UNIVERSIDAD PRIVADA
“FRANZ TAMAYO”
Ingeniería de Sistemas

INTEGRANTES:

ALEXANDER HEYTAN CALLISAYA VALENCIA

HENRY JAVIER HUARACHI QUISPE

FREDDY MACHACA MAMANI

DOCENTE:

Lic.William Roddy Barra Paredes

El Alto – Bolivia 2022

CAPITULO I

Introducción

La base de datos de una universidad incluiría información relacionada con los estudiantes, docentes, carreras, materias, notas, roles, usuarios, horarios, etc. Esta información se almacenaría en tablas relacionadas entre sí para permitir el acceso y la búsqueda de información precisa. La base de datos se diseñaría para permitir la manipulación de datos, la recuperación de información y el seguimiento de los cambios en los datos con el fin de desarrollar lo mencionado se usará el MariaDb como gestor de base de datos.

PROBLEMA GENERAL

La gestión de información sobre estudiantes, profesores, cursos y calificaciones. La base de datos podría almacenar información como nombres, apellidos, números de identificación, carreras, asignaturas, grupos, horarios, calificaciones, etc. y permitiría a la universidad llevar un registro preciso y actualizado de todos estos datos. La base de datos podría ser utilizada por diferentes usuarios, como profesores, estudiantes, administradores, etc. para acceder a la información de manera rápida y fácil, realizar consultas, generar reportes, etc. De esta manera, la base de datos podría ayudar a la universidad a gestionar de manera eficiente y efectiva la información sobre sus estudiantes, profesores y cursos.

OBJETIVOS

Objetivos Generales

Gestionar de manera mucho más eficiente y efectiva la información sobre sus estudiantes, profesores y cursos.

Objetivos Específicos

Facilitar el acceso a la información de manera rápida y fácil, a través de consultas y búsquedas sencillas.

Facilitar el acceso a la información de manera rápida y fácil, a través de consultas y búsquedas sencillas.

Proporcionar una fuente centralizada y confiable de información sobre estudiantes, profesores, cursos y calificaciones, que pueda ser utilizada por diferentes usuarios de la universidad.

CAPITULO II

MARCO TEÓRICO

Fuente: Google

Base de datos relacional

En resumen, una base de datos es un sistema que permite almacenar y gestionar grandes cantidades de información de manera organizada y eficiente. Las bases de datos pueden ser utilizadas por diferentes usuarios para acceder a la información de manera rápida y fácil, realizar consultas, generar reportes, etc. Las bases de datos relacionales son un tipo específico de base de datos que utiliza un esquema de tablas para almacenar y gestionar la información, y que ofrece una gran flexibilidad y escalabilidad

MariaDB

Es un sistema de gestión de base de datos relacional (DBMS) de código abierto, que se ha desarrollado como una alternativa a MySQL, otro DBMS muy popular. MariaDB fue creada por los mismos desarrolladores originales de MySQL, con el objetivo de mantener la estabilidad, la calidad y la compatibilidad con éste, pero añadiendo nuevas características y mejoras.

SQL

Es un lenguaje esencial para quienes trabajan con bases de datos relacionales, ya que les permite gestionar y manipular la información de manera sencilla y eficiente. En general se dividen en 4 categorías principales:

DDL, DML, DCL, DQL.

DDL (Data Definition Language):

Estas son las declaraciones utilizadas para definir la estructura de una base de datos, como la creación, modificación y eliminación de tablas e índices.

DML (Data Manipulation Language):

Estas son las instrucciones utilizadas para manipular datos en una base de datos, como insertar, actualizar y eliminar registros.

DCL (Data Control Language):

Estas son las declaraciones utilizadas para controlar el acceso a los datos en una base de datos, como conceder y revocar permisos a usuarios.

SQL

Es un acrónimo que se refiere a Data Query Language, es decir, un lenguaje de consulta de datos. DQL es una categoría de SQL que se utiliza para realizar consultas a una base de datos y recuperar información de ella. En general, DQL se utiliza para realizar consultas a una base de datos y recuperar datos de ella de manera eficiente y precisa.

SELECT: esta declaración se utiliza para recuperar datos de una base de datos.

WHERE: esta cláusula se utiliza para especificar condiciones que deben cumplirse para que un registro sea incluido en el resultado de una consulta.

GROUP BY: esta cláusula se utiliza para agrupar resultados de una consulta por uno o más campos.

HAVING: esta cláusula se utiliza para especificar condiciones que deben cumplirse para que un grupo sea incluido en el resultado de una consulta agrupada.

ORDER BY: esta cláusula se utiliza para ordenar los resultados de una consulta por uno o más campos

FUNCIONES DEL MARIADB

Una función es una rutina creada para tomar unos parámetros, procesarlos y retornar en una salida.

¿Cómo crear, modificar y cómo eliminar una función?

Para crear una función debemos de usar la sentencia **CREATE FUNCTION** dándole un nombre.

Para modificar una función usamos el comando **ALTER FUNCTION** más el nombre de la función.

Para eliminar una función usamos el comando **DROP FUNCTION** más el nombre de la función.

Create Function Ejemplo

```
CREATE FUNCTION get_total_sales(product_id INT) // Nombre de la función y parámetros
RETURNS DECIMAL(10,2) (tipo de dato)
BEGIN
  DECLARE total_sales DECIMAL(10,2);
  SELECT SUM(quantity * price) INTO total_sales // Atributos de la rutina
  FROM sales
  WHERE product_id = product_id;  //Bloque de instrucciones
  RETURN total_sales;
END;
```

Función Alter

```
ALTER FUNCTION get_total_sales(product_id INT)
RETURNS DECIMAL(10,2)
BEGIN
  RETURN (SELECT SUM(quantity * price) FROM sales WHERE product_id = product_id);
END;
```

Función CONCAT

La función se usa para concatenar dos o más cadenas juntas. Aquí hay un ejemplo de cómo usarlo:

```
SELECT CONCAT('Hello', ' ', 'World') AS greeting;
```

Esta declaración devolvería la cadena 'Hello World' como `saludar greeting columna.

Función SUBSTRING

Se utiliza para extraer una subcadena o una parte de la cadena contra la cadena de entrada. Como sugiere el nombre, la función Substring opera en una cadena de entrada y devuelve una subcadena más pequeña contra las opciones especificadas.

```
SELECT SUBSTRING('Hello World', 6, 5) AS greeting;
```

La función`**SUBSTRING**() toma tres argumento SUBSTR() como una **SUBSTRING**()

Función STRCMP

La función STRCMP compara dos cadenas de caracteres y devuelve un número entero que indica si la primera cadena es menor (-1), igual (0) o mayor (1) que la segunda. En este caso, la función devolverá -1 porque la cadena 'hello' es alfabéticamente menor que la cadena 'world'. Si quieres saber si dos cadenas son iguales, puedes usar la función STRCMP junto con la sentencia IF para ejecutar una acción si las cadenas son iguales:

```
IF STRCMP('hello', 'hello') = 0 THEN

SELECT 'The strings are equal.';

ELSe

SELECT 'The strings are not equal.';
END IF;
```

Función CHAR_LENGTH y LOCATE

La función CHAR_LENGTH devuelve la longitud de una cadena de caracteres, mientras que la función LOCATE devuelve la posición de una subcadena dentro de una cadena de caracteres.

```
SELECT CHAR_LENGTH('hello world') AS length,
LOCATE('world', 'hello world') AS position;
```

¿Cuál es la diferencia entre las funciones de agresión y funciones creados por el DBA?

Las funciones de agregación son funciones predefinidas en SQL que se utilizan para realizar cálculos sobre un conjunto de valores y devolver un único resultado, mientras que las funciones creadas por el DBA son funciones personalizadas que se crean y definen por el administrador de base de datos o por desarrolladores de aplicaciones para realizar cálculos o procesos específicos de la aplicación. Las funciones creadas por el DBA ofrecen un mayor grado de flexibilidad y personalización que las funciones de agregación predefinidas.

Parámetros de entrada y de salida

IN: Es un parámetro que se utiliza por defecto, y requiere que la aplicación o código que invoca al procedimiento pase un argumento para él. Cuando el procedimiento se ejecuta, trabaja con una copia del valor del argumento, dejando el parámetro con su valor original al finalizar la ejecución del procedimiento.

OUT: Es un parámetro cuyo valor puede ser modificado en el procedimiento, y además su valor modificado se envía de vuelta al código o programa que invoca al procedimiento.

INOUT: Es una combinación de los conceptos anteriores. La aplicación o código que invoca al

procedimiento puede pasarle un valor, y el procedimiento devuelve el valor modificado al finalizar la ejecución.

Trigger en MariaDB

Un trigger es una acción que se ejecuta automáticamente en una base de datos cuando se produce un evento específico, como insertar, actualizar o eliminar datos en una tabla. Los triggers se utilizan para implementar reglas de negocio y validaciones en una base de datos, y pueden ser una herramienta útil para garantizar la integridad y consistencia de los datos.

```
CREATE TRIGGER validate_email BEFORE INSERT ON users
FOR EACH ROW
BEGIN
  IF NOT (NEW.email LIKE '%@%.%') THEN
    SIGNAL SQLSTATE 'error email'
    SET MESSAGE_TEXT = 'Invalid email address.';
  END IF;
END;
```

En este ejemplo, el trigger se ejecuta antes de que se inserte una nueva fila en la tabla users, y comprueba si el campo email de la fila tiene un formato válido de dirección de correo electrónico. Si el formato no es válido, el trigger lanza una excepción utilizando la sentencia SIGNAL para indicar que ha ocurrido un error y evitar que se inserte la fila en la tabla. De esta manera, el trigger se asegura de que solo se inserten filas con direcciones de correo electrónico válidas en la tabla de usuarios.

Vistas en MariaDB

Las vistas en MariaDB son objetos de base de datos que representan una consulta SQL guardada de manera permanente. Las vistas se utilizan para simplificar la consulta de datos en una base de datos, ya que permiten almacenar y reutilizar consultas complejas de manera más sencilla.

```
CREATE VIEW product_sales AS

SELECT product_id, SUM(quantity * price) AS total_sales

FROM sales
GROUP BY product_id;
```

En este ejemplo, la vista se llama product_sales y contiene el resultado de una consulta SELECT que suma el total de ventas por cada producto en la tabla sales. Una vez creada, la vista puede ser utilizada en otras consultas SQL como si fuera una tabla, lo que permite obtener información detallada sobre las ventas de cada producto de forma sencilla y rápida.

CAPÍTULO III

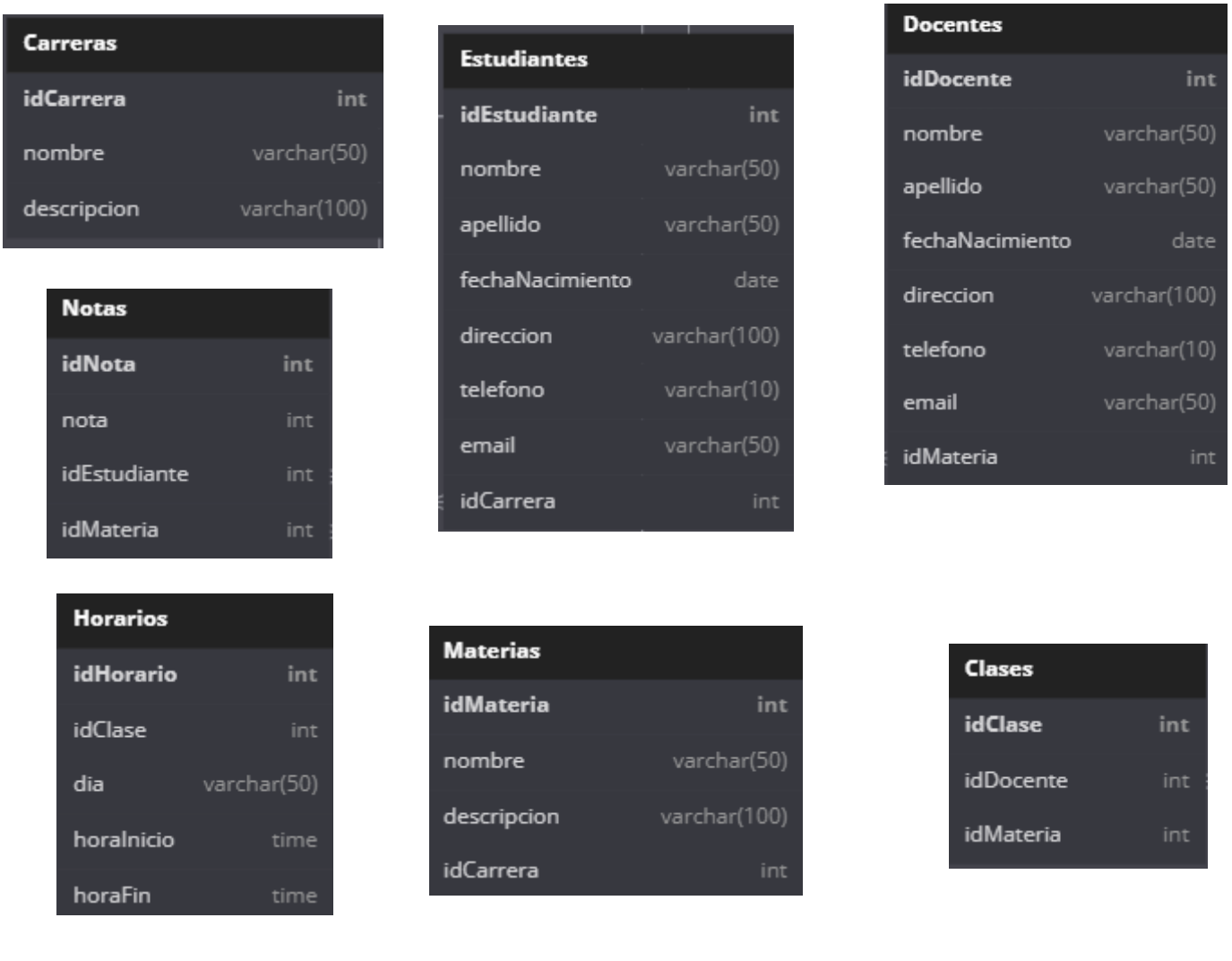
Marco Aplicativo

Análisis y diseño de la Base de Datos

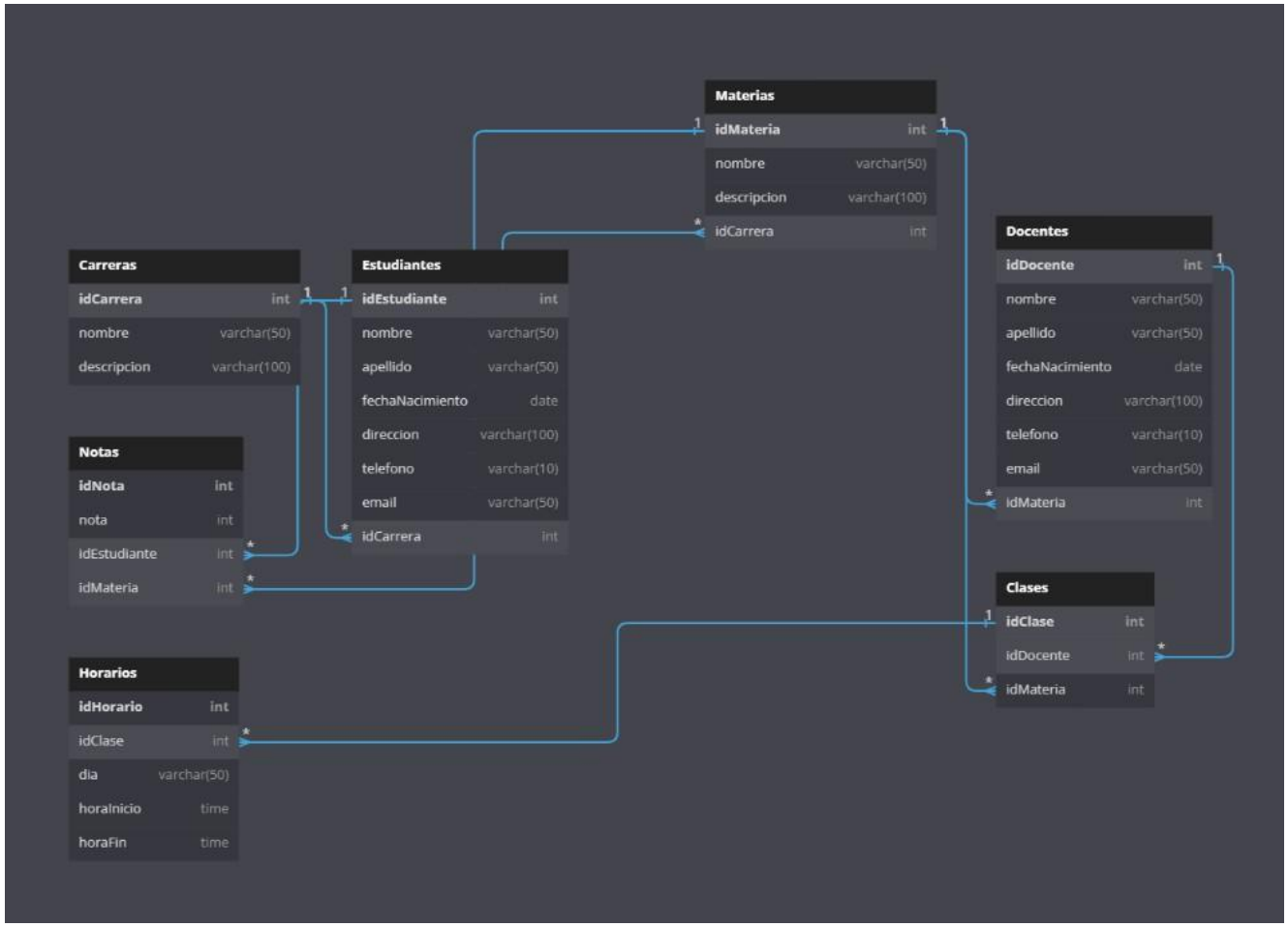
Contexto de la Base de Datos:

Debido a que la base de datos se trata de la información respecto a una universidad se identifica como nombre adecuado para la base de datos ser “DB_Universidad”.

Entidades/tablas de sistema



Diseño de la base de datos.



Código SQL

```
create database DB_Universidad;
use DB_Universidad;


-- Tabla de Carreras create table Carreras(
idCarrera int primary key auto_increment, nombre varchar(50) not null,
descripcion varchar(100) not null
);


-- Tabla de Estudiantes create table Estudiantes(
idEstudiante int primary key auto_increment, nombre varchar(50) not null,
apellido varchar(50) not null, fechaNacimiento date not null, direccion varchar(100) not
null, telefono varchar(10) not null, email varchar(50) not null, idCarrera int not null,
foreign key (idCarrera) references Carreras(idCarrera)
);


-- Tabla de Materias create table Materias(
idMateria int primary key auto_increment, nombre varchar(50) not null,
descripcion varchar(100) not null, idCarrera int not null,
foreign key (idCarrera) references Carreras(idCarrera)
);


-- Tabla de docentes create table Docentes(
idDocente int primary key auto_increment, nombre varchar(50) not null,
apellido varchar(50) not null, fechaNacimiento date not null, direccion varchar(100) not
null, telefono varchar(10) not null, email varchar(50) not null, idMateria int not null,
foreign key (idMateria) references Materias(idMateria)
);


-- Tabla de notas create table Notas(
idNota int primary key auto_increment, nota int not null,
idEstudiante int not null, idMateria int not null,
foreign key (idEstudiante) references Estudiantes(idEstudiante), foreign key (idMateria)
references Materias(idMateria)
```

```
);


-- tabla de clases create table Clases(
idClase int primary key auto_increment, idDocente int not null,
idMateria int not null,
foreign key (idDocente) references Docentes(idDocente), foreign key (idMateria) references
Materias(idMateria)
);
-- tabla de horarios create table Horarios(
idHorario int primary key auto_increment, idClase int not null,
dia varchar(50) not null, horaInicio time not null, horaFin time not null,
foreign key (idClase) references Clases(idClase)
);

-- inserción de datos

-- insertar datos en la tabla carreras insert into Carreras(nombre, descripcion)
values('Ingenieria en Sistemas', 'Ingenieria en Sistemas'), ('Ingenieria en Mecatronica',
'Ingenieria en Mecatronica'), ('Ingenieria en Mecanica', 'Ingenieria en Mecanica'),
('Ingenieria en Electrica', 'Ingenieria en Electrica'), ('Ingenieria en Electronica',
'Ingenieria en Electronica'), ('Ingenieria en Industrial', 'Ingenieria en Industrial'),
('Ingenieria en Mecanica Industrial', 'Ingenieria en Mecanica
Industrial'),
('Derecho',' Derecho'),
('Medicina', 'Medicina'), ('Psicologia', 'Psicologia');

-- insertar datos en la tabla estudiantes
insert into Estudiantes(nombre, apellido, fechaNacimiento, direccion, telefono, email,
idCarrera)
values('Juan', 'Perez', '1990-01-01', 'Calle 1', '1234567890',
'juan@gmial.com', 1),
('Maria', 'Gomez', '1990-01-01', 'Calle 2', '1234567890',
'maria@gmail.com', 2),
('Pedro', 'Gonzales', '1990-01-01', 'Calle 3', '1234567890',
'pedro@gmail.com', 3),
('Jose', 'Lopez', '1990-01-01', 'Calle 4', '1234567890',
'jose@gmail.com', 4),
('Luis', 'Martinez', '1990-01-01', 'Calle 5', '1234567890',
'luis@gmail.com', 5),
('Ana', 'Rodriguez', '1990-01-01', 'Calle 6', '1234567890',
'ana@gmail.com', 6),
```

```
('Lorena', 'Gutierrez', '1990-01-01', 'Calle 7', '1234567890',
'lorena@gmail.com', 7),
('Sofia', 'Hernandez', '1990-01-01', 'Calle 8', '1234567890',
'sofia@gmail.com', 8),
('Miguel', 'Diaz', '1990-01-01', 'Calle 9', '1234567890',
'miguel@gmail.com', 9),
('Carlos', 'Perez', '1990-01-01', 'Calle 10', '1234567890',
'carlos@gmail.com', 10);

-- insertar datos en la tabla materias

insert into Materias(nombre, descripcion, idCarrera) values
('Matematicas', 'Matematicas',1),
('Fisica', 'Fisica', 2),
('Quimica', 'Quimica', 3),
('Biologia', 'Biologia', 4),
('Programacion', 'Programacion', 5), ('Base de datos', 'Base de datos', 6),
('Redes', 'Redes', 7),
('Ingles', 'Ingles', 8),
('Frances', 'Frances', 9),
('Aleman', 'Aleman', 10);

-- insertar datos en la tabla docentes
insert into Docentes(nombre, apellido, fechaNacimiento, direccion, telefono, email,
idMateria)
values('Carlos', 'Condori', '1990-01-01', 'Calle 1', '1234567890','carlos@gmail.com', 1),
('Alejandro', 'Plata', '1990-01-01', 'Calle 2', '1234567890','alejandro@gmail.com', 2),
('Abel', 'Quispe', '1990-01-01', 'Calle 3', '1234567890','abel@gmail.com', 3),
('Alex', 'Gonzales', '1990-01-01', 'Calle 4', '1234567890','alex@gmail.com', 4),
('Luis', 'Rojas', '1990-01-01', 'Calle 5', '1234567890','luis@gmail.com', 5),
('Bocaro', 'Rodriguez', '1990-01-01', 'Calle 6', '1234567890','bocaro@gmail.com', 6),
('Raul', 'Gutierrez', '1990-01-01', 'Calle 7', '1234567890','raul@gmail.com', 7),
('Misael', 'Paredes', '1990-01-01', 'Calle 8', '1234567890','misael@gmail.com', 8),
('Minerva', 'Torrez', '1990-01-01', 'Calle 9', '1234567890','minerva@gmail.com', 9),
('Estefani', 'Maldonado', '1990-01-01', 'Calle 10', '1234567890','estefani@gmail.com', 10);

-- insertar datos en la tabla notas
insert into Notas(idEstudiante, idMateria, nota)
values (1,1, 15),
        (2,2, 16),
        (3,3, 17),
        (4,4, 18),
```

```
        (5,5, 19),
        (6,6, 20),
        (7,7, 21),
        (8,8, 22),
        (9,9, 23),
        (10, 10, 24)
-- insertar datos en la tabla clases
insert into Clases(idDocente, idMateria) values(1, 1); insert into Clases(idDocente,
idMateria) values(2, 2); insert into Clases(idDocente, idMateria) values(3, 3); insert into
Clases(idDocente, idMateria) values(4, 4); insert into Clases(idDocente, idMateria)
values(5, 5); insert into Clases(idDocente, idMateria) values(6, 6); insert into
Clases(idDocente, idMateria) values(7, 7); insert into Clases(idDocente, idMateria)
values(8, 8); insert into Clases(idDocente, idMateria) values(9, 9); insert into
Clases(idDocente, idMateria) values(10, 10);



-- insertar datos en la tabla horarios
insert into Horarios(idClase, dia, horaInicio, horaFin) values(1, 'Lunes', '08:00:00',
'10:00:00'),
(2, 'Martes', '08:00:00', '10:00:00'),
(3, 'Miercoles', '08:00:00', '10:00:00'),
(4, 'Jueves', '08:00:00', '10:00:00'),
(5, 'Viernes', '08:00:00', '10:00:00'),
(6, 'Sabado', '08:00:00', '10:00:00'),
(7, 'Domingo', '08:00:00', '10:00:00'),
(8, 'Lunes', '10:00:00', '12:00:00'),
(9, 'Martes', '10:00:00', '12:00:00'),
(10, 'Miercoles', '10:00:00', '12:00:00');



-- insertar datos en la tabla estudiantes todos en la misma carrera y materia para probar el
procedimiento almacenado
insert into Estudiantes(nombre, apellido, fechaNacimiento, direccion, telefono, email,
idCarrera)
values ('Carlos', 'Condori', '1990-01-01', 'Calle 1', '1234567890', '', 6),
        ('Alejandro', 'Plata', '1990-01-01', 'Calle 2', '1234567890', '', 6),
        ('Abel', 'Quispe', '1990-01-01', 'Calle 3', '1234567890', '', 6),
        ('Alex', 'Gonzales', '1990-01-01', 'Calle 4', '1234567890', '', 6),
        ('Luis', 'Rojas', '1990-01-01', 'Calle 5', '1234567890', '', 6),
        ('Bocaro', 'Rodriguez', '1990-01-01', 'Calle 6', '1234567890', '',6),
        ('Raul', 'Gutierrez', '1990-01-01', 'Calle 7', '1234567890', '', 6),
```

```
('Misael', 'Paredes', '1990-01-01', 'Calle 8', '1234567890', '', 6),
('Minerva', 'Torrez', '1990-01-01', 'Calle 9', '1234567890', '', 6),
('Estefani', 'Maldonado', '1990-01-01', 'Calle 10', '1234567890', '',6)


-- todos los estudiantes de la carrera de base de datos select * from Estudiantes
inner join Carreras on Estudiantes.idCarrera = Carreras.idCarrera inner join Materias M
on Carreras.idCarrera = M.idCarrera
where idMateria = 6;



# 1. Consultas SQL que maneja JOINS = 5 Consultas # 2. Funciones UDF = 3
FUNCIONES
# 3. Vistas = 5 Vistas
# a. Reutiliza 3 consultas # b. Crear 2 nuevas
# 4. Triggers = 3 TRIGGERS
# a. 1 trigger de validación # b. 2 triggers de autoria


# 1. Consultas SQL que maneja JOINS = 5 Consultas
# 1.1. Cuales son los estudiantes que tienen notas mayores a 18 select e.nombre, e.apellido,
n.nota
from Estudiantes e
inner join Notas n on e.idEstudiante = n.idEstudiante where n.nota > 5;

# 1.2. Cuales son los estudiantes que estan en la carrera de Ingenieria de Sistemas
select e.nombre, e.apellido, c.nombre from Estudiantes e
inner join Carreras c on e.idCarrera = c.idCarrera where c.nombre = 'Ingenieria en
Sistemas';


# 1.3. que docentes imparten clases los lunes y miercoles select d.nombre, d.apellido, h.dia
from Docentes d
inner join Clases c on d.idDocente = c.idDocente inner join Horarios h on c.idClase =
h.idClase where h.dia = 'Lunes' or h.dia = 'Miercoles';

# 1.4. que docentes tienen a cargo la materia de Base de datos select d.nombre, d.apellido,
m.nombre
from Docentes d
inner join Clases c on d.idDocente = c.idDocente inner join Materias m on c.idMateria =
m.idMateria where m.nombre = 'Base de datos';
```

```
# 1.5. que estudiantes tienen notas mayores a 5 en la materia de Base de datos
select e.nombre, e.apellido, m.nombre, n.nota from Estudiantes e
inner join Notas n on e.idEstudiante = n.idEstudiante inner join Materias m on n.idMateria
= m.idMateria where m.nombre = 'Base de datos' and n.nota > 5;

# 2. Funciones UDF = 3 FUNCIONES
# 2.1. Funcion que retorne el promedio de notas de un estudiantes mediante su Id
create function promedioEstudiante(idEstudiante int) returns float
begin
declare promedio float;
select avg(n.nota) into promedio from Notas n
where n.idEstudiante = idEstudiante; return promedio;
end;

select promedioEstudiante(1);

# 2.2. Funcion para obtener la informacion de un estudiante matriculado en una materia
# la funcion debe recibir como parametro el nombre del estudiante y el nombre de la
materia
create function informacionEstudiante(nombreEstudiante varchar(50), nombreMateria
varchar(50))
returns varchar(100) begin
declare informacion varchar(100);
select concat(e.nombre, ' ', e.apellido, ' ', m.nombre) into informacion
from Estudiantes e
inner join Notas n on e.idEstudiante = n.idEstudiante inner join Materias m on n.idMateria
= m.idMateria
where e.nombre = nombreEstudiante and m.nombre = nombreMateria; return
informacion;
end;

select informacionEstudiante('Ana', 'Base de datos');

# 2.3. Funcion para eligir el mejor estudiante de una materia
# la funcion debe recibir como parametro el nombre de la materia create function
mejorEstudiante(nombreMateria varchar(50))

returns varchar(100) begin
declare mejor varchar(100);
select concat(e.nombre, ' ', e.apellido, ' ', m.nombre) into mejor from Estudiantes e
inner join Notas n on e.idEstudiante = n.idEstudiante inner join Materias m on n.idMateria
```

```
= m.idMateria
where m.nombre = nombreMateria and n.nota = (select max(nota) from Notas);
return mejor; end;

select mejorEstudiante('programacion');

VISTAS
# 3. Vistas = 5 Vistas
# a. Reutiliza 3 consultas # b. Crear 2 nuevas

# 3.1. Vistas de estudiantes que tienen notas mayores a 5 en la materia de Base de datos
create view estudiantesNotasBaseDatos as select e.nombre as 'Nombre',
e.apellido as 'Apellido', m.nombre as 'Materia', n.nota as 'Nota'
from Estudiantes e
inner join Notas n on e.idEstudiante = n.idEstudiante inner join Materias m on n.idMateria
= m.idMateria where m.nombre = 'Base de datos' and n.nota > 5;

select * from estudiantesNotasBaseDatos;

# 3.2. Vistas de estudiantes que estan en la carrera de Ingenieria de Sistemas
create view estudiantesCarreraIngenieriaSistemas as select e.nombre as 'Nombre',
e.apellido as 'Apellido', c.nombre as 'Carrera'
from Estudiantes e
inner join Carreras c on e.idCarrera = c.idCarrera where c.nombre = 'Ingenieria en
Sistemas';

select * from estudiantesCarreraIngenieriaSistemas;

# 3.3. Vistas de que docentes imparten clases los lunes y miercoles create view
docentesImpartenClasesLunesMiercoles as
select d.nombre as 'Nombre', d.apellido as 'Apellido', h.dia as 'Dia'
from Docentes d
inner join Clases c on d.idDocente = c.idDocente inner join Horarios h on c.idClase =
h.idClase where h.dia = 'Lunes' or h.dia = 'Miercoles';

select * from docentesImpartenClasesLunesMiercoles;

# 3.4. Vistas de que docentes tienen a cargo la materia de programacion create view
docentesMateriaProgramacion as
select d.nombre as 'Nombre', d.apellido as 'Apellido', m.nombre as 'Materia'

from Docentes d
```

```
inner join Clases c on d.idDocente = c.idDocente inner join Materias m on c.idMateria =
m.idMateria where m.nombre = 'Programacion';

select * from docentesMateriaProgramacion;

# 3.5. Vistas de estudiantes que tiene clases con el docente william barra create view
estudiantesClasesDocenteWilliamBarra as
select e.nombre as 'Nombre', e.apellido as 'Apellido', d.nombre as 'Docente'
from Estudiantes e
inner join Clases c on e.idEstudiante = c.idMateria inner join Docentes d on c.idDocente =
d.idDocente where d.nombre = 'William' and d.apellido = 'Barra';

select * from estudiantesClasesDocenteWilliamBarra;
TRIGGERS
# 4. Triggers = 3 Triggers
# a. 1 trigger de validacion # b. 2 triggers de auditoria

# 4.1. Trigger de validacion para que se elimine un estudiante solo si no tiene notas
create trigger eliminarEstudiante before delete on Estudiantes
for each row begin
if (select count(*) from Notas where idEstudiante = old.idEstudiante) >
0 then

notas';
signal sqlstate '45000'
set message_text = 'No se puede eliminar el estudiante porque tiene

end if; end;

delete from Estudiantes where idEstudiante = 1;


# 4.2. Trigger de auditoria para que se registre cuando se haga un cambio en las notas

create table AuditoriaNotas ( idEstudiante int, idMateria int, notaAnterior int, notaActual
int
);

create trigger auditoriaNotas after update on Notas
for each row begin
insert into AuditoriaNotas values (old.idEstudiante, old.idMateria, old.nota, new.nota);
end;
```

```
update Notas set nota = 5 where idEstudiante = 1 and idMateria = 1;

select * from AuditoriaNotas;
```

4.3. Trigger de auditoria para que se registre cuando se haga un cambio en las materias

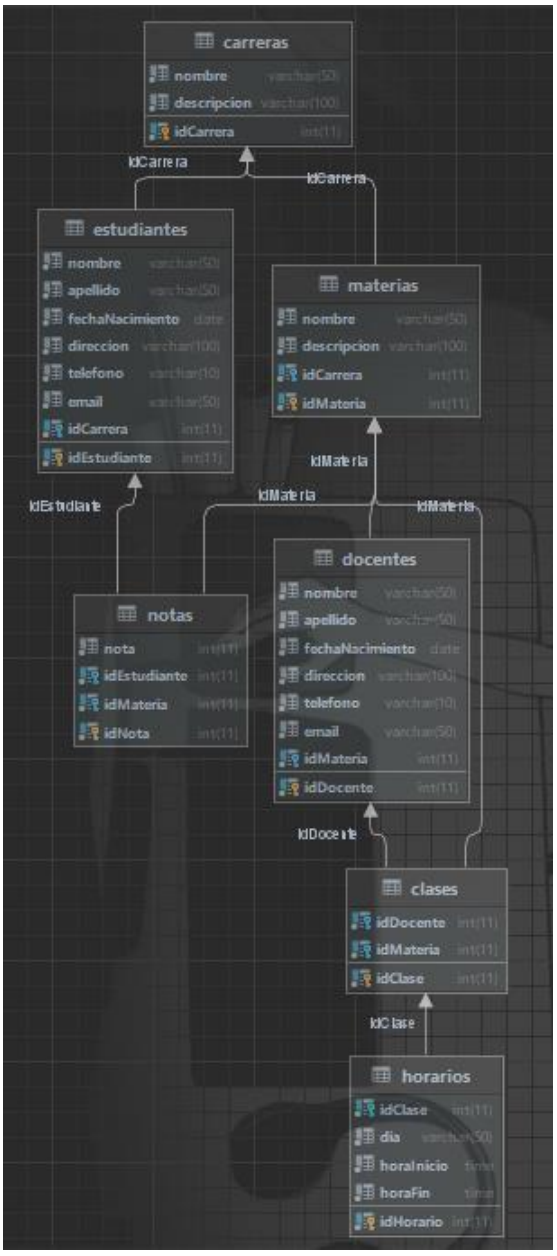
```
create or replace table AuditoriaMaterias (
   idMateria int,
   materiaAnterior varchar(50),
   materiaActual varchar(50)
);

create or replace trigger auditoriaMaterias
after update on Materias
for each row
begin
   insert into AuditoriaMaterias values (old.idMateria, old.nombre, new.nombre);
end;

update Materias set nombre = 'Base de datos' where idMateria = 2;

select * from AuditoriaMaterias;
```

DISEÑO INTELLIJ



Usabilidad

Video del sistema: https://youtu.be/sXkik8vP3Dc

Conclusiones

- Programar en una base de datos para una universidad proporciona una gran cantidad de beneficios, como una mejor organización de los datos, una mayor seguridad de los datos, una mejor eficiencia en el procesamiento de los datos y una mayor capacidad de recuperar los datos si se producen problemas.

- La programación en una base de datos para una universidad también puede proporcionar una mejor comunicación entre los departamentos y mejorar la gestión de los recursos humanos.

- La programación en una base de datos para una universidad también puede ayudar a mejorar la seguridad de los datos y reducir el riesgo de pérdida o alteración de datos.