UNIVERSIDAD CÉSAR VALLEJO

# FACULTAD DE INGENIERÍA Y ARQUITECTURA

## Escuela de Ingeniería de Sistemas

**TÍTULO DEL PROYECTO:**

Implementación de un sistema para la Institución Educativa Privada "Señor de Burgos", Lima, 2023.

**INTEGRANTES:**

Castillo Vilchez, Madeleine Rosmery (orcid.org/0000-0003-2543-9370)

Conde Chacpi, Carlos Hiroshi (orcid.org/0000-0002-0354-6522)

Huajardo Yactayo, Daniel Alexis(https://orcid.org/0000-0002-4804-9327)

Ordoñez Milla, Lizeth Jackeline (orcid.org/0000-0001-5000-2176)

Otiniano Palacios, Jerson (orcid.org/0009-0009-8429-028X)

Rios Quijandria, Alessandro del Piero (orcid.org/0000-0003-2873-8974)

**CURSO:**

GESTIÓN DE DATOS E INFORMACIÓN II

**ASESOR:**

Mg. IVAN CRISPIN SANCHEZ

**2023**

UNIVERSIDAD CÉSAR VALLEJO

## ÍNDICE

# 1. ASPECTOS GENERALES

## 1.1. Definición del Problema

El principal problema que hemos podido identificar actualmente se centra especialmente en la falta de un sistema académico en las instituciones educativas. En lo cual la falta de esta carencia ha generado una serie de retos que afectan negativamente las actividades del estudiante generando desafíos y limitaciones.

A su vez se ha podido visualizar que en los centros educativos hace falta un sistema de gestión académica para resolver las distintas problemáticas.

### 1.1.1. Descripción del Problema

La falta de un sistema centralizado dificulta la organización y gestión de datos. Esto resulta en una mayor carga de trabajo manual para mantener registros de alumnos, profesores, horarios, notas y asistencia. Esta falta de eficiencia puede llevar a errores, retrasos y una utilización ineficiente del tiempo.

El manejo manual de datos aumenta el riesgo de errores en la inserción y cálculo de información, lo que puede afectar la precisión de las notas, horarios y registros académicos. Estos errores pueden tener un impacto negativo en la evaluación justa de los estudiantes y en la toma de decisiones académicas.

Sin un sistema que facilite la asignación eficiente de aulas, horarios y profesores, se pueden producir conflictos de programación y asignaciones inadecuadas. Esto puede causar interrupciones en el proceso educativo y generar frustración entre los estudiantes y el personal docente.

La falta de un sistema de gestión académica puede causar ineficiencias, errores, dificultades en la planificación, problemas de comunicación y limitaciones en el análisis de datos, lo que impacta negativamente en la calidad general de la educación y la gestión escolar.

**Problema General:**

Deficiencia en la gestión académica en las instituciones educativas, que incluye procesos manuales, falta de tecnología adecuada y demoras en los procedimientos, lo que afecta negativamente en la eficacia y la calidad en administración educativa.

**Problemas Específicos:**

- Errores en el registro de notas y evaluaciones: La falta de automatización en el cálculo de notas aumenta el riesgo de errores humanos al ingresar o calcular las calificaciones, lo que podría afectar la precisión y equidad en la evaluación académica.
- Gestión Manual y Descentralizada de Datos Académicos: la gestión hecha a mano y la falta de una plataforma centralizada dificultan el acceso y la actualización eficiente de los datos.
- Problemas de asignación de horarios y aulas: La ausencia de una plataforma que gestione horarios y aulas puede llevar a conflictos de programación, superposición de clases o asignación inadecuada de recursos, generando interrupciones en el proceso educativo.
- Dificultad en el Seguimiento del Desempeño Académico: La falta de una plataforma de seguimiento desfavorece el monitoreo del rendimiento de los estudiantes, lo que afectará su progreso.

### 1.1.2. Objetivo General

Desarrollar una aplicación que satisfaga las necesidades del colegio al abordar la gestión y supervisión en todas las etapas de las diferentes opciones de registros.

### 1.1.3. Objetivos Específicos

Implementar un Sistema de Registro Centralizado: Establecer una plataforma que permita el registro centralizado de datos de estudiantes, incluyendo información personal, historial académico y asistencia.

Crear un Sistema de Planificación de Horarios y Aulas: Desarrollar una herramienta que facilite la asignación eficiente de aulas, horarios y profesores, minimizando los conflictos y optimizando la distribución de recursos.

Automatizar la Gestión de Notas y Evaluaciones: Desarrollar un sistema que automatice el cálculo de notas y evaluaciones, asegurando la precisión y reduciendo la carga de trabajo manual para los profesores.

Facilitar la Gestión del Personal Docente: Implementar un sistema que registre la disponibilidad, asignaciones y desempeño de los profesores para una distribución equitativa de cargas laborales y una mejor coordinación.

### 1.1.4. Alcances y Limitaciones.

La relevancia fundamental de la propuesta radica en abordar la ausencia actual de una plataforma web que automatice y gestione el proceso existente, que en la actualidad se basa en gran medida en documentos digitales en Word, Excel y documentos físicos. Esta carencia también significa que no existe una herramienta para la gestión académica y el seguimiento de actividades por parte de estudiantes e instructores.

El sistema web propuesto para el seguimiento de actividades tiene como objetivo simplificar la gestión académica, automatizar las selecciones de estudiantes y proporcionar un seguimiento confiable y eficiente. Beneficia a coordinadores, instructores y estudiantes al ofrecer capacidades de control administrativo y seguimiento para procesos específicos.

El sistema consta de varios módulos, incluyendo Administración, Temas,Cursos, Asignación de Tutores/Cursos, Registro de Actividades, Ingreso de Calificaciones y Consulta de Solicitudes y Generación de Informes.

## 2. MARCO TEÓRICO

### 2.1. Antecedentes

**Antecedentes Nacionales:**

La investigación de Amasifuen (2020) se centró en el desarrollo de un "Sistema de Registro Web" en la Facultad de Educación y Humanidades de la Universidad Nacional de la Amazonía Peruana, con el objetivo principal de mejorar los procesos de admisión y la gestión financiera de los programas educativos. Utilizando un enfoque de ingeniería aplicada y un diseño experimental, el estudio logró reducir significativamente el tiempo de registro, con una disminución del 80,5%, así como una considerable reducción del 74% en las inscripciones pagadas. Además, se mejoró la eficiencia en la preparación de informes de matrícula y económicos, mientras que la confiabilidad de las consultas sobre la situación económica aumentó del 86,45% al 93,19%.

Osorio (2017) Investigación titulada “Diseño e implementación de sistemas” Inscripción web con software libre en el Centro Educativo España en la zona de Breña 2013” identificó problemas

como el proceso de registro lento y la necesidad de registrarse. Brindar un mejor servicio en términos de tiempo. Se desarrolla un software web para este propósito. Registro de PHP como lenguaje de programación y uso como motor de base de datos MySQL, enfoque orientado a objetos (OOP) y proceso como marco metodológico Desarrollo Integrado de Software (RUP).

**Antecedentes internacionales:**

Arribas(2019).En su investigación titulada "Análisis y Evaluación de la Aplicación de Sistemas de Gestión de Calidad en Instituciones Educativas"Buscó analizar el impacto de la introducción de sistemas de gestión de calidad.impacta en la organización escolar y los resultados de centros educativos. Utilizó un enfoque mixto con entrevistas y encuestas, involucrando a 20,259 participantes, incluyendo estudiantes, padres de familia y personal docente y administrativo de la Fundación SAFA.

## 2.2. Fundamento Teórico

Originalmente los métodos aplicativos cubrían necesidades de procesamientos de modo que se almacenaban y reúnen información en las mismas que eran a través de los sistemas de ficheros manuales.

Estos procedimientos posibilitan que cada programa gestione sus propios datos; no obstante, conllevan desventajas significativas como la inconsistencia de datos, la dificultad para acceder a la información, la dependencia de la estructura del archivo, así como problemas relacionados con la seguridad y la integridad de los datos.

Con el propósito de representar de manera eficiente los procesos de la realidad objetiva, surgen las bases de datos y los sistemas

de gestión de datos. Estos se desarrollan para prevenir la inconsistencia que podría surgir al utilizar los mismos datos en diversos archivos manuales de procesos independientes.

En nuestro proyecto de investigación, emplearemos una base de datos para recopilar información de manera organizada, permitiendo que un programa informático acceda eficientemente a los datos y realice las operaciones necesarias de manera rápida.

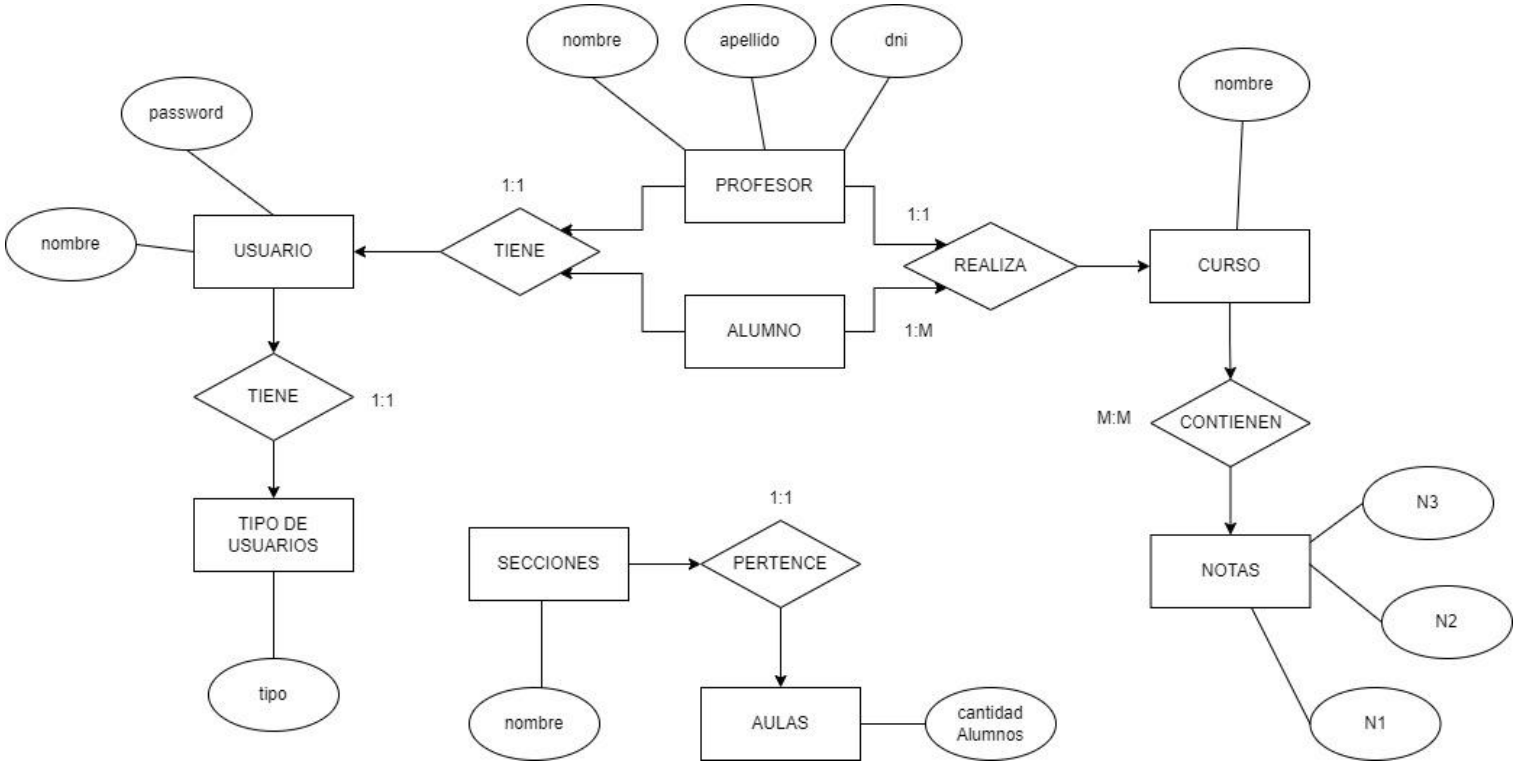
"Las bases de datos modelan hechos y objetos de una parcela de la realidad, proporcionando el respaldo necesario a una aplicación informática" (Riscos Nuñez, 2020).

"Un Sistema de Gestión de Bases de Datos (SIGMUR, 2019) se define como una colección de datos interrelacionados junto con un conjunto de programas diseñados para acceder y gestionar dichos datos".

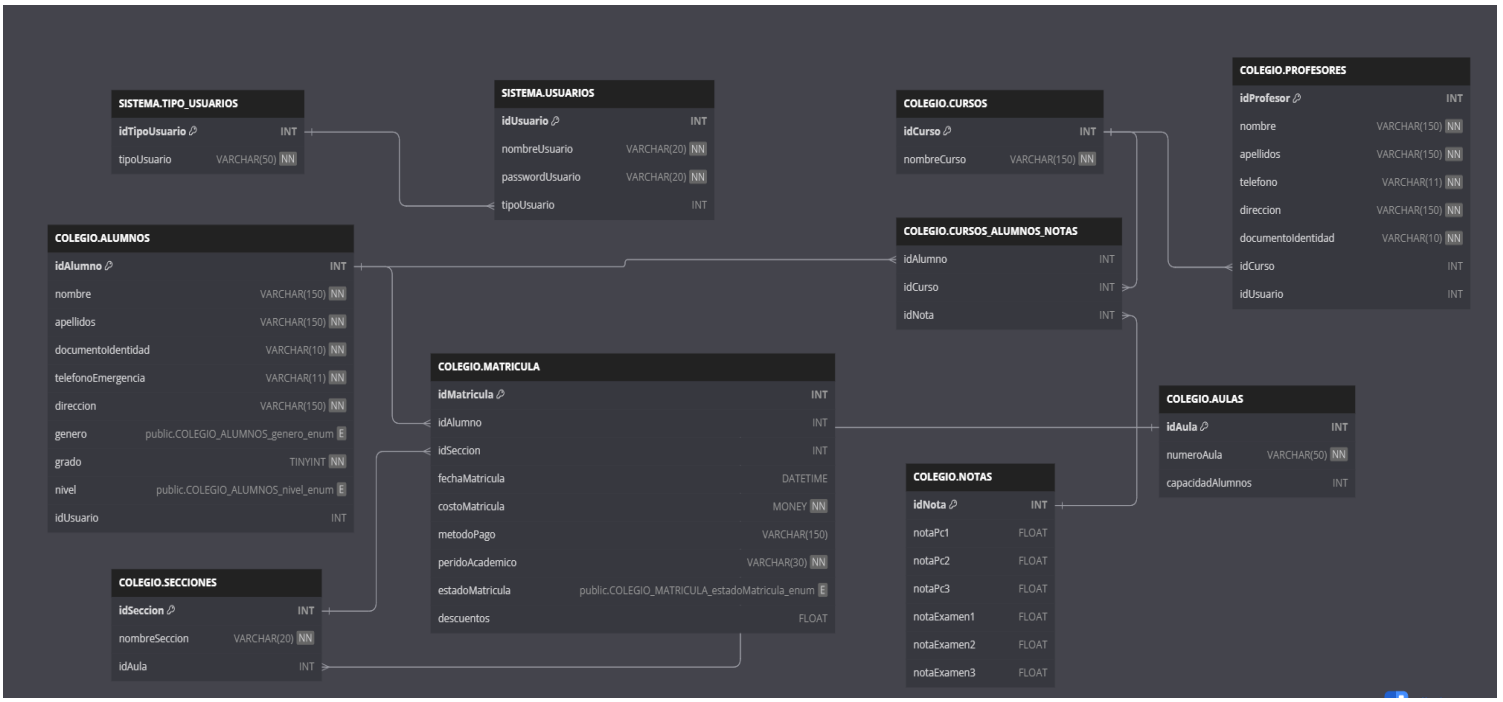
# 3. DESARROLLO DE LA SOLUCIÓN

## 3.1. Modelo Conceptual, Lógico y Físico de la base de datos

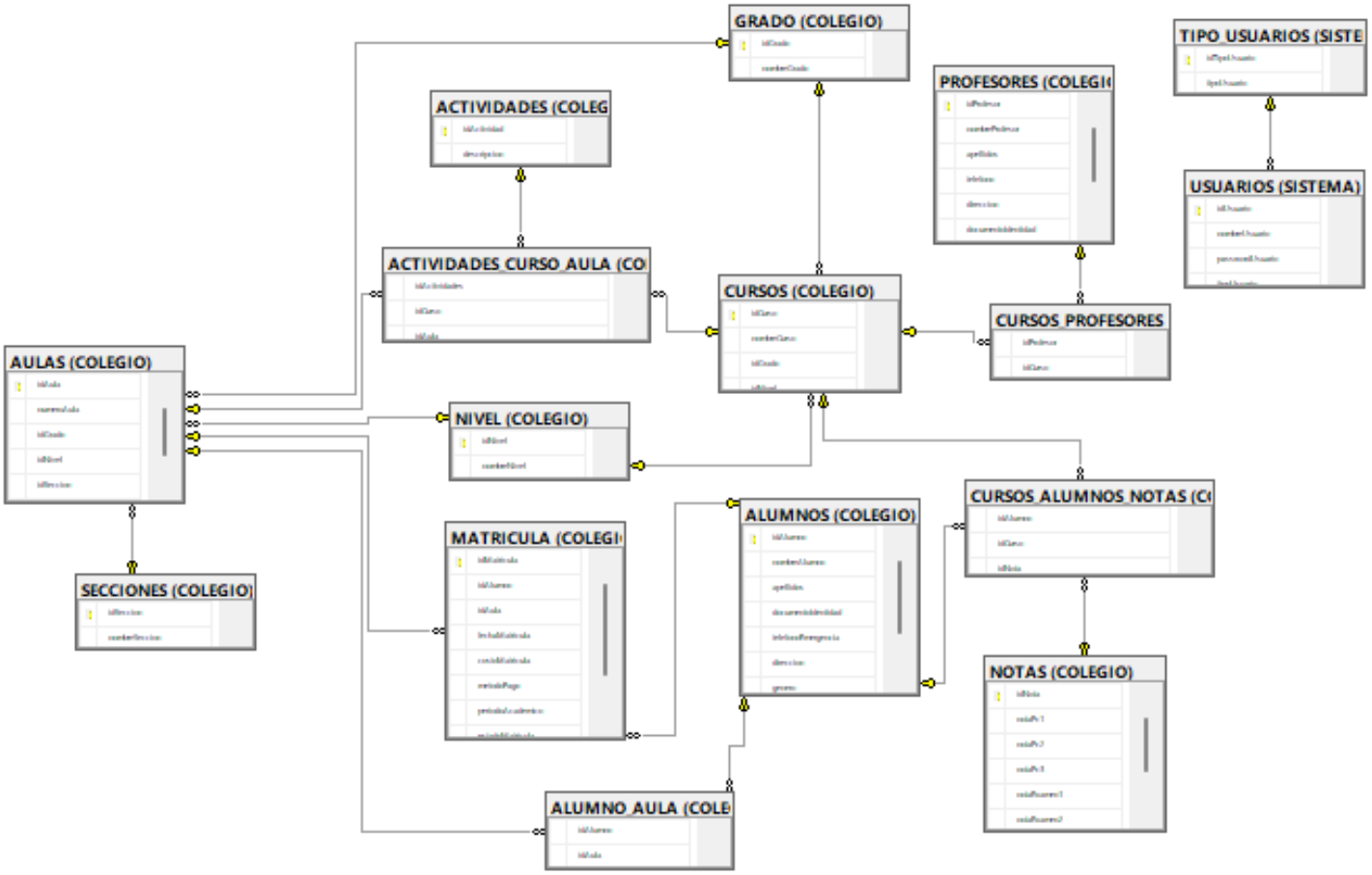
### 3.1.1. Modelo Conceptual




### 3.1.2. Modelo Lógico



### 3.1.3. Modelo Físico




## 3.2. Requerimientos funcionales y no funcionales.

### 3.2.1. Requerimientos funcionales.

- Registro de Usuarios: Los usuarios (profesores, alumnos y administrativos) deben poder registrarse en el sistema proporcionando información personal y credenciales de inicio de sesión.
- Gestión de Cursos: Los administradores deben poder agregar, editar y eliminar cursos en la base de datos, incluyendo su nombre y descripción.
- Matrícula de Alumnos: Los alumnos deben matricularse en cursos específicos, seleccionando la sección y el período académico.

- Al registrar nuevas personas en el sistema, se deben crear automáticamente su usuario y contraseña
- El sistema debe mostrar un informe de los alumnos matriculados en la pantalla del profesor.
- El sistema debe mostrar los profesores y los cursos que cada profesor está asignado.
- Registro de Notas: Los profesores deben poder ingresar notas para los alumnos en cada curso, incluyendo notas de PC (participación) y exámenes.
- Consulta de notas: Los alumnos y profesores deben poder ver sus propias notas en sus respectivos cursos.
- Gestión de Profesores: Los administradores deben poder agregar, editar y eliminar información de profesores, incluyendo datos personales y cursos asignados.
- Generación de Reportes: Los usuarios autorizados deben poder generar informes personalizados de calificaciones, listas de alumnos, etc.
- Asignación de Aulas: Los administradores deben poder asignar aulas a las secciones y cursos.
- Gestión de Secciones: Los administradores deben poder agregar, editar y eliminar secciones para asignar alumnos a cursos.
- Cambio de contraseña: Los usuarios deben poder cambiar sus contraseñas de inicio de sesión de forma segura.
- Recuperación de contraseña: Los usuarios deben poder restablecer sus contraseñas en caso de olvido.
- Gestión de Períodos Académicos: Los administradores deben poder crear y gestionar

períodos académicos, lo que incluye establecer fechas de inicio y finalización.

- Gestión de Descuentos: Los administradores deben poder aplicar descuentos a las matrículas de los alumnos según políticas específicas.

### 3.2.2. Requerimientos no funcionales.

- El sistema debe asegurar la protección de los datos del usuario y salvaguardarlos contra posibles amenazas de seguridad.
- El sistema debe contar con la capacidad de escalar para manejar un crecimiento futuro en términos de usuarios y datos.
- La eficacia del sistema es crucial; debe ser veloz y eficiente, proporcionando tiempos de respuesta rápidos incluso en condiciones de carga elevada.
- Se requiere que el sistema esté disponible las 24 horas del día, los 7 días de la semana, con un tiempo de inactividad mínimo para realizar tareas de mantenimiento.
- Interfaz de Usuario Intuitiva: La interfaz de usuario debe ser fácil de utilizar, ofreciendo una experiencia agradable y sin complicaciones al usuario.
- El sistema debe ser compatible con diversos navegadores web populares para garantizar su accesibilidad.
- La facilidad de mantenimiento y actualización del sistema es esencial, sin interrupciones significativas en el servicio.
- Debe existir una documentación completa que detalle el funcionamiento, configuración y mantenimiento del sistema.
- Se deben implementar registros y auditorías para rastrear actividades y cambios en el sistema.

## 3.3. Objetos de la Base Datos del Proyecto

### 3.3.1. Esquemas

```
--------ESQUEMAS-----
CREATE SCHEMA SISTEMA--Todas las entidades referentes al sistema con operaciones dentro de el
GO
CREATE SCHEMA COLEGIO--todas las entidades que representen personas o transacciones en el colegio
GO
```

### 3.3.2. Tablas

```
--------TABLAS-------
CREATE TABLE SISTEMA.TIPO_USUARIOS(
idTipoUsuario INT IDENTITY(1,1) PRIMARY KEY,
tipoUsuario VARCHAR(50) NOT NULL --PROFESOR,ALUMNO,administrativo
)
GO
```

```
CREATE TABLE SISTEMA.USUARIOS(
idUsuario INT IDENTITY(1,1) PRIMARY KEY,
nombreUsuario VARCHAR(20) UNIQUE NOT NULL, --Se creara apartir de la combinacion del nombre y apellido de la persona
passwordUsuario VARCHAR(20) NOT NULL,
tipoUsuario INT REFERENCES SISTEMA.TIPO_USUARIOS(idTipoUsuario)
)--cada empleado autorizado que tenga acceso al sistema
GO
```

```
CREATE TABLE COLEGIO.GRADO(
idGrado INT IDENTITY PRIMARY KEY,
nombreGrado VARCHAR(20) NOT NULL
)
GO

CREATE TABLE COLEGIO.NIVEL(
idNivel INT IDENTITY PRIMARY KEY,
nombreNivel VARCHAR(20) NOT NULL
)
GO
```

```
CREATE TABLE COLEGIO.CURSOS(
idCurso INT IDENTITY(1,1) PRIMARY KEY,
nombreCurso VARCHAR(150) NOT NULL,
idGrado INT REFERENCES COLEGIO.GRADO(idGrado),
idNivel INT REFERENCES COLEGIO.NIVEL(idNivel)
)
GO
```

```
CREATE TABLE COLEGIO.SECCIONES(
idSeccion INT IDENTITY(1,1) PRIMARY KEY,
nombreSeccion VARCHAR(20) NOT NULL,
)
GO

CREATE TABLE COLEGIO.AULAS(
idAula INT IDENTITY(1,1) PRIMARY KEY,
numeroAula VARCHAR(50) NOT NULL,
idGrado INT REFERENCES COLEGIO.GRADO(idGrado),
idNivel INT REFERENCES COLEGIO.NIVEL(idNivel),
idSeccion INT REFERENCES COLEGIO.SECCIONES(idSeccion),
capacidadAlumnos INT NOT NULL CHECK(capacidadAlumnos>20),
)
GO


CREATE TABLE COLEGIO.ALUMNOS(
idAlumno INT IDENTITY(1,1) PRIMARY KEY,
nombreAlumno VARCHAR(150) NOT NULL,
apellidos VARCHAR(150) NOT NULL,
documentoIdentidad VARCHAR(10) UNIQUE NOT NULL,
telefonoEmergencia VARCHAR(11) NOT NULL,
direccion VARCHAR(150) NOT NULL,
genero CHAR(1)CHECK(genero IN('M','F')) ,--M o F
idUsuario INT
)
GO


CREATE TABLE COLEGIO.PROFESORES(
idProfesor INT IDENTITY(1,1) PRIMARY KEY,
nombreProfesor VARCHAR(150) NOT NULL,
apellidos VARCHAR(150) NOT NULL,
telefono VARCHAR(11) NOT NULL,
direccion VARCHAR(150) NOT NULL,
documentoIdentidad VARCHAR(10) UNIQUE NOT NULL,
idUsuario INT
)--Entidades que trabajan de forma fisica en el colegio
GO
```

```
CREATE TABLE COLEGIO.NOTAS(
idNota INT IDENTITY(1,1) PRIMARY KEY,
notaPc1 FLOAT DEFAULT 0 CHECK(notaPc1 between 0 and 20),
notaPc2 FLOAT DEFAULT 0  CHECK(notaPc2 between 0 and 20),
notaPc3 FLOAT DEFAULT 0  CHECK(notaPc3 between 0 and 20),
notaExamen1 FLOAT DEFAULT 0  CHECK(notaExamen1 between 0 and 20),
notaExamen2 FLOAT DEFAULT 0  CHECK(notaExamen2  between 0 and 20),
notaExamen3 FLOAT DEFAULT 0  CHECK(notaExamen3 between 0 and 20),
)
GO
```

```
CREATE TABLE COLEGIO.CURSOS_ALUMNOS_NOTAS(
idAlumno INT REFERENCES COLEGIO.ALUMNOS(idAlumno),
idCurso INT REFERENCES COLEGIO.CURSOS(idCurso),
idNota INT REFERENCES COLEGIO.NOTAS(idNota)
)--Tabla compuesta para almacenar las notas del alumno en sus cursos
GO
```

```
CREATE TABLE COLEGIO.MATRICULA(
idMatricula INT IDENTITY(1,1) PRIMARY KEY,
idAlumno INT REFERENCES COLEGIO.ALUMNOS(idAlumno),
idAula INT REFERENCES COLEGIO.AULAS(idAula),
fechaMatricula DATETIME DEFAULT GETDATE(),
costoMatricula MONEY CHECK (costoMatricula>0) NOT NULL,
metodoPago VARCHAR(150),
periodoAcademico VARCHAR(30) NOT NULL,--año en el cual se inscribe el alumno
estadoMatricula VARCHAR(50) DEFAULT 'ACTIVO' CHECK(estadoMatricula IN('ACTIVO','INACTIVO')),
--estado que determinara si el alumno continua en el periodo de matricula
descuentos FLOAT DEFAULT 0 ,--en caso de ser necesesario
)
GO
```

```
--Tabla en la cual iran los cursos que enseña cada profesor
CREATE TABLE COLEGIO.CURSOS_PROFESORES(
idProfesor INT REFERENCES COLEGIO.PROFESORES(idProfesor),
idCurso INT REFERENCES COLEGIO.CURSOS(idCurso)
)
GO
--AULA A LA QUE PERTENECE EL ALUMNO
CREATE TABLE COLEGIO.ALUMNO_AULA(
idAlumno INT REFERENCES COLEGIO.ALUMNOS(idAlumno),
idAula INT REFERENCES COLEGIO.AULAS(idAula)
)
GO
```

```
CREATE TABLE COLEGIO.ACTIVIDADES(
idActividad INT IDENTITY PRIMARY KEY,
descripcion TEXT
)
GO

CREATE TABLE COLEGIO.ACTIVIDADES_CURSO_AULA(
idActividades INT REFERENCES COLEGIO.ACTIVIDADES(idActividad),
idCurso INT REFERENCES COLEGIO.CURSOS(idCurso),
idAula INT REFERENCES COLEGIO.AULAS(idAula)
)
GO
```

### 3.3.3. Funciones

Las funciones, tanto escalares como de tabla, son herramientas fundamentales en SQL que permiten modularizar y reutilizar lógica, mejoran la claridad del código, contribuyen a la optimización y eficiencia de las consultas, y facilitan la manipulación y el procesamiento de datos en un entorno de base de datos.

En nuestro proyecto de investigación tenemos 4 funciones:

- **Tipo table:**

  Función que retorne los datos de una matrícula, pasando el usuario del alumno como parámetro.

```
CREATE FUNCTION uf_ObtenerDatosAlumnoByIdUsuario(
@idUsuario INT
)
RETURNS TABLE
AS
RETURN(
    SELECT a.*,m.estadoMatricula,m.fechaMatricula,m.idMatricula,m.periodoAcademico,ul.*,n.nombreNivel
    FROM COLEGIO.ALUMNOS a
    INNER JOIN COLEGIO.MATRICULA m ON a.idAlumno=m.idAlumno
    INNER JOIN COLEGIO.ALUMNO_AULA au ON a.idAlumno = au.idAlumno
    INNER JOIN COLEGIO.AULAS ul ON au.idAula=ul.idAula
    INNER JOIN COLEGIO.NIVEL n ON ul.idNivel=n.idNivel
    INNER JOIN SISTEMA.USUARIOS u ON a.idUsuario=u.idUsuario
    WHERE u.idUsuario=@idUsuario
)
GO
```

Función que retorne los datos de un profesor  pasando como parámetro el id

```
--------------TIPO TABLE------
---FUNCION PARA DEVOLVER LOS DATOS DEL PROFESOR
CREATE FUNCTION uf_datosProfesorById(
@idProfesor INT
)
RETURNS TABLE
AS RETURN(
    SELECT p.*,cp.idCurso,c.nombreCurso
    FROM COLEGIO.CURSOS_PROFESORES cp
    INNER JOIN COLEGIO.PROFESORES p ON cp.idCurso=p.idProfesor
    INNER JOIN COLEGIO.CURSOS c ON cp.idCurso=c.idCurso
    WHERE cp.idProfesor=@idProfesor
)
GO
```

Funcion para logear al usuario

```
---funcion para logear al usuario
CREATE FUNCTION uf_LogearUsuario(
@nombreUsuario VARCHAR(30),
@password VARCHAR(30)
)
RETURNS TABLE
AS RETURN(
    SELECT u.idUsuario,u.nombreUsuario,u.passwordUsuario,u.tipoUsuario
    FROM SISTEMA.USUARIOS u
    WHERE @nombreUsuario=u.nombreUsuario AND u.passwordUsuario=@password
)
GO
```

Funcion que devuelve los cursos a los cuales tiene registrado un alumno

```
 --Funcion para obtener todos lo cursos de un alumno por su id
CREATE FUNCTION uf_ObtenerCursosAlumno(
@idAlumno INT
)
RETURNS TABLE
AS RETURN(
    SELECT can.*,c.nombreCurso,p.*
    FROM COLEGIO.CURSOS_ALUMNOS_NOTAS can
    INNER JOIN  COLEGIO.CURSOS c ON can.idCurso=c.idCurso
    LEFT JOIN COLEGIO.CURSOS_PROFESORES cp ON c.idCurso=cp.idCurso
    LEFT JOIN COLEGIO.PROFESORES p ON p.idProfesor=cp.idProfesor
    WHERE can.idAlumno=@idAlumno
)
GO
```

Funcion que mostrara todas las notas de un curso de un alumno por el id del curso y del alumno

```
--Funcion para devolver todas las notas de un curso de un alumno asignado por id
CREATE FUNCTION uf_NotasCursoAlumnoById(
@idAlumno INT,
@idCurso INT
)
RETURNS TABLE AS
RETURN(
    SELECT n.*,ca.idCurso,c.nombreCurso
    FROM COLEGIO.CURSOS_ALUMNOS_NOTAS ca
    INNER JOIN COLEGIO.NOTAS n ON ca.idNota=n.idNota
    INNER JOIN COLEGIO.CURSOS c ON ca.idCurso=c.idCurso
    WHERE ca.idAlumno=@idAlumno AND ca.idCurso=@idCurso
)
GO
```

Funcion que devuelve las notas de un curso en especifico de un alumno

```
--Funcion que devuelve todas las notas del alumno de todos sus cursos
CREATE FUNCTION uf_NotasCursoAlumno(
@idAlumno INT
)
RETURNS TABLE AS
RETURN(
    SELECT n.*,c.nombreCurso
    FROM COLEGIO.CURSOS_ALUMNOS_NOTAS ca
    INNER JOIN COLEGIO.NOTAS n ON ca.idNota=n.idNota
    INNER JOIN COLEGIO.CURSOS c ON c.idCurso=ca.idCurso
    WHERE ca.idAlumno=@idAlumno
)
GO
```

Funcion que devuelve las aulas que se encuentran disponibles

```
--Funcion que devuelve las aulas que estan disponibles mediante el grado , seccion y nivel seleccionado
CREATE FUNCTION uf_AulaDisponible(
@idGrado INT ,
@idNivel INT,
@idSeccion INT
)
RETURNS TABLE AS
RETURN(
    SELECT a.idAula,a.numeroAula,g.nombreGrado,n.nombreNivel,s.nombreSeccion,a.capacidadAlumnos
    FROM COLEGIO.AULAS a
    INNER JOIN COLEGIO.GRADO g ON a.idGrado=g.idGrado
    INNER JOIN COLEGIO.NIVEL n ON a.idNivel=n.idNivel
    INNER JOIN COLEGIO.SECCIONES s ON a.idSeccion= s.idSeccion
    WHERE (a.idNivel=@idNivel AND a.idGrado=@idGrado AND a.idSeccion=@idSeccion) AND a.capacidadAlumnos>0
)
GO
```

- Tipo Escalar

Función que devuelve el promedio de notas de un curso de una sección.

```
CREATE FUNCTION CalcularPromedioNotasCursoSeccion(@idCurso INT, @idSeccion INT)
RETURNS FLOAT
AS
BEGIN
    DECLARE @promedio FLOAT;

    SELECT @promedio = AVG((notaPc1 + notaPc2 + notaPc3 + notaExamen1 + notaExamen2 + notaExamen3) / 6)
    FROM COLEGIO.CURSOS_ALUMNOS_NOTAS CAN
    INNER JOIN COLEGIO.NOTAS N ON CAN.idNota = N.idNota
    WHERE CAN.idCurso = @idCurso
    AND CAN.idAlumno IN (SELECT idAlumno FROM COLEGIO.MATRICULA WHERE idSeccion = @idSeccion);

    RETURN @promedio;
END
GO
```

Función escalar que devuelve el promedio de un cursos de un alumno

```
--------------TIPO ESCALAR-----
--FUNCION DEVUELVA EL PROMEDIO DEL ALUMNO PASANDOLE EL ID DEL ALUMNO
CREATE FUNCTION ufe_CursoPromedioFinalById(
@idAlumno INT,
@idCurso INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @promedio float
    SELECT @promedio = (
    (0.4 * ((n.notaPc1 + n.notaPc2 + n.notaPc3) / 3)) +
    (((n.notaExamen1 + n.notaExamen2 + n.notaExamen3) / 3) * 0.6))
    FROM COLEGIO.NOTAS n INNER JOIN
    COLEGIO.CURSOS_ALUMNOS_NOTAS cn ON n.idNota=cn.idNota
    WHERE cn.idAlumno=@idAlumno AND n.idCurso=@idCurso
    RETURN @promedio
END
GO
```

### 3.3.4. Vistas

Las vistas son objetos lógicos que representan conjuntos de datos provenientes de una o más tablas. Estas vistas no contienen datos por sí mismas, sino que muestran una representación de los datos de las tablas subyacentes, y su definición se almacena en la base de datos.

En nuestro proyecto de investigación tenemos 4 vistas

4. Vista Alumnos Matriculados.

```
CREATE VIEW VistaAlumnosMatriculados AS
SELECT
    A.idAlumno,
    A.nombre,
    A.apellidos,
    A.documentoIdentidad,
    S.nombreSeccion,
    M.fechaMatricula,
    M.costoMatricula,
    M.estadoMatricula
FROM COLEGIO.ALUMNOS A
INNER JOIN COLEGIO.MATRICULA M ON A.idAlumno = M.idAlumno
INNER JOIN COLEGIO.SECCIONES S ON M.idSeccion = S.idSeccion
GO
```

5. Vista de los profesores y los cursos que dictan.

```
CREATE VIEW VistaProfesoresCursos AS
SELECT
    P.idProfesor,
    P.nombre,
    P.apellidos,
    C.nombreCurso AS CursoDictado
FROM COLEGIO.PROFESORES P
INNER JOIN COLEGIO.CURSOS C ON P.idCurso = C.idCurso
GO
```

6. Vista de los cursos y notas de los alumnos.

```
CREATE VIEW VistaNotasAlumnos AS
SELECT
    A.idAlumno,
    A.nombre,
    A.apellidos,
    N.notaPc1,
    N.notaPc2,
    N.notaPc3,
    N.notaExamen1,
    N.notaExamen2,
    N.notaExamen3
FROM COLEGIO.ALUMNOS A
INNER JOIN COLEGIO.CURSOS_ALUMNOS_NOTAS CAN ON A.idAlumno = CAN.idAlumno
INNER JOIN COLEGIO.NOTAS N ON CAN.idNota = N.idNota
GO
```

7. Vista de secciones y la cantidad de alumnos por cada sección.

```
CREATE VIEW VistaSeccionesAlumnos AS
SELECT
    S.idSeccion,
    S.nombreSeccion,
    COUNT(M.idAlumno) AS CantidadAlumnos
FROM COLEGIO.SECCIONES S
LEFT JOIN COLEGIO.MATRICULA M ON S.idSeccion = M.idSeccion
GROUP BY S.idSeccion, S.nombreSeccion
GO
```

### 7.1.1. Triggers

Los Triggers, también conocidos como disparadores, en SQL son procedimientos almacenados que se ejecutan automáticamente en respuesta a eventos o acciones específicas llevadas a cabo en una base de datos. Dichos eventos abarcan operaciones como inserciones (INSERT), actualizaciones (UPDATE), eliminaciones (DELETE), e incluso modificaciones en la estructura de la base de datos (DDL: Lenguaje de Definición de Datos).

Empleamos el uso de los trigger para que cree un usuario y contraseña cuando se inserte un nuevo alumno o profesor.

```
CREATE TRIGGER tg_CrearUsuarioProfesor
ON COLEGIO.PROFESORES
AFTER INSERT
AS
BEGIN
    -- Variables para almacenar datos del nuevo usuario
    DECLARE @nombreUsuario VARCHAR(30);
    DECLARE @passwordUsuario VARCHAR(30);

    -- Obtener los datos del profesor recién insertado
    SELECT @nombreUsuario = CONCAT(nombre, apellidos),
           @passwordUsuario = documentoIdentidad
    FROM inserted;

    -- Insertar un nuevo usuario en la tabla SISTEMA.USUARIOS
    INSERT INTO SISTEMA.USUARIOS (nombreUsuario, passwordUsuario, tipoUsuario)
    VALUES (@nombreUsuario, @passwordUsuario, 1);

    -- Obtener el id del usuario recién insertado
    DECLARE @idUsuario INT = SCOPE_IDENTITY();

    -- Asociar el usuario con el profesor en la tabla COLEGIO.PROFESORES
    UPDATE COLEGIO.PROFESORES
    SET idUsuario = @idUsuario
    WHERE idProfesor IN (SELECT idProfesor FROM inserted);
END
GO
```

```
--profesor
CREATE TRIGGER tg_CrearUsuarioProfesor
ON COLEGIO.PROFESORES
AFTER INSERT
AS
BEGIN
    -- Variables para almacenar datos del nuevo usuario
    DECLARE @nombreUsuario VARCHAR(30);
    DECLARE @passwordUsuario VARCHAR(30);

    -- Obtener los datos del profesor recién insertado
    SELECT @nombreUsuario = CONCAT(nombre, apellidos),
           @passwordUsuario = documentoIdentidad
    FROM inserted;

    -- Insertar un nuevo usuario en la tabla SISTEMA.USUARIOS
    INSERT INTO SISTEMA.USUARIOS (nombreUsuario, passwordUsuario, tipoUsuario)
    VALUES (@nombreUsuario, @passwordUsuario, 1);

    -- Obtener el id del usuario recién insertado
    DECLARE @idUsuario INT = SCOPE_IDENTITY();

    -- Asociar el usuario con el profesor en la tabla COLEGIO.PROFESORES
    UPDATE COLEGIO.PROFESORES
    SET idUsuario = @idUsuario
    WHERE idProfesor IN (SELECT idProfesor FROM inserted);
END
GO
```

Trigger para asignar los cursos de un alumno mediante su grado y nivel asigandos

```
--TRIGGER PARA ASIGNAR LOS CURSOS DEL ALUMNO POR GRADO Y NIVEL
CREATE TRIGGER COLEGIO.tg_AsignarCurso
ON COLEGIO.ALUMNO_AULA
AFTER INSERT
AS
BEGIN
DECLARE @idAlumno INT
    -- Insertar automáticamente un curso para cada alumno en la tabla COLEGIO.CURSOS_ALUMNOS_NOTAS
    print 'insertando datos en cursos alumno'
    INSERT INTO COLEGIO.CURSOS_ALUMNOS_NOTAS (idAlumno, idCurso)
    SELECT i.idAlumno, C.idCurso
    FROM inserted I
    INNER JOIN COLEGIO.AULAS A ON I.idAula = A.idAula
    INNER JOIN COLEGIO.CURSOS C ON A.idGrado = C.idGrado AND A.idNivel = C.idNivel
    SET @idAlumno=(SELECT i.idAlumno FROM inserted i)
```

---

```
    PRINT 'Insertando notas del alumno'

    DECLARE cursorNotas CURSOR FORWARD_ONLY FOR
        SELECT can.idNota
        FROM COLEGIO.CURSOS_ALUMNOS_NOTAS can
        WHERE can.idAlumno=@idAlumno
    OPEN cursorNotas
    DECLARE @idNota INT
        FETCH NEXT FROM cursorNotas INTO @idNota
    WHILE @@FETCH_STATUS=0
    BEGIN
        INSERT INTO COLEGIO.NOTAS DEFAULT VALUES;
        SET @idNota=SCOPE_IDENTITY()

        UPDATE COLEGIO.CURSOS_ALUMNOS_NOTAS
        SET idNota=@idNota
        WHERE CURRENT OF cursorNotas
        FETCH NEXT FROM cursorNotas INTO @idNota
    END
    CLOSE cursorNotas
    DEALLOCATE cursorNotas
END
GO
```

### 7.1.2. Indices (Clustered y no clustered)

#### 7.1.2.1. INDICES CLUSTERED

| | Nombre Tabla | Nombre Indice | Tipo de indice | Nombre Columna |
|---|---|---|---|---|
| 1 | sysdiagrams | PK__sysdiagr__C2B05B61B5EEC95F | CLUSTERED | diagram_id |
| 2 | TIPO_USUARIOS | PK__TIPO_USU__03006BFFB6E1245D | CLUSTERED | idTipoUsuario |
| 3 | USUARIOS | PK__USUARIOS__645723A6F9D4BA02 | CLUSTERED | idUsuario |
| 4 | GRADO | PK__GRADO__7AD7DF27E6655F67 | CLUSTERED | idGrado |
| 5 | NIVEL | PK__NIVEL__70D8C14F515427AF | CLUSTERED | idNivel |
| 6 | CURSOS | PK__CURSOS__8551ED054ADF19C1 | CLUSTERED | idCurso |
| 7 | SECCIONES | PK__SECCIONE__94B87A7C5B308275 | CLUSTERED | idSeccion |
| 8 | AULAS | PK__AULAS__D861CCCBC553CFAC | CLUSTERED | idAula |
| 9 | ALUMNOS | PK__ALUMNOS__43FBBAC7ED517BD9 | CLUSTERED | idAlumno |
| 10 | PROFESORES | PK__PROFESOR__E4BBA604CBBD46C0 | CLUSTERED | idProfesor |
| 11 | NOTAS | PK__NOTAS__AD5F462EC7F50277 | CLUSTERED | idNota |
| 12 | MATRICULA | PK__MATRICUL__72013C9928DA23B7 | CLUSTERED | idMatricula |
| 13 | ACTIVIDADES | PK__ACTIVIDA__327F9BEDC84E8FB0 | CLUSTERED | idActividad |

#### 7.1.2.2. ÍNDICES NO CLUSTERED

```
---NON CLUSTERED
--INDICE PARA LA TABLA ALUMNO EN LOS CAMPOS DE NOMBRE
CREATE NONCLUSTERED INDEX IX_ALUMNOS_NOMBRE
ON COLEGIO.ALUMNOS (nombre)
GO


--INDICE PARA LA TABLA PROFESOR EN LOS CAMPOS DE NOMBRE Y APELLIDO
CREATE NONCLUSTERED INDEX IX_PROFESORES_NOMBRE_APELLIDO
ON COLEGIO.PROFESORES (nombre, apellidos)
GO


--INDICE PARA LA TABLA MATRICULA EN EL CAMPO FECHA DE MATRICULA
CREATE NONCLUSTERED INDEX IX_MATRICULA_FECHA_MATRICULA
ON COLEGIO.MATRICULA (fechaMatricula)
GO


--INDICE PARA VER LAS NOTAS DE LOS ALUMNOS
CREATE NONCLUSTERED INDEX IX_NOTAS_CURSOS_ALUMNOS
ON COLEGIO.CURSOS_ALUMNOS_NOTAS(idAlumno,idCurso,idNota)
GO


---NON CLUSTERED
CREATE NONCLUSTERED INDEX IX_USUARIO_NOMBRE
ON SISTEMA.USUARIOS(nombreUsuario,tipoUsuario)
GO
```

### 7.1.3. Cursores

```
--CURSOR PARA LISTAR LOS ALUMNOS MATRICULADOS POR AÑOS
DECLARE @nombreSeccion VARCHAR(20);
DECLARE @fechaMatricula DATETIME;
DECLARE @costoMatricula MONEY;
DECLARE @estadoMatricula VARCHAR(50);
DECLARE cur_AlumnosMatriculados CURSOR FOR
SELECT S.nombreSeccion, M.fechaMatricula, M.costoMatricula, M.estadoMatricula
FROM COLEGIO.ALUMNOS A
INNER JOIN COLEGIO.MATRICULA M ON A.idAlumno = M.idAlumno
INNER JOIN COLEGIO.SECCIONES S ON M.idSeccion = S.idSeccion
WHERE YEAR(M.fechaMatricula) = 2023

OPEN AlumnosMatriculadosCursor;

-- Recorrer y mostrar los datos de los alumnos matriculados
FETCH NEXT FROM AlumnosMatriculadosCursor INTO @nombreSeccion, @fechaMatricula, @costoMatricula, @estadoMatricula;
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Alumno matriculado en Sección: ' + @nombreSeccion;
    PRINT 'Fecha de Matrícula: ' + CONVERT(VARCHAR, @fechaMatricula, 120);
    PRINT 'Costo de Matrícula: ' + CONVERT(VARCHAR, @costoMatricula, 1);
    PRINT 'Estado de Matrícula: ' + @estadoMatricula;
    PRINT '----------------------------';

    FETCH NEXT FROM AlumnosMatriculadosCursor INTO @nombreSeccion, @fechaMatricula, @costoMatricula, @estadoMatricula;
END;

CLOSE AlumnosMatriculadosCursor;
DEALLOCATE AlumnosMatriculadosCursor;
```

```
DECLARE @idAlumno INT;
DECLARE @nombreAlumno VARCHAR(150);
DECLARE @promedio FLOAT;

DECLARE PromedioNotasCursor CURSOR FOR
SELECT A.idAlumno, A.nombre
FROM COLEGIO.ALUMNOS A
INNER JOIN COLEGIO.CURSOS_ALUMNOS_NOTAS CAN ON A.idAlumno = CAN.idAlumno
INNER JOIN COLEGIO.NOTAS N ON CAN.idNota = N.idNota
WHERE CAN.idCurso = 1
OPEN PromedioNotasCursor;

-- Calcular y mostrar el promedio de notas de los alumnos
FETCH NEXT FROM PromedioNotasCursor INTO @idAlumno, @nombreAlumno;
WHILE @@FETCH_STATUS = 0
BEGIN
    SET @promedio = dbo.CalcularPromedioAlumno(@idAlumno); -- Utiliza la función que calcula el promedio de notas

    PRINT 'Alumno: ' + @nombreAlumno;
    PRINT 'Promedio de Notas: ' + CONVERT(VARCHAR, @promedio);
    PRINT '----------------------------';

    FETCH NEXT FROM PromedioNotasCursor INTO @idAlumno, @nombreAlumno;
END;
CLOSE PromedioNotasCursor;
DEALLOCATE PromedioNotasCursor;
```

```sql
---Cursor para listar los profesores y los cursos que dictan
DECLARE @idProfesor INT;
DECLARE @nombreProfesor VARCHAR(150);
DECLARE @cursoDictado VARCHAR(150);

DECLARE ProfesoresCursosCursor CURSOR FOR
SELECT P.idProfesor, CONCAT(P.nombre, ' ', P.apellidos) AS nombreProfesor, C.nombreCurso AS cursoDictado
FROM COLEGIO.PROFESORES P
INNER JOIN COLEGIO.CURSOS C ON P.idCurso = C.idCurso;

OPEN ProfesoresCursosCursor;

-- Recorrer y mostrar los datos de los profesores y cursos
FETCH NEXT FROM ProfesoresCursosCursor INTO @idProfesor, @nombreProfesor, @cursoDictado;
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Profesor: ' + @nombreProfesor;
    PRINT 'Curso Dictado: ' + @cursoDictado;
    PRINT '----------------------------';

    FETCH NEXT FROM ProfesoresCursosCursor INTO @idProfesor, @nombreProfesor, @cursoDictado;
END;

CLOSE ProfesoresCursosCursor;
DEALLOCATE ProfesoresCursosCursor;
```

### 7.1.4. Procedimientos almacenados con Transacciones

Procedimiento almacenado para matricular a un nuevo Alumno con control de transacciones

```
--PROCEDIMIENTO PARA MATRICULAR UN ALUMNO
CREATE PROCEDURE usp_MatricularAlumno(
--Datos Alumno
@NombreAlumno VARCHAR(150),
@Apellidos VARCHAR(150),
@DocumentoIdentidad VARCHAR(10),
@TelefonoEmergencia VARCHAR(11),
@Direccion VARCHAR(150),
@Genero CHAR(1),
--Datos Matricula del Alumno
@costoMatricula FLOAT,
@metodoPago VARCHAR(50),
@periodoAcademico VARCHAR(50),
@idAula INT
)
AS
BEGIN

BEGIN TRAN
    BEGIN TRY
        -- insertamos los datos del alumno
        print 'insertando datos alumno nuevo'
        INSERT INTO COLEGIO.ALUMNOS (NombreAlumno,Apellidos,DocumentoIdentidad,TelefonoEmergencia,Direccion,Genero)
        VALUES (@NombreAlumno,@Apellidos,@DocumentoIdentidad,@TelefonoEmergencia,@Direccion,@Genero)

        --Guardamos el identificador del alumno recien agregado
        DECLARE @idAlumno INT = SCOPE_IDENTITY();
        print 'insertando datos alumno en matricula'
        --insertamos los datos de la matricula
        INSERT INTO COLEGIO.MATRICULA (idAlumno,costoMatricula,metodoPago,periodoAcademico,idAula)
        VALUES (@idAlumno,@costoMatricula,@metodoPago,@periodoAcademico,@idAula)
        print 'insertando alumno_aula'
        --insertamos los datos del aula a la que va ir el alumno
        INSERT INTO COLEGIO.ALUMNO_AULA(idAlumno,idAula) VALUES(@idAlumno,@idAula)

        COMMIT--CONFIRMAMOS LA INSERCION
    END TRY
    BEGIN CATCH
    print error_message()
        ROLLBACK--DESHACEMOS EN CASO DE CUALQUIER ERROR
    END CATCH
END
GO
--PROCEDIMIENTO QUE DEVUELVA LOS DATOS DE USUARIO DE UN ALUMNO MATRICULADO POR EL DNI
```

.

Procedimiento para actualizar datos de un alumno existente con control de transaccione

```sql
----ACTUALIZAR DATOS DE UN ALUMNO
CREATE PROCEDURE usp_ActualizarAlumno(
@idAlumno INT,
@nombreAlumno VARCHAR(150),
@apellidos VARCHAR(150),
@documentoIdentidad VARCHAR(10),
@telefonoEmergencia VARCHAR(11),
@direccion VARCHAR(150)
)
AS
BEGIN
DECLARE @confirmacion INT
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE COLEGIO.ALUMNOS
        SET
            nombreAlumno = @nombreAlumno,apellidos = @apellidos,documentoIdentidad = @documentoIdentidad,
            telefonoEmergencia = @telefonoEmergencia,
            direccion = @direccion
        WHERE idAlumno = @idAlumno;
        COMMIT---Confirmamos la transaccion
        SET @confirmacion=1---se agrego correctamente
    END TRY
    BEGIN CATCH
        ROLLBACK
        SET @confirmacion=0--transaccion fallida
    END CATCH
    RETURN @confirmacion
END
GO
```

Procedimiento para actualizar datos de un profesor existente con control de transacciones

```sql
--- ACTUALIZAR DATOS DEL PROFESOR
CREATE PROCEDURE ActualizarProfesor
    @idProfesor INT,
    @nombreProfesor VARCHAR(150),
    @apellidos VARCHAR(150),
    @telefono VARCHAR(11),
    @direccion VARCHAR(150),
    @documentoIdentidad VARCHAR(10)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
        DECLARE @confirmacion INT
        UPDATE COLEGIO.PROFESORES
        SET
            nombreProfesor = @nombreProfesor,apellidos = @apellidos, telefono = @telefono,direccion = @direccion,
            documentoIdentidad = @documentoIdentidad
        WHERE idProfesor = @idProfesor;
        COMMIT
        SET @confirmacion=1---transaccion realizada
    END TRY
    BEGIN CATCH
        ROLLBACK
        SET @confirmacion=0 --transaccion fallida
    END CATCH
    RETURN @confirmacion
END
GO
```

Procedimiento para obtener la información de un usuario mediante su dni

```
GO
--PROCEDIMIENTO QUE DEVUELVA LOS DATOS DE USUARIO DE UN ALUMNO MATRICULADO POR EL DNI
CREATE PROCEDURE usp_ObtenerUsuarioByDNI(
@dni VARCHAR(8)
)
AS
BEGIN
    SELECT *
    FROM SISTEMA.USUARIOS u
    WHERE u.passwordUsuario LIKE @dni
END
GO
--PROCEDIMIENTO QUE INVOCARA A UNA FUNCION PARA MOSTRAR LAS NOTAS Y EL CURSO DE UN ALUMNO
```

Procedimiento para listar las actividades de un alumno y un curso mediante su id

```
--FUNCION PARA MOSTRAR ACTIVIDADES POR ID DE ALUMNO Y CURSO
CREATE PROCEDURE usp_ActividadesAlumnoCurso(
@idGrado INT,
@idNivel INT,
@idCurso INT
)
AS
BEGIN
    SELECT a.*,aca.idCurso,cur.nombreCurso
    FROM COLEGIO.ACTIVIDADES a
    INNER JOIN COLEGIO.ACTIVIDADES_CURSO_AULA aca ON a.idActividad=aca.idActividades
    INNER JOIN COLEGIO.CURSOS cur ON aca.idCurso=cur.idCurso
    INNER JOIN COLEGIO.AULAS au ON aca.idAula=au.idAula
    WHERE au.idGrado=@idGrado AND au.idNivel=@idNivel AND aca.idCurso=@idCurso
END
GO
```

## 7.2. Roles y privilegios de los usuarios de la Base Datos del Proyecto

CREAMOS ROLES DE ADMINISTRADOR, PROFESOR Y ALUMNO

```
----------------ROLES DE SEGURIDAD----
CREATE ROLE Administrador;
CREATE ROLE Profesor;
CREATE ROLE Alumno;
```

ASIGNAMOS LOS PERMISOS QUE TENDRÁ EL ADMINISTRADOR SOBRE LA BASE DE DATOS

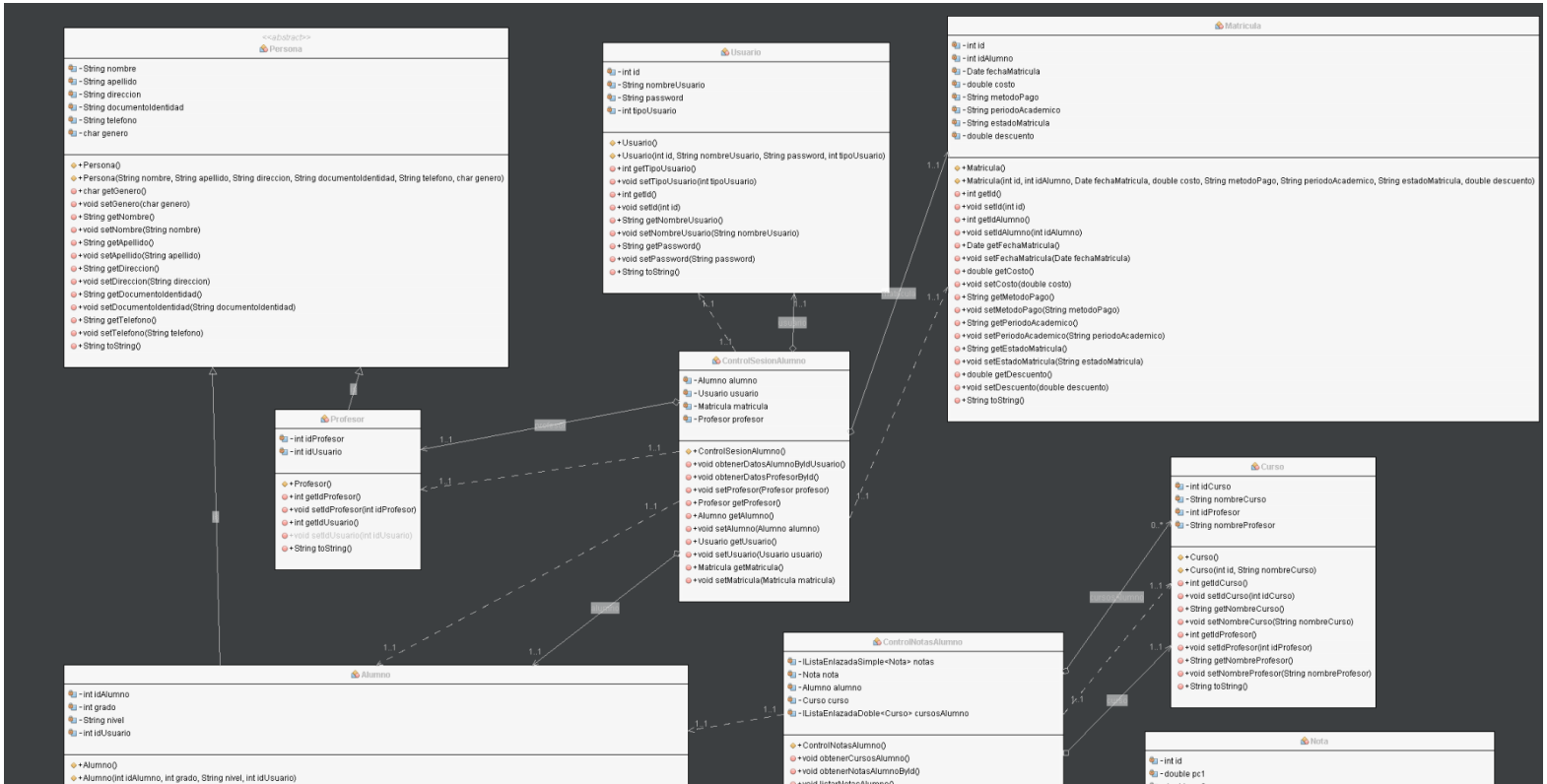
```
----PERMISOS----
---ADMIN
GRANT SELECT, INSERT, UPDATE, DELETE ON SISTEMA.TIPO_USUARIOS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON SISTEMA.USUARIOS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.AULAS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.GRADO TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.NIVEL TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.CURSOS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.SECCIONES TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.ALUMNOS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.PROFESORES TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.NOTAS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.CURSOS_ALUMNOS_NOTAS TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.MATRICULA TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.CURSOS_PROFESORES TO Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.ALUMNO_GRADO_NIVEL TO Administrador;
```

ASIGNAMOS LOS PERMISOS PARA PROFESOR Y ALUMNO

```
-- PROFESOR
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.CURSOS TO Profesor;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.CURSOS_ALUMNOS_NOTAS TO Profesor;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.NOTAS TO Profesor;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.PROFESORES TO Profesor;
GRANT SELECT, INSERT, UPDATE, DELETE ON COLEGIO.ALUMNO_GRADO_NIVEL TO Profesor;

--ALUMNO
GRANT SELECT ON COLEGIO.CURSOS TO Alumno;
GRANT SELECT ON COLEGIO.ALUMNOS TO Alumno;
GRANT SELECT ON COLEGIO.NOTAS TO Alumno;
GRANT SELECT ON COLEGIO.CURSOS_ALUMNOS_NOTAS TO Alumno;
GRANT SELECT ON COLEGIO.MATRICULA TO Alumno;
```

## 7.3. Arquitectura del proyecto (MVC o DAO) en UML



## 8. PROTOTIPO

### 8.1. Login


### 8.2. Matricula

## 8.3. Vista Alumno



## 9. CONCLUSIONES

Hemos concluido con éxito el desarrollo de un sistema de base de datos para abordar la deficiencia en la gestión académica de instituciones educativas. Esta aplicación web centralizada resuelve problemas como errores en el registro de notas y asignación de horarios, automatizando procesos clave. Beneficia a coordinadores, profesores y estudiantes al mejorar la eficiencia y transparencia en la administración educativa. La implementación exitosa de registros centralizados, planificación de horarios y automatización de notas contribuye a una gestión más efectiva. El sistema reduce la carga manual, minimiza errores y optimiza la asignación de recursos, mejorando la calidad educativa. La plataforma sienta bases sólidas para una gestión académica eficiente y mejora la experiencia del estudiante. Nuestro compromiso con la mejora continua refuerza la causa de proporcionar un entorno educativo efectivo y de calidad.