.Bind.DBEngExt.hpp>
#include <Fmx.Bind.Editors.hpp>
#include <Fmx.Bind.Grid.hpp>
#include <FMX.Grid.hpp>
#include <FMX.Grid.Style.hpp>
#include <FMX.ScrollBox.hpp>
#include <System.Bindings.Outputs.hpp>
#include <System.Rtti.hpp>
#include <Data.Bind.Controls.hpp>
#include <Fmx.Bind.Navigator.hpp>
#include <FMX.Layouts.hpp>
#include <Data.DBXInterBase.hpp>
#include <Data.SqlExpr.hpp>

//-----
class TShowStudentsForm : public TForm
{
__published:    // IDE-managed Components
    TStringGrid *StringGrid1;
    TFDConnection *EmployeeConnection;
    TFDQuery *StudentsTable;
    TBindSourceDB *BindSourceDB1;
    TBindingsList *BindingsList1;
    TLinkGridToDataSource *LinkGridToDataSourceBindSourceDB1;
    TButton *Back;
    TBindNavigator *BindNavigator1;
    TSQLConnection *SQLConnection1;
    TLabel *Banner;
    void __fastcall BackClick(TObject *Sender);
private:    // User declarations
public:    // User declarations
    __fastcall TShowStudentsForm(TComponent* Owner);
};

//-----

```

```
extern PACKAGE TShowStudentsForm *ShowStudentsForm;
//-----
#endif
```

## C++ file

```
//-----

#include <fmx.h>
#pragma hdrstop

#include "ShowStudents.h"
#include "HomePage.h"
#include <vcl.h>
#include <stdio.h>
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>

//-----

#pragma package(smart_init)
#pragma resource "*.fmx"
TShowStudentsForm *ShowStudentsForm;
//-----

__fastcall TShowStudentsForm::TShowStudentsForm(TComponent* Owner)
: TForm(Owner)
{
}
//-----

struct CSVData {
    int id;
    AnsiString name;
    double price;
};

void __fastcall TShowStudentsForm::BackClick(TObject *Sender)
{
    this->Close();
    HomePageForm->Show();
}
```



```
}
//-----
```

## Search Students

Here in this window there is a table that gives the output results of search result. The search results are based up upon the entered features that too look up into the database. As of now we just have a less search option and they aren't allowed to club together. The attributes on which the student data can be searched as of now are name, registration number branch course and year of the registration. There is a search and a back button in UI components and as the name suggest search button after entering the details when pressed would show up the data; and the back button clears the above data entry sections exits this window and switches back to previous window that is homepage.

The search feature was previously also based on file systems work like the data was searched from the data stored in csv file or from the text file after registration of the user but in the newer version of the application a SQL query is sent and then the data is processed.

**Search Student Details**

**Search Students**

- Name
- Registration
- Branch ☐ CSE ☐ ECE
- Course ☐ B.Tech ☐ M.Tech ☐ Ph.D
- Year of Registration

**Search**

**Back**

Code for the following page is below:

Header file:

```
// -----
#ifndef SearchStudentH
#define SearchStudentH
// -----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <FMX.Edit.hpp>
#include <FMX.Grid.hpp>
#include <FMX.Grid.Style.hpp>
#include <FMX.ScrollBox.hpp>
#include <System.Rtti.hpp>
#include <Data.DB.hpp>
#include <Data.DBXInterBase.hpp>
#include <Data.FMTBcd.hpp>
#include <Data.SqlExpr.hpp>
#include <Data.Bind.Components.hpp>
#include <Data.Bind.DBScope.hpp>
#include <Data.Bind.EngExt.hpp>
#include <Data.Bind.Grid.hpp>
#include <Fmx.Bind.DBEngExt.hpp>
#include <Fmx.Bind.Editors.hpp>
#include <Fmx.Bind.Grid.hpp>
#include <System.Bindings.Outputs.hpp>
#include <Web.DBWeb.hpp>
#include <Web.DBXpressWeb.hpp>
#include <Web.HTTPApp.hpp>
#include <Data.Bind.Controls.hpp>
#include <Fmx.Bind.Navigator.hpp>
#include <FMX.Layouts.hpp>

// -----
class TSearchStudentForm : public TForm {
__published: // IDE-managed Components
    TButton *Back;
    TLabel *Banner;
    TRadioButton *Name;
    TEdit *NameBox;
```

```

TRadioButton *Registration;
TRadioButton *Branch;
TCheckBox *CSE;
TCheckBox *ECE;
TEdit *RegistrationBox;
TRadioButton *Course;
TCheckBox *BTech;
TCheckBox *MTech;
TCheckBox *PhD;
TRadioButton *Year;
TEdit *YearBox;
TButton *Search;
TSQLConnection *SQLConnection1;
TSQLQueryTableProducer *SQLQueryTable1;
TSQLQuery *SQLQuery1;
TBindSourceDB *BindSourceDB1;
TBindingsList *BindingsList1;
TStringGrid *StringGrid1;
TLinkGridToDataSource *LinkGridToDataSourceBindSourceDB1;
TBindNavigator *BindNavigator1;

void __fastcall BackClick(TObject *Sender);
void __fastcall NameChange(TObject *Sender);
void __fastcall RegistrationChange(TObject *Sender);
void __fastcall BranchChange(TObject *Sender);
void __fastcall CourseChange(TObject *Sender);
void __fastcall YearChange(TObject *Sender);
void __fastcall CSEChange(TObject *Sender);
void __fastcall ECEChange(TObject *Sender);
void __fastcall BTechChange(TObject *Sender);
void __fastcall MTechChange(TObject *Sender);
void __fastcall PhDChange(TObject *Sender);
void __fastcall SearchClick(TObject *Sender);

private: // User declarations
public: // User declarations
    __fastcall TSearchStudentForm(TComponent* Owner);
};

// -----
extern PACKAGE TSearchStudentForm *SearchStudentForm;
// -----
#endif

```

CPP file:

```
// -----
```

```

#include <fmx.h>
#pragma hdrstop
#include<iostream>
#include <string>
#include <sstream>
#include "SearchStudent.h"
#include "HomePage.h"
// -----
#pragma package(smart_init)
#pragma resource "*.fmx"
using namespace std;
TSearchStudentForm *SearchStudentForm;

// -----
__fastcall TSearchStudentForm::TSearchStudentForm(TComponent* Owner)
: TForm(Owner) {
}
// -----

AnsiString name, course1, registration, branch1, year;
char s;

void __fastcall TSearchStudentForm::BackClick(TObject *Sender) {

    NameBox->Text = "";
    RegistrationBox->Text = "";
    YearBox->Text = "";

    BTech->IsChecked = false;
    MTech->IsChecked = false;
    PhD->IsChecked = false;
    Year->IsChecked = false;
    Branch->IsChecked = false;
    Course->IsChecked = false;
    Name->IsChecked = false;
    CSE->IsChecked = false;
    ECE->IsChecked = false;

    SQLQuery1->Close();
    SQLConnection1->Close();
    this->Close();
    HomePageForm->Show();
}
// -----

```

```

void __fastcall TSearchStudentForm::NameChange(TObject *Sender)
{
    s = 'n';
}
//-----

void __fastcall TSearchStudentForm::RegistrationChange(TObject *Sender)
{
    s = 'r';
}
//-----

void __fastcall TSearchStudentForm::BranchChange(TObject *Sender)
{
    s = 'b';
}
//-----

void __fastcall TSearchStudentForm::CourseChange(TObject *Sender)
{
    s = 'c';
}
//-----

void __fastcall TSearchStudentForm::YearChange(TObject *Sender)
{
    s = 'y';
}
//-----

void __fastcall TSearchStudentForm::CSEChange(TObject *Sender)
{
    branch1 ="CSE"    ;
}
//-----

void __fastcall TSearchStudentForm::ECEChange(TObject *Sender)
{
    branch1 ="ECE" ;
}
//-----

void __fastcall TSearchStudentForm::BTechChange(TObject *Sender)
{
    course1 = "B.Tech";
}
//-----

```

```

void __fastcall TSearchStudentForm::MTechChange(TObject *Sender)
{
    course1 = "M.Tech";
}
//-----

void __fastcall TSearchStudentForm::PhDChange(TObject *Sender)
{
    course1 = "Ph.D";
}
//-----

void __fastcall TSearchStudentForm::SearchClick(TObject *Sender)
{
    SqlConnection1->Params->Values["Database"] =
    "localhost:C:\\\\Users\\Avadh\\Desktop\\DeleteMe\\STUDENTS.ib";
    SqlConnection1->Open() ;
    SqlConnection1->BeginTransaction();

    SQLQuery1->SQL->Clear();
    SQLQuery1->SQL->Add("SELECT * FROM STUDENTS");

    stringstream ss;
    AnsiString str;
    label:
    switch(s)
    {
        case 'n':
            name = NameBox->Text ;
            ss<<"WHERE name = '"<<name<<"'";
            break;

        case 'r':
            registration = RegistrationBox->Text;
            ss<<"WHERE REGISTRATION = '"<<registration<<"'";
            break;

        case 'b':
            ss<<"WHERE BRANCH = '"<<branch1<<"'";
            break;

        case 'c':
            ss<<"WHERE COURSE = '"<<course1<<"'";
            break;
        case 'y':
            year = YearBox->Text;
            ss<<"WHERE YEAR_ADMISSION = '"<<year<<"'";
    }
}

```

```

        break;
    default:
        ShowMessage("Select a feature to search");
        goto label;
        break;
}

ss>>str;
SQLQuery1->SQL->Add(str);
ShowMessage(str);
SQLQuery1->Prepared;
SQLQuery1->Open();
SQLQuery1->First();

//necarjs code
//no error till now... all the graphics element have now been optimized. It 7 25 am and
everything is fucked
//
// //Clear the StringGrid1
// StringGrid1->RowCount = 1;
// StringGrid1->SelectColumn(11);
//
// //StringGrid1->ColCount = SQLQuery1->FieldCount;
// //StringGrid1->Coloumns->Clear();
//
//
//
// // Set the column captions
// for (int i = 0; i < SQLQuery1->FieldCount; i++) {
//     StringGrid1->Cells[i][0] = SQLQuery1->Fields->Fields[i]->FieldName;
// }
//
// // Loop through the retrieved data and add it to the TStringGrid
// while (!SQLQuery1->Eof) {
//     StringGrid1->RowCount++;
//     for (int i = 0; i < SQLQuery1->FieldCount; i++) {
//         StringGrid1->Cells[i][StringGrid1->RowCount - 1] = SQLQuery1->Fields->Fields[i]-
>AsString;
//     }
//     SQLQuery1->Next();
// }

NameBox->Text = "";
RegistrationBox->Text = "";
YearBox->Text = "";

BTech->IsChecked = false;

```

```
MTech->IsChecked = false;
PhD->IsChecked = false;
Year->IsChecked = false;
Branch->IsChecked = false;
Course->IsChecked = false;
Name->IsChecked = false;
CSE->IsChecked = false;
ECE->IsChecked = false;

}
//-----
```

## Environment Setup

### Environment Setup

So here we have finished a major part of the documentation. Now comes the Environment setup. This project was created in Windows Environment. So suggested OS is obviously Windows 10

First install gcc or mingw systemwide in windows.

Second Install Rad Studio C++ Builder latest version from embarcadero website.

Third Install interbase sql from embarcadero. It could be again downloaded from the same rad studio.

Now run interbase server manager and start localhost. You can connect it to any manual mode startup or automatic.

Now open Ibconsole, connect to the local server and create a database named STUDENTS.ib in any safe directory and then host it.

Add SQL table to it/ initialize it. Commands for that could be found in the documentation below or Github Repository whose link will be mentioned in the end.

Now clone the project files in working directory and open the ums.cbproj file in rad studio.

Do some checks and after that run to build the code.

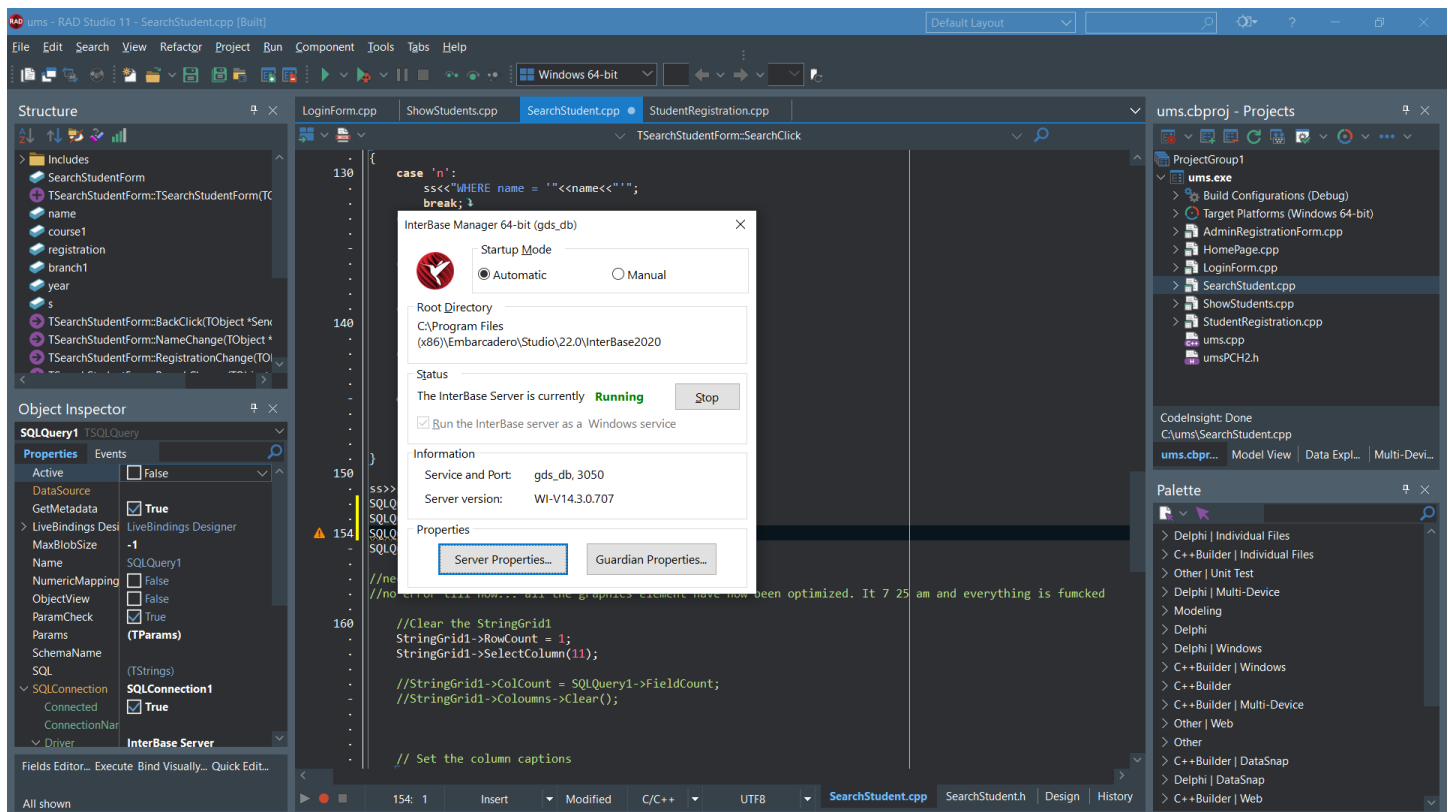


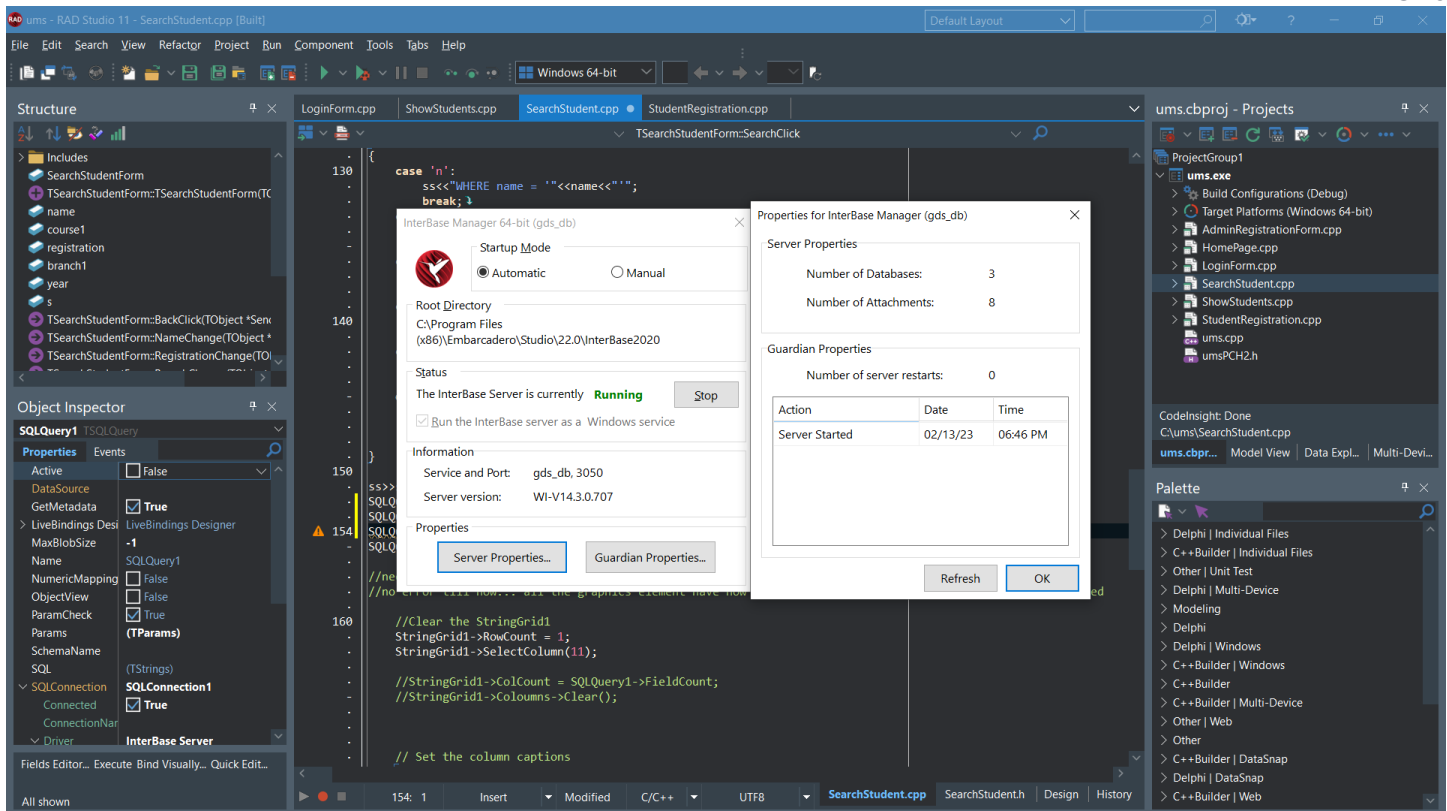
The source code is hosted on github and the application could be found in ums folder in target system name folder /debug ums.exe.

## SQL Database setup

SQL setup should be done properly, or the application wouldn't work properly. Here below are some pictures of setup and SQL code for the following is also given out.

First, we setup the server instance for the system. One can set it up to their need i.e., automatic on startup or manual start. Here we can also see the number of connections to the database. We add a new database, or we add an existing one in inter base console.





Now as we can see the setup for the network is done now, we proceed to go on creating tables in SQL making a highly scalable database and easy to maintain.

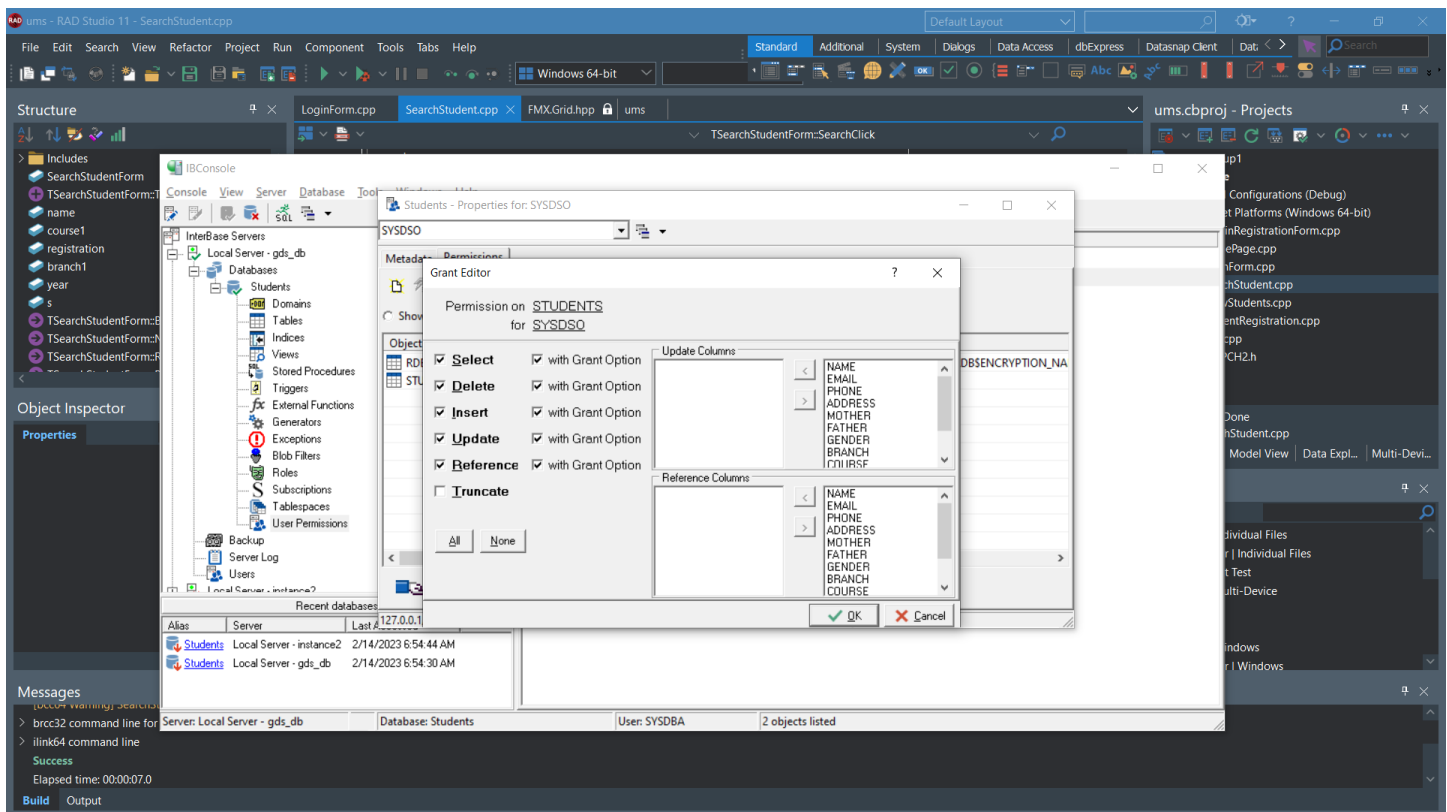
There are several benefits of setting up a database on SQL:

- **Efficient data management:** SQL databases are designed to efficiently manage large volumes of data, enabling fast retrieval, storage, and analysis of data.
- **Data consistency:** SQL databases ensure that data is consistent and accurate by enforcing rules and constraints that maintain data integrity.
- **Data security:** SQL databases provide a high level of security for data by implementing user authentication and access control, encryption, and backup and recovery mechanisms.
- **Scalability:** SQL databases can easily scale up to accommodate increasing amounts of data and users, making them ideal for growing businesses.
- **Standardization:** SQL is a standard database language, making it easy to transfer data between different database systems and ensuring compatibility across platforms.

- Analytics and Reporting: SQL databases enable powerful analytics and reporting capabilities that can help organizations make informed decisions based on their data.

Overall, setting up a database on SQL can provide a reliable, secure, and scalable solution for efficient data management and analysis.

Now the IBConsole Screenshots and the SQL code.



IBConsole

Console View Server Database Tools Windows Help

InterBase Servers

- Local Server - gds\_db
  - Databases
    - Students
      - Domains
      - Tables
      - Indices
      - Views
      - Stored Procedures
      - Triggers
      - External Functions
      - Generators
      - Exceptions
      - Blob Filters
      - Roles
      - Subscriptions
      - Tablespaces
      - User Permissions
    - Backup
    - Server Log
    - Users
  - Local Server - instance2

Recent databases

Alias	Server	Last Accessed
Students	Local Server - instance2	2/14/2023 6:54:44 AM
Students	Local Server - gds_db	2/14/2023 6:54:30 AM

Server: Local Server - gds\_db Database: Students User: SYSDBA 50 objects listed

Name	Owner	Description	Table Type	Tablespace
RDB\$CHARACTER_SETS	SYSDBA			
RDB\$CHECK_CONSTRAINTS	SYSDBA			
RDB\$COLLATIONS	SYSDBA			
RDB\$DATABASE	SYSDBA			
RDB\$DEPENDENCIES	SYSDBA			
RDB\$ENCPTIONS	SYSDBA			
RDB\$EXCEPTIONS	SYSDBA			
RDB\$FIELD_DIMENSIONS	SYSDBA			
RDB\$FIELDS	SYSDBA			
RDB\$FILES	SYSDBA			
RDB\$FILESPPACES	SYSDBA			
RDB\$FILTERS	SYSDBA			
RDB\$FORMATS	SYSDBA			
RDB\$FUNCTION_ARGUMENTS	SYSDBA			
RDB\$FUNCTIONS	SYSDBA			
RDB\$GENERATORS	SYSDBA			
RDB\$INDEX_SEGMENTS	SYSDBA			
RDB\$INDICES	SYSDBA			
RDB\$JOURNAL_ARCHIVES	SYSDBA			
RDB\$LOG_FILES	SYSDBA			
RDB\$PAGES	SYSDBA			
RDB\$PROCEDURE_PARAMETERS	SYSDBA			
RDB\$PROCEDURES	SYSDBA			
RDB\$REF_CONSTRAINTS	SYSDBA			
RDB\$RELATION_CONSTRAINTS	SYSDBA			
RDB\$RELATION_FIELDS	SYSDBA			
RDB\$RELATIONS	SYSDBA			
RDB\$ROLES	SYSDBA			
RDB\$SECURITY_CLASSES	SYSDBA			
RDB\$SUBSCRIBERS	SYSDBA			
RDB\$SUBSCRIPTIONS	SYSDBA			
RDB\$TRANSACTIONS	SYSDBA			
RDB\$TRIGGER_MESSAGES	SYSDBA			
RDB\$TRIGGERS	SYSDBA			
RDB\$TYPES	SYSDBA			
RDB\$USER_PRIVILEGES	SYSDBA			
RDB\$USERS	SYSDBA			
RDB\$VIEW_RELATIONS	SYSDBA			
STUDENTS	SYSDBA			

Database Metadata - 127.0.0.1/gds\_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib

File Edit

```

SET SQL DIALECT 3;

/* CREATE DATABASE '127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib' USER 'SYSDBA' password 'Enter_Password_here' PAGE_SIZE 4096
; */
;
COMMIT;
/*CONNECT '127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib' USER 'SYSDBA' PASSWORD 'Enter_Password_here'; */
/* ALTER DATABASE SET SYSTEM ENCRYPTION PASSWORD 'Enter_Password_here'; */
COMMIT;
/*CONNECT '127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib' USER 'SYSDBA' PASSWORD 'Enter_Password_here'; */
/* Domain definitions */
CREATE DOMAIN D_ADDRESS AS VARCHAR(50) NOT NULL;
CREATE DOMAIN D_BRANCH AS VARCHAR(5) NOT NULL;
CREATE DOMAIN D_COURSE AS VARCHAR(7) NOT NULL;
CREATE DOMAIN D_EMAIL AS VARCHAR(20) CHARACTER SET UTF8 NOT NULL;
CREATE DOMAIN D_FATHER AS VARCHAR(25) NOT NULL;
CREATE DOMAIN D_GENDER AS VARCHAR(8) NOT NULL;
CREATE DOMAIN D_ID AS INTEGER
    DEFAULT -1 NOT NULL;
CREATE DOMAIN D_MOTHER AS VARCHAR(25) CHARACTER SET UTF8 NOT NULL;
CREATE DOMAIN D_PHONE AS VARCHAR(13) CHARACTER SET UTF8 NOT NULL;
CREATE DOMAIN D_REGISTRATION AS VARCHAR(15) NOT NULL;
CREATE DOMAIN D_STUDENT_NAME AS VARCHAR(25) CHARACTER SET UTF8
    DEFAULT 'NAME' NOT NULL;
CREATE DOMAIN D_YEAR_ADMISSION AS VARCHAR(4) NOT NULL;

/* Table: STUDENTS, Owner: SYSDBA */
CREATE TABLE STUDENTS
(
    NAME    D_STUDENT_NAME NOT NULL,
    EMAIL   D_EMAIL NOT NULL,
    PHONE   D_PHONE NOT NULL,
    ADDRESS D_ADDRESS NOT NULL,
    MOTHER  D_MOTHER NOT NULL,
    FATHER  D_FATHER NOT NULL,
    GENDER  D_GENDER NOT NULL,
    BRANCH  D_BRANCH NOT NULL,
    COURSE  D_COURSE NOT NULL,
    REGISTRATION    D_REGISTRATION NOT NULL,
    YEAR_ADMISSION  D_YEAR_ADMISSION NOT NULL
);

/* Subscriptions */
SET AUTODDL OFF;
1: 1

```

```

Database Metadata - 127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib
File Edit

CREATE DOMAIN D_REGISTRATION AS VARCHAR(15) NOT NULL;
CREATE DOMAIN D_STUDENT_NAME AS VARCHAR(25) CHARACTER SET UTF8
    DEFAULT 'NAME' NOT NULL;
CREATE DOMAIN D_YEAR_ADMISSION AS VARCHAR(4) NOT NULL;

/* Table: STUDENTS, Owner: SYSDBA */
CREATE TABLE STUDENTS
(
    NAME      D_STUDENT_NAME NOT NULL,
    EMAIL     D_EMAIL NOT NULL,
    PHONE     D_PHONE NOT NULL,
    ADDRESS   D_ADDRESS NOT NULL,
    MOTHER    D_MOTHER NOT NULL,
    FATHER    D_FATHER NOT NULL,
    GENDER    D_GENDER NOT NULL,
    BRANCH    D_BRANCH NOT NULL,
    COURSE    D_COURSE NOT NULL,
    REGISTRATION D_REGISTRATION NOT NULL,
    YEAR_ADMISSION D_YEAR_ADMISSION NOT NULL
);

/* Subscriptions */
SET AUTODDL OFF;
SET AUTODDL ON;
COMMIT;
COMMIT WORK;

/* Stored procedures */
COMMIT WORK;

/* Grant Roles for this database */

/* Grant permissions for this database */
GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES, DECRYPT ON STUDENTS TO PUBLIC;
GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES, DECRYPT ON STUDENTS TO SYSDSO WITH GRANT OPTION;

/* Meta data descriptions. This syntax requires InterBase 2020 or higher. Some tables require ODS18 and higher */

```

## SQL Code

```

SET SQL DIALECT 3;

/* CREATE DATABASE '127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib' USER
'SYSDBA' password 'Enter_Password_here' PAGE_SIZE 4096
WITH ADMIN OPTION
; */
COMMIT;
/* Users in Database (passwords need to be added)*/
/* CREATE USER SYSDSO SET PASSWORD 'Enter_Password_here' ;*/
/*CONNECT '127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib' USER 'SYSDSO'
PASSWORD 'Enter_Password_here'; */
/* ALTER DATABASE SET SYSTEM ENCRYPTION PASSWORD 'Enter_Password_here';*/
CREATE ENCRYPTION DOPES FOR DES WITH LENGTH 56 BITS INIT_VECTOR NULL PAD NULL;
/* CREATE ENCRYPTION Neeraj FOR DES WITH LENGTH 56 BITS INIT_VECTOR NULL PAD NULL PASSWORD
'Enter_Password_here';*/
GRANT ENCRYPT ON ENCRYPTION DOPES TO SYSDBA;
GRANT ENCRYPT ON ENCRYPTION Neeraj TO SYSDBA;
COMMIT;
/*CONNECT '127.0.0.1/gds_db:C:\Users\Avadh\Desktop\DeleteMe\STUDENTS.ib' USER 'SYSDBA'
PASSWORD 'Enter_Password_here'; */
/* Domain definitions */
CREATE DOMAIN D_ADDRESS AS VARCHAR(100) NOT NULL;
CREATE DOMAIN D_BRANCH AS VARCHAR(10) NOT NULL;
CREATE DOMAIN D_COURSE AS VARCHAR(10) NOT NULL;

```

```

CREATE DOMAIN D_EMAIL AS VARCHAR(50) CHARACTER SET UTF8 NOT NULL;
CREATE DOMAIN D_FATHER AS VARCHAR(50) NOT NULL;
CREATE DOMAIN D_GENDER AS VARCHAR(10) NOT NULL;
CREATE DOMAIN D_ID AS INTEGER
    DEFAULT -1 NOT NULL;
CREATE DOMAIN D_MOTHER AS VARCHAR(50) CHARACTER SET UTF8 NOT NULL;
CREATE DOMAIN D_PHONE AS VARCHAR(12) CHARACTER SET UTF8 NOT NULL;
CREATE DOMAIN D_REGISTRATION AS VARCHAR(15) NOT NULL;
CREATE DOMAIN D_STUDENT_NAME AS VARCHAR(50) CHARACTER SET UTF8
    DEFAULT 'NAME' NOT NULL;
CREATE DOMAIN D_YEAR_ADMISSION AS VARCHAR(4) NOT NULL;

/* Table: STUDENTS, Owner: SYSDBA */
CREATE TABLE STUDENTS
(
    NAME      D_STUDENT_NAME NOT NULL,
    EMAIL     D_EMAIL NOT NULL,
    PHONE     D_PHONE NOT NULL,
    ADDRESS   D_ADDRESS NOT NULL,
    MOTHER    D_MOTHER NOT NULL,
    FATHER    D_FATHER NOT NULL,
    GENDER    D_GENDER NOT NULL,
    BRANCH    D_BRANCH NOT NULL,
    COURSE    D_COURSE NOT NULL,
    REGISTRATION D_REGISTRATION NOT NULL,
    YEAR_ADMISSION D_YEAR_ADMISSION NOT NULL
);

/* Subscriptions */
SET AUTODDL OFF;
SET AUTODDL ON;
COMMIT;
COMMIT WORK;

/* Stored procedures */

COMMIT WORK;

/* Grant Roles for this database */

/* Grant permissions for this database */

GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES, DECRYPT ON STUDENTS TO PUBLIC;
GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES, DECRYPT ON STUDENTS TO SYSDSO WITH GRANT
OPTION;

/* Meta data descriptions. This syntax requires InterBase 2020 or higher. Some tables
require ODS18 and higher */

```

## Conclusion/Summary

What better could be done with this project? Well adding grading system to it and management of fees along with the record go hand in hand. These would be really nice addition to work on.

As to sum up everything the University Management System project is an example of how OOP concepts can be used to develop complex software systems. The project is implemented in C++ using RAD Studio, which provides a visual programming environment that allows developers to create user interfaces using drag and drop tools. The system consists of several modules, including student admissions, enrollments, class schedules, academic records, and other related administrative tasks. The system is designed to be modular and scalable, allowing for easy expansion and customization.

There are several benefits of setting up a database on SQL:

- **Efficient data management:** SQL databases are designed to efficiently manage large volumes of data, enabling fast retrieval, storage, and analysis of data.
- **Data consistency:** SQL databases ensure that data is consistent and accurate by enforcing rules and constraints that maintain data integrity.
- **Data security:** SQL databases provide a high level of security for data by implementing user authentication and access control, encryption, and backup and recovery mechanisms.
- **Scalability:** SQL databases can easily scale up to accommodate increasing amounts of data and users, making them ideal for growing businesses.
- **Standardization:** SQL is a standard database language, making it easy to transfer data between different database systems and ensuring compatibility across platforms.
- **Analytics and Reporting:** SQL databases enable powerful analytics and reporting capabilities that can help organizations make informed decisions based on their data.

Overall, setting up a database on SQL can provide a reliable, secure, and scalable solution for efficient data management and analysis.

## Contact Information and other links

**GitHub Repository link:** <https://github.com/blacklistperformer/University-Management-System-In-c-gui.git>

**Rad Studio:** <https://www.embarcadero.com/products/rad-studio>

**Interbase:** <https://www.embarcadero.com/products/interbase>

### Contributors:

**Avadhesh Kumar** 🙋🙋🙋

Instagram [https://www.instagram.com/avadh\\_ak\\_/](https://www.instagram.com/avadh_ak_/)

Email [avadheshkumarajay@gmail.com](mailto:avadheshkumarajay@gmail.com)

Linkedin <https://www.linkedin.com/in/avadhesh-kumar-70b2681b2/>

GitHub <https://github.com/Avadhak47>

**Neeraj Nikhil Roy** 🙋🙋🙋

Instagram <https://www.instagram.com/blacklistperformer/>

Email [neerajroy06502@gmail.com](mailto:neerajroy06502@gmail.com)

Linkedin <https://www.linkedin.com/in/neeraj-roy-556968192/>

GitHub <https://github.com/blacklistperformer>

**Kshama Meena** ❤️❤️❤️

Instagram [https://www.instagram.com/meenakshama\\_11\\_/](https://www.instagram.com/meenakshama_11_/)

Email [kshamameena7@gmail.com](mailto:kshamameena7@gmail.com)

Linkedin <https://www.linkedin.com/in/kshama-meena-1851a8207/>

GitHub <https://github.com/kshamameena>

**Chirag Beniwal** 😊😊😊

Instagram <https://www.instagram.com/chxbeni/>

Email [chiragbeniwal498@gmail.com](mailto:chiragbeniwal498@gmail.com)

Linkedin <https://www.linkedin.com/in/chirag-beniwal-08691a1b4/>

//hue hue hue