UNIVERSIDAD MARIANO GALVEZ DE

GUATEMALA EXTENSION PETÉN

CURSO:

PROGRAMACIÓN I

CATEDRÁTICO:

LUIS ANTONIO GARCÍA AGUIRRE

PROYECTO

SISTEMA DE ESTUDIANTES CON CURSOS

ALUMNOS:

FREDY ELIAN MORÁN RAMOS: 1690-24-11782

MARIO RIGOBERTO CORTEZ RODAS: 1690-22-12638

LEYSER ARIEL PEÑA ORELLANA: 1690-24-20172

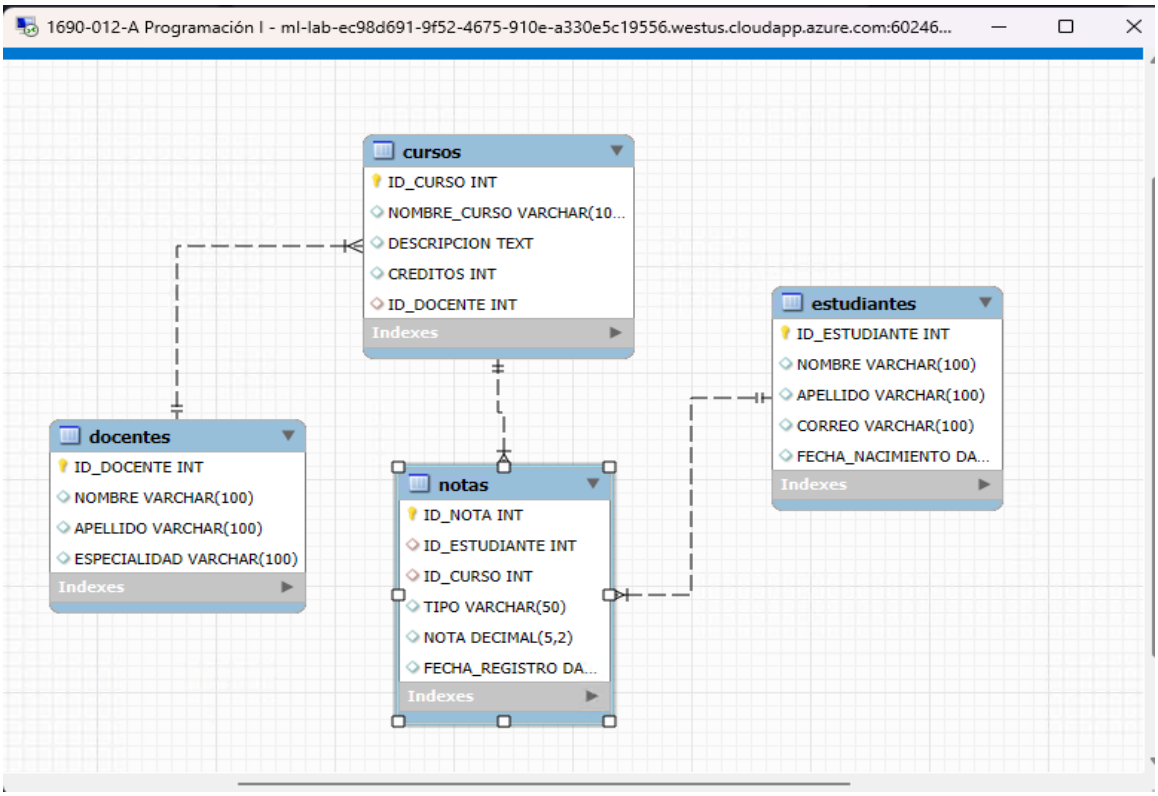
FACULTAD: INGENIERÍA EN SISTEMAS DE INFORMACIÓN Y CIENCIAS DE LA COMPUTACION

SANTA ELENA DE LA CRUZ, FLORES, PETÉN, MAYO 2025

ÍNDICE

2. SISTEMA DE ESTUDIANTES CON CURSOS

2.1 Modelo Entidad Relación



2.2 Código del sistema

2.2.1 Librerías

- El desarrollo del sistema se basa en librerías tanto estándar como personalizadas para facilitar la conexión con la base de datos, el manejo de entradas del usuario y el control general del flujo del programa:

```
1  #include "MySQLConexion.h"
2  #include "EloquentORM.h"
3  #include <iostream>
4  #include <string>
5  #include <limits>
6  #include <regex>
7
8  using namespace std;
9
```

2.2.2 Funciones

- El sistema implementa funciones auxiliares que mejoran la interacción del usuario y refuerzan la validación de datos durante la ejecución del programa. Estas funciones están diseñadas para mantener una interfaz limpia, eficiente y amigable:

```cpp
// =================Funciones=====================
/*
 * @brief Función para limpiar la pantalla.
 */
void limpiar_pantalla() {
    cout << "\033[2J\033[H";
}
/*
 * @brief Función para validar el id.
 * @param id Referencia al id a validar.
 */
void validar_id(int& id) {
    string entrada;
    regex patron("^[1-9][0-9]*$"); // Solo números enteros positivos, sin ceros a la izquierda
    while (true) {
        getline(cin, entrada);
        if (regex_match(entrada, patron)) {
            id = stoi(entrada);
            break;
        }
        if (cin.fail() || id <= 0) {
            // Si la entrada no es un número válido o es menor o igual a 0
            cout << "ID incorrecto. Ingrese un ID valido: " << endl;
            cin.clear(); // Limpiar el estado de error
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignorar la entrada inválida
        } else {
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Limpiar el buffer de entrada
            break;
        }
    }
}

enum Colors {
    BLACK = 0,
    BLUE = 1,
    GREEN = 2,
    CYAN = 3,
    RED = 4,
    MAGENTA = 5,
    BROWN = 6,
    LGREY = 7,
    DGREY = 8,
    LBLUE = 9,
    LGREEN = 10,
    LCYAN = 11,
    LRED = 12,
    LMAGENTA = 13,
    YELLOW = 14,
    WHITE = 15
};
void Color(int Background, int Text){ // function to change colors
 HANDLE Console = GetStdHandle(STD_OUTPUT_HANDLE);
 int    New_Color= Text + (Background * 16);
 SetConsoleTextAttribute(Console, New_Color)
}
```

2.2.3 Clases y Métodos

- Se crearon 4 clases para la manipulación y representación de las cuatro tablas en la base de datos, cada una con sus respectivos métodos CRUD: Create. -crear-, Read -leer-, Upgrade -actualizar y Delete -eliminar-.
- Cada clase contiene dos atributos, uno es una Referencia a la conexión con la base de datos MySQL y el otro es un Objeto de mapeo relacional que facilita el trabajo con registros de la tabla

```
65    // ================ ENTIDADES ====================
66    /*
67    @brief Clase que representa un estudiante
68    *
69    * Permite realizar operaciones CRUD sobre la tabla de estudiantes.
70    */
71    class Estudiante {
72        MySQLConexion& conn;
73        EloquentORM orm;
74    public:
75        /*
76         * @brief Constructor de la clase Estudiante.
77         *
78         * Inicializa la conexión a la base de datos y la tabla de estudiantes.
79         *
80         * @param conexion Referencia a la conexión MySQL.
81         */
82        Estudiante(MySQLConexion& conexion)
83            : conn(conexion), orm(conexion, "ESTUDIANTES", {"NOMBRE", "APELLIDO", "CORREO", "FECHA_NACIMIENTO"}) {}
84
85        /*
86         * @brief Crea un nuevo estudiante.
87         *
88         * Solicita al usuario los datos del estudiante y lo inserta en la base de datos.
89         */
90        void crear_estudiante() {
91            // Crear un nuevo objeto EloquentORM para la tabla estudiantes
92            EloquentORM alumno(conn, "estudiantes", {"NOMBRE", "APELLIDO", "CORREO", "FECHA_NACIMIENTO"});
93            // Solicitar datos al usuario
94            // Variables para almacenar los datos
95            string nombre, apellido, correo, fecha;
96            cout << "Nombre: "; cin >> nombre;
97            cout << "Apellido: "; cin >> apellido;
98            cout << "Correo: "; cin >> correo;
99            cout << "Fecha de nacimiento (YYYY-MM-DD): "; cin >> fecha;
00
01            alumno.set("NOMBRE", nombre);
02            alumno.set("APELLIDO", apellido);
```

```
103             alumno.set("CORREO", correo);
104             alumno.set("FECHA_NACIMIENTO", fecha);
105
106             cout << (alumno.create() ? "Alumno creado.\n" : "Error al crear el alumno.\n");
107         }
108         /*
109          * @brief Lee todos los estudiantes de la base de datos.
110          *
111          * Muestra en pantalla los datos de todos los estudiantes registrados.
112          */
113         void leer_estudiantes() {
114             // Obtener todos los registros de la tabla estudiantes
115             auto lista = orm.getAll();
116             for (auto& reg : lista) {
117                 cout << "ID: " << reg["id"] << ", Nombre: " << reg["NOMBRE"]
118                      << ", Apellido: " << reg["APELLIDO"] << ", Correo: " << reg["CORREO"]
119                      << ", Fecha: " << reg["FECHA_NACIMIENTO"] << endl;
120             }
121             cout << "Presiona Enter para continuar" << endl;
122         }
123         /*
124          * @brief Actualiza los datos de un estudiante.
125          *
126          * Solicita al usuario el ID del estudiante y los nuevos datos, y actualiza la base de datos.
127          */
128         void actualizar_estudiante() {
129             EloquentORM alumno(conn, "estudiantes", {"id", "NOMBRE", "APELLIDO", "CORREO", "FECHA_NACIMIENTO"});
130             // Solicitar el ID del estudiante a actualizar
131             int id;
132             cout << "ID del estudiante: "; cin >> id; validar_id(id);
133             string nombre, apellido, correo, fecha;
134             cout << "Nuevo Nombre: "; cin >> nombre;
135             cout << "Nuevo Apellido: "; cin >> apellido;
136             cout << "Nuevo Correo: "; cin >> correo;
137             cout << "Nueva Fecha (YYYY-MM-DD): "; cin >> fecha;
139              alumno.set("id", to_string(id));
140              alumno.set("NOMBRE", nombre);
141              alumno.set("APELLIDO", apellido);
142              alumno.set("CORREO", correo);
143              alumno.set("FECHA_NACIMIENTO", fecha);
144
145              cout << (alumno.update() ? "Estudiante actualizado.\n" : "Error al actualizar estudiante.\n");
146          }
147          /*
148           * @brief Elimina un estudiante de la base de datos.
149           *
150           * Solicita al usuario el ID del estudiante a eliminar y lo elimina de la base de datos.
151           */
152          void eliminar_estudiante() {
153              EloquentORM alumno(conn,"estudiantes",{"NOMBRE", "APELLIDO", "CORREO","FECHA_NACIMIENTO"});
154              // Solicitar el ID del estudiante a eliminar
155              int id;
156              cout << "Ingrese el id de estudiante que desea eliminar: \n";
```

```
157             cin >> id;
158             // Validar el ID ingresado
159             if (cin.fail() || id <= 0) {
160                 cout << "ID incorrecto. Ingrese un ID valido: " << endl;
161                 cin.clear(); // Limpiar el estado de error
162                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignorar la entrada inválida
163             }
164             else if (alumno.find(id)) {
165             cout << "Se elimino al alumno: " << alumno.get("NOMBRE") << endl;
166             alumno.remove();
167             }
168             else {
169                 cout << "No se encontro un estudiante con el ID: " << id << endl;
170                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Limpiar el buffer de entrada
171             }
172         }
173     };
174     /*
175     * @brief Clase que representa un docente
176     */
177     class Docente {
178         MySQLConexion& conn;
179         EloquentORM orm;
180     public:
181         Docente(MySQLConexion& conexion)
182             : conn(conexion), orm(conexion, "DOCENTES", {"NOMBRE", "ESPECIALIDAD"}) {}
183 
184         /*
185          * @brief Crea un nuevo docente.
186          *
187          * Solicita al usuario los datos del docente y lo inserta en la base de datos.
188          */
189         void crear_docente() {
190             EloquentORM profesor(conn, "DOCENTES", {"NOMBRE", "APELLIDO", "ESPECIALIDAD"});
191             // Solicitar datos al usuario
192             string nombre, apellido, especialidad;
193             cout << "Nombre: "; getline(cin, nombre);
194             cout << "Apellido: "; getline(cin, apellido);
195             cout << "Especialidad: "; cin >> especialidad;
196 
197             profesor.set("NOMBRE", nombre);
198             profesor.set("APELLIDO", apellido);
199             profesor.set("ESPECIALIDAD", especialidad);
200 
```

```
201             cout << (profesor.create() ? "Docente creado.\n" : "Error al crear docente.\n");
202         }
203         /*
204          * @brief Lee todos los docentes de la base de datos.
205          *
206          * Muestra en pantalla los datos de todos los docentes registrados.
207          */
208         void leer_docentes() {
209             // Obtener todos los registros de la tabla docentes
210             auto lista = orm.getAll();
211             for (auto& reg : lista) {
212                 cout << "ID: " << reg["id"] << ", Nombre: " << reg["NOMBRE"]
213                      << ", Apellido: " << reg["APELLIDO"]
214                      << ", Especialidad: " << reg["ESPECIALIDAD"] << endl;
215             }
216         }
217         /*
218          * @brief Actualiza los datos de un docente.
219          *
220          * Solicita al usuario el ID del docente y los nuevos datos, y actualiza la base de datos.
221          */
222         void actualizar_docente() {
223             EloquentORM profesor(conn, "docentes", {"id", "NOMBRE", "APELLIDO", "ESPECIALIDAD"});
224             // Solicitar el ID del docente a actualizar
225             int id;
226             cout << "ID del docente: "; cin >> id; validar_id(id);
227             string nombre, apellido, especialidad;
228             cout << "Nuevo Nombre: "; cin >> nombre;
229             cout << "Nuevo Apellido: "; cin >> apellido;
230             cout << "Nueva Especialidad: "; getline(cin, especialidad);
231
232          profesor.set("id", to_string(id));
233          profesor.set("NOMBRE", nombre);
234          profesor.set("APELLIDO", apellido);
235          profesor.set("ESPECIALIDAD", especialidad);
237             cout <<  profesor.update() ? "Docente actualizado.\n" : "Error al actualizar docente.\n";
238         }
239         /*
240          * @brief Elimina un docente de la base de datos.
241          *
242          * Solicita al usuario el ID del docente a eliminar y lo elimina de la base de datos.
243          */
244         void eliminar_docente() {
245             EloquentORM profesor(conn,"docentes",{"NOMBRE", "APELLIDO", "ESPECIALIDAD"});
246             // Solicitar el ID del docente a eliminar
247             int id;
248             cout << "Ingrese el id de docente que desea eliminar: \n";
249             cin >> id;
```

```
250             // Validar el ID ingresado
251             if (cin.fail() || id <= 0) {
252                 cout << "ID incorrecto. Ingrese un ID valido: " << endl;
253                 cin.clear(); // Limpiar el estado de error
254                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignorar la entrada inválida
255             }
256             else if (profesor.find(id)) {
257             cout << "Se elimino al docente: " << profesor.get("NOMBRE") << endl;
258             profesor.remove();
259             }
260             else {
261                 cout << "No se encontro un docente con el ID: " << id << endl;
262                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Limpiar el buffer de entrada
263             }
264         }
265     };
266     /*
267     *
268     * @brief Clase que representa un curso
269     */
270     class Curso {
271         MySQLConexion& conn;
272         EloquentORM orm;
273     public:
274         /*
275          * @brief Constructor de la clase Curso.
276          *
277          * Inicializa la conexión a la base de datos y la tabla de cursos.
278          *
279          * @param conexion Referencia a la conexión MySQL.
280          */
281         Curso(MySQLConexion& conexion)
282             : conn(conexion), orm(conexion, "CURSOS", {"NOMBRE", "CREDITOS", "ID_DOCENTE"}) {}
283         /*
284          * @brief Crea un nuevo curso.
285          *
286          * Solicita al usuario los datos del curso y lo inserta en la base de datos.
287          */
288         void crear_curso() {
289             EloquentORM materia(conn, "cursos", {"NOMBRE_CURSO", "CREDITOS", "DESCRIPCION", "ID_DOCENTE"});
290             // Solicitar datos al usuario
291             string nombre, descripcion;
292             int creditos, idDocente;
293             cout << "Nombre del curso: "; getline(cin, nombre);
294             cout << "Creditos: "; cin >> creditos;
295             cout << "Descripcion: "; getline(cin, descripcion);
296             cout << "ID del docente: "; cin >> idDocente;
297
```

```
298             materia.set("NOMBRE_CURSO", nombre);
299             materia.set("CREDITOS", to_string(creditos));
300             materia.set("DESCRIPCION", descripcion);
301             materia.set("ID_DOCENTE", to_string(idDocente));
302 
303             cout << (materia.create() ? "Curso creado.\n" : "Error al crear curso.\n");
304         }
305         /*
306          * @brief Lee todos los cursos de la base de datos.
307          *
308          * Muestra en pantalla los datos de todos los cursos registrados.
309          */
310         void leer_cursos() {
311             // Obtener todos los registros de la tabla cursos
312             auto lista = orm.getAll();
313             for (auto& reg : lista) {
314                 cout << "ID: " << reg["id"] << ", Nombre: " << reg["NOMBRE_CURSO"]
315                      << ", Creditos: " << reg["CREDITOS"]
316                      << ", Descripcion: " << reg["DESCRIPCION"]
317                      << ", ID Docente: " << reg["ID_DOCENTE"] << endl;
318             }
319         }
320         /*
321          * @brief Actualiza los datos de un curso.
322          *
323          * Solicita al usuario el ID del curso y los nuevos datos, y actualiza la base de datos.
324          */
325         void actualizar_curso() {
326             EloquentORM materia(conn, "cursos", {"id", "NOMBRE_CURSO", "CREDITOS", "DESCRIPCION", "ID_DOCENTE"});
327             // Solicitar el ID del curso a actualizar
328             int id;
329             cout << "ID del curso: "; cin >> id; validar_id(id);
330             string nombre, descripcion;
331             int creditos, idDocente;
332             cout << "Nuevo Nombre: "; getline(cin, nombre);
333             cout << "Nueva Descripcion: "; getline(cin, descripcion);
334             cout << "Creditos: "; cin >> creditos;
335             cout << "ID Docente: "; cin >> idDocente;
336 
337             materia.set("id", to_string(id));
338             materia.set("NOMBRE_CURSO", nombre);
339             materia.set("DESCRIPCION", descripcion);
340             materia.set("CREDITOS", to_string(creditos));
341             materia.set("ID_DOCENTE", to_string(idDocente));
342 
343             cout << (materia.update() ? "Curso actualizado.\n" : "Error al actualizar curso.\n");
344         }
```

```
345         /*
346          * @brief Elimina un curso de la base de datos.
347          *
348          * Solicita al usuario el ID del curso a eliminar y lo elimina de la base de datos.
349          */
350         void eliminar_curso() {
351             EloquentORM materia(conn,"cursos",{"NOMBRE_CURSO", "CREDITOS", "DESCRIPCION", "ID_DOCENTE"});
352             int id;
353             cout << "Ingrese el id de curso que desea eliminar: \n";
354             cin >> id;
355             // Validar el ID ingresado
356             if (cin.fail() || id <= 0) {
357                 cout << "ID incorrecto. Ingrese un ID valido: " << endl;
358                 cin.clear(); // Limpiar el estado de error
359                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignorar la entrada inválida
360             }
361             else if (materia.find(id)) {
362             cout << "Se elimino al curso: " << materia.get("NOMBRE_CURSO") << endl;
363             materia.remove();
364             }
365             else {
366                 cout << "No se encontro un curso con el ID: " << id << endl;
367                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Limpiar el buffer de entrada
368             }
369         }
370     };
371     /*
372     *
373     * @brief Clase que representa una nota
374     */
375     class Nota {
376         MySQLConexion& conn;
377         EloquentORM orm;
378     public:
379         Nota(MySQLConexion& conexion)
380             : conn(conexion), orm(conexion, "NOTAS", {"ID_ESTUDIANTE", "ID_CURSO", "CALIFICACION"}) {}
381         /*
382          * @brief Crea una nueva nota.
383          *
384          * Solicita al usuario los datos de la nota y lo inserta en la base de datos.
385          */
386         void crear_nota() {
387             // Crear un nuevo objeto EloquentORM para la tabla notas
388             EloquentORM nota(conn, "NOTAS", {"ID_ESTUDIANTE", "ID_CURSO", "TIPO", "NOTA", "FECHA_REGISTRO"});
389             // Solicitar datos al usuario
390             int idEstudiante, idCurso;
391             double calificacion;
```

```
392             string fecha, tipo;
393             cout << "ID Estudiante: "; cin >> idEstudiante;
394             cout << "ID Curso: "; cin >> idCurso;
395             cout << "Tipo (PRESENCIAL/ONLINE): "; cin >> tipo;
396             cout << "Calificacion: "; cin >> calificacion;
397             cout << "Fecha de registro (YYYY-MM-DD): "; cin >> fecha;
398
399             nota.set("ID_ESTUDIANTE", to_string(idEstudiante));
400             nota.set("ID_CURSO", to_string(idCurso));
401             nota.set("NOTA", to_string(calificacion));
402             nota.set("TIPO", tipo);
403             nota.set("FECHA_REGISTRO", fecha);
404
405             cout << (nota.create() ? "Nota registrada.\n" : "Error al registrar nota.\n");
406         }
407         /*
408          * @brief Lee todas las notas de la base de datos.
409          *
410          * Muestra en pantalla los datos de todas las notas registradas.
411          */
412         void leer_notas() {
413             // Obtener todos los registros de la tabla notas
414             auto lista = orm.getAll();
415             for (auto& reg : lista) {
416                 cout << "ID: " << reg["id"] << ", Estudiante: " << reg["ID_ESTUDIANTE"]
417                      << ", Curso: " << reg["ID_CURSO"]
418                      << ", Calificacion: " << reg["NOTA"] << endl
419                      << "Tipo: " << reg["TIPO"] << ", Fecha: " << reg["FECHA_REGISTRO"] << endl;
420             }
421         }
422         /*
423          * @brief Actualiza los datos de una nota.
424          *
425          * Solicita al usuario el ID de la nota y los nuevos datos, y actualiza la base de datos.
426          */
427         void actualizar_nota() {
428             EloquentORM nota(conn, "NOTAS", {"id", "ID_ESTUDIANTE", "ID_CURSO", "TIPO", "NOTA", "FECHA_REGISTRO"});
429             // Solicitar el ID de la nota a actualizar
430             int id;
431             cout << "ID de la nota: "; cin >> id; validar_id(id);
432             // Solicitar nuevos datos
433             int idEstudiante, idCurso;
434             double calificacion;
435             string fecha, tipo;
436             cout << "Nuevo ID Estudiante: "; cin >> idEstudiante;
437             cout << "Nuevo ID Curso: "; cin >> idCurso;
438             cout << "Nuevo Tipo (PRESENCIAL/ONLINE): "; cin >> tipo;
```

```
439             cout << "Nueva Calificacion: "; cin >> calificacion;
440             cout << "Nueva Fecha (YYYY-MM-DD): "; cin >> fecha;
441 
442             nota.set("id", to_string(id));
443             nota.set("ID_ESTUDIANTE", to_string(idEstudiante));
444             nota.set("ID_CURSO", to_string(idCurso));
445             nota.set("TIPO", tipo);
446             nota.set("NOTA", to_string(calificacion));
447             nota.set("FECHA_REGISTRO", fecha);
448 
449             cout << (nota.update() ? "Nota actualizada.\n" : "Error al actualizar nota.\n");
450         }
451         /*
452          * @brief Elimina una nota de la base de datos.
453          *
454          * Solicita al usuario el ID de la nota a eliminar y lo elimina de la base de datos.
455          */
456         void eliminar_nota() {
457             EloquentORM nota(conn, "NOTAS", {"ID_ESTUDIANTE", "ID_CURSO", "TIPO", "NOTA", "FECHA_REGISTRO"});
458             // Solicitar el ID de la nota a eliminar
459             int id;
460             cout << "ID de la nota: "; cin >> id;
461             // Validar el ID ingresado
462             if (cin.fail() || id <= 0) {
463                 cout << "ID incorrecto. Ingrese un ID valido: " << endl;
464                 cin.clear(); // Limpiar el estado de error
465                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignorar la entrada inválida
466             }
467             else if (nota.find(id)) {
468             cout << "Se elimino la nota del estudiante con ID: " << nota.get("ID_ESTUDIANTE") << endl;
469             nota.remove();
470             }
471             else {
472                 cout << "No se encontro un estudiante con el ID: " << id << endl;
473                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Limpiar el buffer de entrada
474             }
475         }
```

2.2.4 Función Principal

- En primera instancia, se crea la conexión a la base de datos para después crear las instancias de las entidades.

```
513  int main() {
514      // Crear una conexión a la base de datos
515      MySQLConexion conn("root", "2020", "sistema_de_estudiantes");
516
517      // Conectar a la base de datos
518      if (!conn.open()) {
519          cerr << "No se pudo conectar a la base de datos." << endl;
520          return 1;
521      }
522
523      // Crear instancias de las entidades
524      Estudiante estudiante(conn);
525      Docente docente(conn);
526      Curso curso(conn);
527      Nota nota(conn);
```

- Cabe recalcar que antes se implementó una función que muestra el menú entidad para pedirle al usuario la opción de la acción que el usuario desea realizar sobre la tabla que seleccione en el menú principal:

```
496  void mostrar_menu_entidad(const string& nombreEntidad) {
497      cout << "\n--- " << nombreEntidad << " ---\n";
498      cout << "1. Crear\n";
499      cout << "2. Leer\n";
500      cout << "3. Actualizar\n";
501      cout << "4. Eliminar\n";
502      cout << "5. Volver al menu principal\n";
503  }
```

- Se inicia el menú principal para que después se llame la función de menú entidad para determinar qué acción se hará sobre la tabla seleccionada.
- Las opciones de acción se manejan con un switch y dependiendo del número que ingrese, esa acción realizara sobre la tabla.

```
529     // Mostrar el menú principal
530     cout << "Bienvenido al sistema de gestion de estudiantes.\n";
531     int opcionEntidad = 0, opcionAccion = 0;
532 
533     do {
534         limpiar_pantalla();
535         Color(WHITE, BLACK);
536         cout << "\n==== MENU PRINCIPAL ====\n";
537         cout << "1. Estudiantes\n";
538         cout << "2. Docentes\n";
539         cout << "3. Cursos\n";
540         cout << "4. Notas\n";
541         cout << "5. Salir\n";
542         cout << "Seleccione una opcion: ";
543         cin >> opcionEntidad;
544 
545         if (opcionEntidad == 5) break;
546 
547         limpiar_pantalla();
548         mostrar_menu_entidad(
549             opcionEntidad == 1 ? "Estudiantes" :
550             opcionEntidad == 2 ? "Docentes" :
551             opcionEntidad == 3 ? "Cursos" : "Notas"
552         );
553 
554         cin >> opcionAccion;
555 
556         switch (opcionEntidad) {
557             case 1:
558                 if (opcionAccion == 1) estudiante.crear_estudiante();
559                 else if (opcionAccion == 2) estudiante.leer_estudiantes();
560                 else if (opcionAccion == 3) estudiante.actualizar_estudiante();
561                 else if (opcionAccion == 4) estudiante.eliminar_estudiante();
562                 break;
563             case 2:
564                 if (opcionAccion == 1) docente.crear_docente();
565                 else if (opcionAccion == 2) docente.leer_docentes();
566                 else if (opcionAccion == 3) docente.actualizar_docente();
567                 else if (opcionAccion == 4) docente.eliminar_docente();
568                 break;
569             case 3:
570                 if (opcionAccion == 1) curso.crear_curso();
571                 else if (opcionAccion == 2) curso.leer_cursos();
572                 else if (opcionAccion == 3) curso.actualizar_curso();
573                 else if (opcionAccion == 4) curso.eliminar_curso();
574                 break;
575             case 4:
576                 if (opcionAccion == 1) nota.crear_nota();
577                 else if (opcionAccion == 2) nota.leer_notas();
```

```
578                 else if (opcionAccion == 3) nota.actualizar_nota();
579                 else if (opcionAccion == 4) nota.eliminar_nota();
580                 break;
581             }
582         } while (true);
583
584         // Cerrar la conexión a la base de datos
585         conn.close();
586         cout << "Conexion cerrada. Saliendo del sistema...\n";
587         return 0;
588     }
589     // FIN DE CÓDIGO
590
```

2.3 Ejecución del Programa

2.3.1 Inicio

```
==== MENU PRINCIPAL ====
1. Estudiantes
2. Docentes
3. Cursos
4. Notas
5. Salir
Seleccione una opcion: 1
```

2.3.2 Tabla estudiantes _Crear:

```
--- Estudiantes ---
1. Crear
2. Leer
3. Actualizar
4. Eliminar
5. Volver al menu principal
1
ID Estudiante: 5

Nombre: Benjamin
Apellido: Recinos
Correo: Ben@miumg.edu.gt
Fecha de nacimiento (YYYY-MM-DD): 2005-03-12
Alumno creado.
```

3.3.3 Tabla estudiantes _Leer

```
--- Estudiantes ---
1. Crear
2. Leer
3. Actualizar
4. Eliminar
5. Volver al menu principal
2
ID: 1, Nombre: Marlon, Apellido: Agustin, Correo: Marlon@gmail.com, Fecha: 2001-02-20
ID: 2, Nombre: ELIAN, Apellido: RAMOS, Correo: Elianmoran2019@gmial.com, Fecha: 2006-11-28
ID: 3, Nombre: ARIEL, Apellido: GARCIA, Correo: ARIEDECORTEZ@MIUMG.EDU.MX, Fecha: 1950-02-18
ID: 4, Nombre: Wilson, Apellido: Ortiz, Correo: wilson@miumg.edu.gt, Fecha: 2006-10-15
ID: 5, Nombre: Benjamin, Apellido: Recinos, Correo: Ben@miumg.edu.gt, Fecha: 2005-03-12
ID: 6, Nombre: RIGOBERTO, Apellido: RODAS, Correo: RIGO@MIUMG.EDU.GT, Fecha: 2006-02-01
```

3.3.4 Tabla estudiantes _Actualizar

```
--- Estudiantes ---
1. Crear
2. Leer
3. Actualizar
4. Eliminar
5. Volver al menu principal
3
ID del estudiante: 1

Nuevo Nombre: Antonio
Nuevo Apellido: Ramirez
Nuevo Correo: AtRamirez@gmail.edu.mx
Nueva Fecha (YYYY-MM-DD): 2006-02-14
Estudiante actualizado.
```

#	id	NOMBRE	APELLIDO	CORREO	FECHA_NACIMIENTO
1	1	Antonio	Ramirez	AtRamirez@gmail.edu.mx	2006-02-14
2	2	ELIAN	RAMOS	Elianmoran2019@gmial.com	2006-11-28
3	3	ARIEL	GARCIA	ARIEDECORTEZ@MIUMG.EDU.MX	1950-02-18
4	4	Wilson	Ortiz	wilson@miumg.edu.gt	2006-10-15
5	5	Benjamin	Recinos	Ben@miumg.edu.gt	2005-03-12
6	6	RIGOBERTO	RODAS	RIGO@MIUMG.EDU.GT	2006-02-01

3.3.5 Tabla estudiantes _Eliminar

```
--- Estudiantes ---
1. Crear
2. Leer
3. Actualizar
4. Eliminar
5. Volver al menu principal
4
Ingrese el id de estudiante que desea eliminar:
6
Se elimino al alumno: RIGOBERTO
```