Tecnología y educación permanente

Materia:
BASES DE DATOS

Profesor:

Rolando Antonio Del Rosario Mariñez

Id:
10153998

Estudiante:
Esmerly Rafael Fernando Roman perez

Santiago de los caballeros, República Dominicana

CONTENIDO

- Objetivo del proyecto
- Alcance
- Pasos a seguir
- Esquemas
  - Esquema E/R
  - Esquema Relacional
- Diagrama entidad relacion
- Tablas
  - Entidades (y superentidades)
  - Relaciones
- Funciones y procedimientos
- Disparadores
- Inserción de datos
- Diseño lógico
- Diseño conceptual

Descripción General

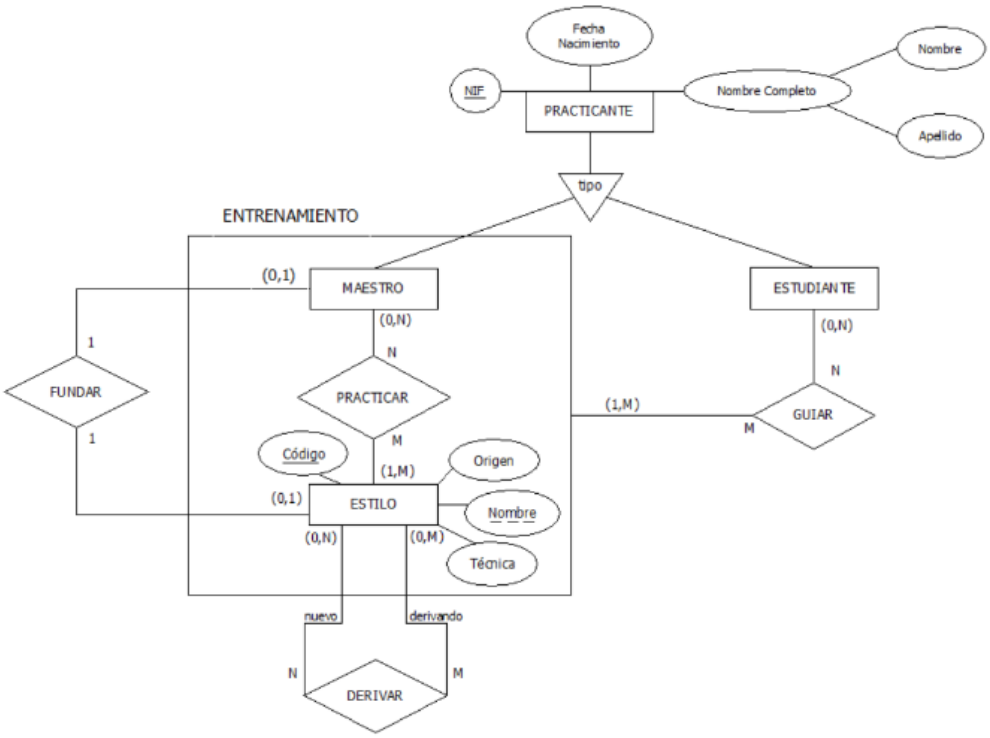
Objetivo del Proyecto:

Este proyecto tiene como objetivo gestionar la información académica de estudiantes, profesores, materias y becas, facilitando el registro y seguimiento de inscripciones y becas otorgadas.

Alcance:

La base de datos cubre el almacenamiento y manejo de información de estudiantes y profesores, así como el seguimiento de materias y becas. Se incluye una tabla adicional para gestionar las inscripciones de estudiantes en las materias.

ESQUEMA



En este modelo he supuesto las siguientes características:

TIPO: Es una jerarquía parcial solapada. Un practicante puede ser tanto maestro como estudiante a la vez (un maestro que quiera aprender nuevos estilos) y además no todos los practicantes tienen porqué ser de alguno de los dos tipos, pues se puede dar por ejemplo el caso de que un estudiante halla terminado su entrenamiento y esté en proceso de ser maestro, pero en ese momento no es ninguno de los dos (si fuera jerarquía total, en ese momento habría que eliminarlo de practicantes y perderíamos todos sus datos: nombre, apellidos... y tendríamos que volver a pedírselos y añadirlos cuando se haga maestro).

PRACTICAR: Cada maestro tiene que practicar al menos un estilo (obvio, pues de lo contrario de qué sería maestro). Por otro lado, un estilo puede no ser practicado por ningún maestro, en el caso de que fuera antiguo y con el paso del tiempo se haya perdido.

FUNDAR: Cada estilo puede ser fundado por 1 maestro (o ninguno) y cada maestro puede fundar a lo sumo, un estilo, pues este será el que representa su forma de hacer Kung Fu, no tiene sentido que pueda fundar más de un estilo.

DERIVAR: Cada estilo puede derivar de varios otros estilos. Como la mayoría (según mi suposición) no derivan de otros estilos es mejor crear una relación derivar, en vez de tener un atributo derivados en la entidad estilo, para así poder evitarnos la aparición de muchos nulos. Además, un estilo podría derivar de más de un estilo, por lo que el atributo derivados tendría que ser multivaluado, y nos daría problemas a la hora de hacer la tablas.

GUIAR : Cada estudiante sigue mínimo un entrenamiento, de lo contrario no sería estudiante y sería un practicante. Y cada entrenamiento puede ser seguido por varios estudiantes, aunque también se puede dar el caso de que ninguno lo siga, clase vacía.

ESQUEMA RELACION:

Notación: clave primaria, clave alterna.

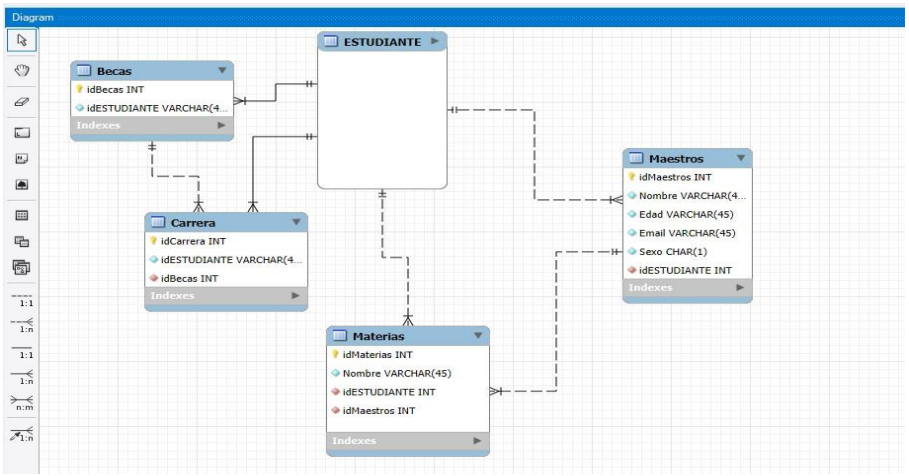
- PRACTICANTES (**NIF**, nombre, primer_apellido, segundo_apellido, fecha_nacimiento)
- ESTILOS (**codigo**, nombre, origen, tipo)
- ENTRENAMIENTOS (**NIF_maestro, codigo_estilo**)
  - entrenamientos(NIF_maestro) ⟶ practicantes(NIF)
  - entrenamientos(codigo_estilo) ⟶ estilos(codigo)
- GUIAR (**NIF_estudiante, NIF_maestro, codigo_estilo**)
  - guiar(NIF_estudiante) ⟶ practicantes(NIF)
  - guiar(NIF_maestro,codigo_estilo) ⟶ entrenamientos(NIF_maestro,codigo_estilo)
- ESTUDIANTES (**NIF**)
  - estudiantes(NIF) ⟶ guiar(NIF_estudiante)
- MAESTROS (**NIF**)
  - maestros(NIF) ⟶ entrenamientos(NIF_maestro)
- DERIVAR (**codigo_estilo_nuevo, codigo_estilo_derivador**)
  - derivar(codigo_estilo_nuevo) ⟶ estilos(codigo)
  - derivar(codigo_estilo_derivador) ⟶ estilos(codigo)
- FUNDAR (**NIF_maestro**, *codigo_estilo*)
  - fundar(NIF_maestro) ⟶ maestros(NIF)
  - fundar(codigo_estilo) ⟶ estilos(codigo)

En todas las relaciones el borrado y la modificación está puesto en cascade. Los motivos de borrado son claros:

Si un practicante se borra de todos sus clases entonces es obvio que se borrará de estudiantes.
Si un maestro se borra de todos los entrenamientos que daba, entonces es obvio que ya no será maestro (esto puede ocurrir si por ejemplo se cambia de aldea y pasa a tomar clase en otra escuela, en la nueva seguirá siendo maestro pero aquí ya no tiene sentido que siga siendo maestro si no imparte ninguna clase).
Si eliminamos un estilo, entonces se borrarían las respectivas derivaciones, pues ese estilo ya no existiría.
En fundar, si ya no existe ese estilo o ese maestro, tampoco tiene sentido la tupla fundar.
Análogamente en entrenamientos, si no existen dichos estilos (por que se introdujeron mal los datos o por cualquier motivo) también eliminaremos la respectivas tuplas, pues de lo contrario estaríamos entrenando un estilo que no existe.
y los de modificación seguirían un razonamiento análogo al de borrado.

Para llevar a cabo las tablas he supuesto que, al contrario que el enunciado, la tabla 'PRACTICANTES' puede contener individuos que no estén en clases. ¿Por qué? El hecho es el siguiente, imaginemos que hay un personaje apuntado en clase y cuando lo termina, en el proceso de apuntarse a otro o hacerse maestro, lo eliminamos de PRACTICANTES. Entonces habremos borrado todos sus datos (NIF, nombre, ...) y al volverlo a introducir (a lo mejor minutos más tarde) tendríamos que volver a introducir todos sus datos. Por este motivo he querido mantener una tabla PRACTICANTES con todos los datos personales, y luego individualmente según sean ESTUDIANTES o MAESTROS se les va añadiendo a un lado o a otro.

Diagrama de Entidad-Relación (ERD)



Descripción de Tablas

Entidades y super entidades

```sql
CREATE TABLE IF NOT EXISTS practicantes (
        NIF VARCHAR(9),
        nombre VARCHAR(40) NOT NULL,
        primer_apellido VARCHAR(40) NOT NULL,
        segundo_apellido VARCHAR(40) NOT NULL,
        fecha_nacimiento DATE NOT NULL,
        PRIMARY KEY (NIF)
);

CREATE TABLE IF NOT EXISTS estilos (
        codigo INT AUTO_INCREMENT,
        nombre VARCHAR(40) NOT NULL UNIQUE,
        origen ENUM('S','N') NOT NULL, -- S = sur, N = norte
        tipo ENUM('I','E','IE') NOT NULL, -- I = interno, E = externo, IE = ambos estilos (interno y e
        PRIMARY KEY (codigo)
);

CREATE TABLE IF NOT EXISTS entrenamientos (
        NIF_maestro VARCHAR(9),
        codigo_estilo INT,
        PRIMARY KEY (NIF_maestro, codigo_estilo),
        FOREIGN KEY (NIF_maestro) REFERENCES practicantes(NIF) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (codigo_estilo) REFERENCES estilos(codigo) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS estudiantes (
        NIF VARCHAR(9),
        PRIMARY KEY (NIF),
        FOREIGN KEY (NIF) REFERENCES guiar(NIF_estudiante) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS maestros (
        NIF VARCHAR(9),
        PRIMARY KEY (NIF),
        FOREIGN KEY (NIF) REFERENCES entrenamientos(NIF_maestro) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Relaciones

```
CREATE TABLE IF NOT EXISTS guiar (
        NIF_estudiante VARCHAR(9),
        NIF_maestro VARCHAR(9),
        codigo_estilo INT,
        PRIMARY KEY (NIF_estudiante, NIF_maestro, codigo_estilo),
        FOREIGN KEY (NIF_estudiante) REFERENCES practicantes(NIF) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (NIF_maestro,codigo_estilo) REFERENCES entrenamientos(NIF_maestro,codigo_estilo)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS derivar (
        codigo_estilo_nuevo INT,
        codigo_estilo_derivador INT,
        PRIMARY KEY (codigo_estilo_nuevo, codigo_estilo_derivador),
        FOREIGN KEY (codigo_estilo_nuevo) REFERENCES estilos(codigo) ON DELETE CASCADE ON UPDATE CASCAD
        FOREIGN KEY (codigo_estilo_derivador) REFERENCES estilos(codigo) ON DELETE CASCADE ON UPDATE CA
);

CREATE TABLE IF NOT EXISTS fundar (
        NIF_maestro VARCHAR(9),
        codigo_estilo INT,
        PRIMARY KEY (NIF_maestro),
        UNIQUE (codigo_estilo),
        FOREIGN KEY (NIF_maestro) REFERENCES maestros(NIF) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (codigo_estilo) REFERENCES estilos(codigo) ON DELETE CASCADE ON UPDATE CASCADE
);
```

TABLA 1 ES LA DEL ESTUDIANTE

Nombre de la Tabla: Estudiante
Propósito: Almacena la información personal de los estudiantes.

Columnas:
ID_Estudiante: INT, PRIMARY KEY, AUTO_INCREMENT
Descripción: Identificador único del estudiante.

Nombre: VARCHAR(100), NOT NULL
Descripción: Nombre completo del estudiante.

Fecha_Nacimiento: DATE
Descripción: Fecha de nacimiento del estudiante.

Email: VARCHAR(255), UNIQUE
Descripción: Correo electrónico del estudiante.

TABLA 1 ES LA DEL PROFESOR

Nombre de la Tabla: Profesor
Propósito: Almacena la información de los profesores.

Columnas:
ID_Profesor: INT, PRIMARY KEY, AUTO_INCREMENT
Descripción: Identificador único del profesor.

Nombre: VARCHAR(100), NOT NULL Descripción: Nombre completo del profesor.

Departamento: VARCHAR(100)
Descripción: Departamento al que pertenece el profesor.

TABLA DE MATERIA

Nombre de la Tabla: Materia
Propósito: Almacena la información sobre las materias disponibles.

Columnas:
ID_Materia: INT, PRIMARY KEY, AUTO_INCREMENT
Descripción: Identificador único de la materia.

Nombre: VARCHAR(100), NOT NULL
Descripción: Nombre de la materia.

Creditos: INT
Descripción: Número de créditos asignados a la materia.

TABLA DE BECA

Nombre de la Tabla: Beca
Propósito: Almacena información sobre las becas ofrecidas.
Columnas:
ID_Beca: INT, PRIMARY KEY, AUTO_INCREMENT
Descripción: Identificador único de la beca
.
Nombre: VARCHAR(100), NOT NULL
Descripción: Nombre de la beca.

Monto: DECIMAL(10,2)
Descripción: Monto económico asociado a la beca.

Requisitos: TEXT
Descripción: Requisitos necesarios para aplicar a la beca.

FUNCION Y PROCEDIMIENTO

Por otro lado tenemos una serie de funciones para poder sacar información de nuestra base de datos. El código en SQL para crear estas funciones es el siguiente. Primero redefinimos DELIMITER para poder utilizar el punto y coma en el interior de las funciones, procedimientos y triggers y al final de sus definiciones lo volvemos a redefinir.

Ahora iremos definiendo una por una las funciones y procedimientos creados:

Hallar los personajes que no son ni maestros ni están asignados a ningún entrenamiento por el momento:

```
CREATE PROCEDURE get_0_training()
        BEGIN
        SELECT * FROM practicantes
                WHERE NIF NOT IN (SELECT NIF FROM estudiantes) AND NIF NOT IN (SELECT NIF FROM maestros
        END$$
```

Hallar el número de entrenamientos que sigue un estudiante:

```
CREATE FUNCTION get_num(var_NIF VARCHAR(9)) RETURNS INT
        BEGIN
        RETURN (SELECT COUNT(NIF_estudiante) FROM guiar WHERE NIF_estudiante = var_NIF);
        END$$
```

Mostrar los estudiantes que siguen un cierto entrenamiento:

```
CREATE PROCEDURE show_students(NIF VARCHAR(9), codigo INT)
        BEGIN
        SELECT * FROM practicantes
                WHERE NIF IN (SELECT NIF_estudiante FROM guiar WHERE NIF_maestro = NIF AND codigo_estil
        END$$
```

Mostrar los estudiantes con más de $ n $ entrenamientos:

```
CREATE PROCEDURE students_n(n INT)
        BEGIN
        SELECT * FROM practicantes
                WHERE get_num(NIF) > n;
        END$$
```

Hallar el porcentaje de estudiantes que sigue algún entrenamiento de un estilo sin fundador:

```
CREATE FUNCTION get_percentage_no_founder() RETURNS FLOAT
        BEGIN
        DECLARE n INT;
        DECLARE total INT;
        DECLARE p FLOAT;
        SET n = (SELECT COUNT(NIF_estudiante) FROM
                                (SELECT DISTINCT NIF_estudiante FROM guiar
                                        WHERE codigo_estilo NOT IN (SELECT DISTINCT codigo_estilo FROM
        SET total = (SELECT COUNT(NIF) FROM estudiantes);
        SET p = n/total;
        RETURN p;
        END$$
```

DISPARADORES

Después de insertar algún valor en la tabla guiar nos aseguramos de añadirla luego a la tabla estudiantes (el trigger tiene que ser después de la acción pues si no daría un error de foreign key, pues estudiantes(NIF) apunta a guiar(NIF_estudiante)). Es decir, para no tener que estar insertando valores manualmente cada vez que insertemos en guiar, este disparador nos asegura que todos los practicantes que estén apuntados en algún entrenamiento, estén guardados en la tabla de estudiantes.

```
CREATE TRIGGER after_insert_guiar AFTER INSERT ON guiar FOR EACH ROW
BEGIN
IF NEW.NIF_estudiante NOT IN (SELECT NIF FROM estudiantes) THEN
        INSERT INTO estudiantes VALUES (NEW.NIF_estudiante);
END IF;
END$$
```

Después de insertar valores en la tabla de entrenamientos, si el maestro en cuestión no está en la tabla de maestros, entonces lo añadimos (este trigger daría errores si lo pondríamos antes de la ejecución pues daría un error de foreing key de que maestro(NIF) no 'encuentra' entrenamientos(NIF_maestro))

```
CREATE TRIGGER after_insert_entrenamientos AFTER INSERT ON entrenamientos FOR EACH ROW
BEGIN
IF NEW.NIF_maestro NOT IN (SELECT NIF FROM maestros) THEN
        INSERT INTO maestros VALUES (NEW.NIF_maestro);
END IF;
END$$
```

Antes de insertar una tupla en la tabla fundar, es decir, introducir que un cierto practicante ha fundado un cierto estilo, añadiríamos antes a la tabla entrenamientos que ese practicante practica ese nuevo estilo (ya que no tendría sentido que no lo practique si él mismo es el que lo ha fundado). Como podemos observar, para que un practicante funde un estilo, este no ha de ser maestro previamente aunque fundar(NIF_maestro) apunte a maestros(NIF), pues antes de insertar en fundar, este trigger inserta un entrenamiento y otro disparador,after_insert_entrenamientos, inserta el NIF en maestros. Lo que nos asegura que fundar(NIF_maestro) apunte a maestros(NIF) es, que este proceso se ha ejecutado correctamente y que si hay un practicante en fundar, este está en la tabla maestros también.

Este disparador se lanza si ya existe dicho estilo (para no añadir un entrenamiento antes de lanzar un error y cambiar datos) y obviamente si dicho entrenamiento no está ya añadido en la tabla entrenamientos (que quien manipule la base de datos no sepa que eso era automático y lo haya añadido antes manualmente).

```
CREATE TRIGGER before_insert_fundar BEFORE INSERT ON fundar FOR EACH ROW
        BEGIN
        IF (NEW.NIF_maestro, NEW.codigo_estilo) NOT IN
                (SELECT NIF_maestro, codigo_estilo FROM entrenamientos)
                        AND codigo_estilo IN (SELECT codigo FROM estilos) THEN
                                INSERT INTO entrenamientos VALUES (NEW.NIF_maestro, NEW.codigo_estilo);
        END IF;
        END$$
```

Depués de eliminar una tupla de la tabla fundar, eliminamos el entrenamiento que añadimos en su momento al añadir la tupla. Esta acción ha de hacerse después pues en el caso de ser antes, se lanzaría este disparador eliminaría el entrenamiento en cuestión y además la eliminación en cascada de mestros, si solo había ese entrenamiento con ese maestro, eliminaría el maestro de la tabla maestros y por lo tanto, es este mismo momento (y hasta que se ejecute finalmente la acción de eliminar la tupla de fundar) tendríamos que NEW.NIF_maestro no apunta a ningún maestro, por lo que obtendríamos un error.

```
CREATE TRIGGER after_delete_fundar AFTER DELETE ON fundar FOR EACH ROW
BEGIN
DELETE FROM entrenamientos WHERE NIF_maestro = OLD.NIF_maestro AND codigo_estilo = OLD.codigo_estilo;
END$$
```

Antes de insertar en guiar nos aseguramos de que el estudiante no se apunte a aprender un estilo del cual ya es maestro y de que no se de clase a sí mismo (esto último se podría hacer con un check en la propia tabla, pero razones de SQL y que ya había claves foráneas no se podía (Error code: 3823)).

```
CREATE TRIGGER before_insert_guiar_check BEFORE INSERT ON guiar FOR EACH ROW
        BEGIN
        IF NEW.NIF_estudiante LIKE NEW.NIF_maestro THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un maestro no se puede dar clase a sí mismo
        END IF;
        IF NEW.NIF_estudiante IN (SELECT NIF FROM maestros) THEN
                IF (NEW.codigo_estilo, NEW.NIF_estudiante) IN
                        (SELECT codigo_estilo, NIF_maestro FROM entrenamientos) THEN
                                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
                                'Un estudiante no puede entrenar un estilo del que ya es maestro';
                END IF;
        END IF;
        END$$
```

Antes de insertar en derivar, vemos que tenga sentido la tupla introducida, es decir, que un estilo no puede derivar de uno y a su vez ser derivador de ese mismo y tampoco, obviamente, derivar de sí mismo.

```
CREATE TRIGGER before_insert_derivar BEFORE INSERT ON derivar FOR EACH ROW
        BEGIN
        IF NEW.codigo_estilo_nuevo LIKE NEW.codigo_estilo_derivador THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un estilo no puede derivar de sí mismo';
        END IF;
        IF (NEW.codigo_estilo_derivador, NEW.codigo_estilo_nuevo) IN
                (SELECT codigo_estilo_nuevo, codigo_estilo_derivador FROM derivar) THEN
                        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
                                'Un estilo no puede derivar de uno y a su vez ser derivador de ese mismo
        END IF;
        END$$
```

INICIO DE DATOS

Finalmente añadimos una serie de datos random para poder probar nuestra base de datos. Para ello creamos la tabla practicantes y los estilos desde la página mockaroo y apartir de esas dos tablas, con el script insertarSQL.py de python (guardado en la carpeta data), nos rellenamos las demás tablas con sentido, es decir, cumpliendo todas las restricciones impuestas y finalmente generamos una función (en ese mismo script) que nos devuelva un texto para poder introducir todos estos datos por SQL. El código en python sin entrar en detalles sería el siguiente y gracias a él generamos el código en SQL de insertar_datos.sql..

Primero cargamos los datos recogidos de mockaroo (guardados en los archivos practicantes.csv y estilos.csv).

```
import pandas as pd
import numpy as np

main_path = "data/"
f1 = "practicantes.csv"
f2 = "estilos.csv"

practicantes = pd.read_csv(f1)
estilos = pd.read_csv(f2)
estilos["codigo"] += 1

np.random.seed(99)
```

Generamos los datos de clases y maestros

```
entrenamientos = []
maestros = []
d_entrena = {}
for i in range(80):
        nif, cod = np.random.choice(practicantes["NIF"]), np.random.choice(estilos["codigo"])
        if d_entrena.get(nif) == None:
                d_entrena[nif] = [cod]
        else:
                d_entrena[nif].append(cod)
for nif in d_entrena:
        d_entrena[nif] = list(set(d_entrena[nif]))
        for cod in d_entrena[nif]:
                maestros.append(nif)
                entrenamientos.append([nif, cod])

maestros = list(set(maestros))
```

Generamos los datos de fundar

```
fundar = []
nifs = np.random.choice(maestros, 15)
cods = np.random.choice(estilos["codigo"], 15)
nifs = list(set(nifs))
cods = list(set(cods))
n_min = min(len(nifs), len(cods))
d_fundar = {}
for i in range(n_min):
        nif, cod = nifs[i], cods[i]
        fundar.append([nif, cod])
        if d_fundar.get(nif) == None:
                d_fundar[nif] = [cod]
        else:
                d_fundar[nif].append(cod)
```

Generamos los datos de derivar

```
derivar = []
cods = [np.random.choice(estilos["codigo"]) for i in range(10)]
cods  = list(set(cods))
if len(cods) % 2 != 0: cods = cods[:-1]
derivar = [[cods[2*i], cods[2*i+1]] for i in range(int(len(cods)/2))]
```

Generamos los datos de guiar

```
guiar = []
nif_estudiantes = np.random.choice(practicantes["NIF"], 200)
entrenas_index = np.random.randint(0, len(entrenamientos), 200)
entrenas = [entrenamientos[i] for i in entrenas_index]

d = {}
for i in range(200):
        if d.get(str(entrenas[i])) == None:
                d[str(entrenas[i])] = [nif_estudiantes[i]]
        else:
                d[str(entrenas[i])].append(nif_estudiantes[i])
for entrena in d:
        for NIF_est in set(d[entrena]):
                s = entrena.split("'")
                nif_mae, cod = s[1], int(s[2][1:-1])
                if d_entrena.get(NIF_est) != None:
                        if cod in d_entrena[NIF_est]:
                                continue
                if d_fundar.get(NIF_est) != None:
                        if cod in d_fundar[NIF_est]:
                                continue
                if NIF_est != nif_mae:
                        guiar.append([NIF_est, nif_mae, cod])
```

Definimos un serie de funciones para a través de un dataframe o una lista, devolver un comando SQL para generar dicha tabla.

```
  def insert_df_in_SQL(df, table, integers=[], drop=False):
          insert = ""
          insert += f"INSERT INTO {table} VALUES"
          for i in range(len(df)):
                  string = ""
                  for j in range(len(df[0])):
                          if drop and j == 0:
                                  continue
                          if j in integers:
                                  string += str(df[i][j]) + ","
                          else:
                                  string += "'" + str(df[i][j]) + "',"
                  string = string[:-1]
                  insert += f"\n\t({string}),"
          insert = insert[:-1]
          insert += "\n;"
          print(insert)

  def insert_list_in_SQL(lista, table):
          insert = ""
          insert += f"INSERT INTO {table} VALUES "
          for i in range(len(lista)):
                  insert += f'\n\t("{lista[i]}"),'
          insert = insert[:-1]
          insert += "\n;"
          print(insert)
```

Imprimimos los datos por pantalla

```
insert_df_in_SQL(practicantes.values, "practicantes")
insert_df_in_SQL(estilos.values, "estilos(nombre,origen,tipo)", [0], True)
insert_df_in_SQL(fundar, "fundar", [1])
insert_df_in_SQL(derivar, "derivar", [0,1])
insert_df_in_SQL(entrenamientos, "entrenamientos", [1])
insert_df_in_SQL(guiar, "guiar", [2])
```

Consultas SQL Ejemplares

Consulta de Selección: Obtener todos los estudiantes inscritos en una materia específica:

```
SELECT e.Nombre, i.Fecha_Inscripcion
FROM Estudiante e
JOIN Inscripcion i ON e.ID_Estudiante = i.ID_Estudiante
WHERE i.ID_Materia = ?;
```

Consulta de Inserción: Agregar un nuevo estudiante:

```
INSERT INTO Estudiante (Nombre, Fecha_Nacimiento, Email)
VALUES ('Juan Pérez', '2000-01-15', 'juan.perez@example.com');
```

Consulta de Actualización: Actualizar el email de un profesor:

```
UPDATE Profesor
SET Email = 'nuevo.email@example.com'
WHERE ID_Profesor = ?;
```

Consulta de Eliminación: Eliminar una inscripción:

```
DELETE FROM Inscripcion
WHERE ID_Inscripcion = ?;
```

Procedimientos y Triggers

Procedimientos Almacenados: Describe cualquier procedimiento almacenado utilizado.

```
CREATE PROCEDURE ObtenerEstudiantesPorMateria(IN materiaID INT)
BEGIN
   SELECT e.Nombre, i.Fecha_Inscripcion
   FROM Estudiante e
   JOIN Inscripcion i ON e.ID_Estudiante = i.ID_Estudiante
   WHERE i.ID_Materia = materiaID;
END;
```

Consideraciones de Rendimiento

Índices:
Añadir índices en ID_Estudiante e ID_Materia en la tabla Inscripcion para mejorar el rendimiento de las búsquedas.
Optimización: Revisión de consultas y ajuste de índices según sea necesario para mantener el rendimiento.

DIEÑO LOGICO

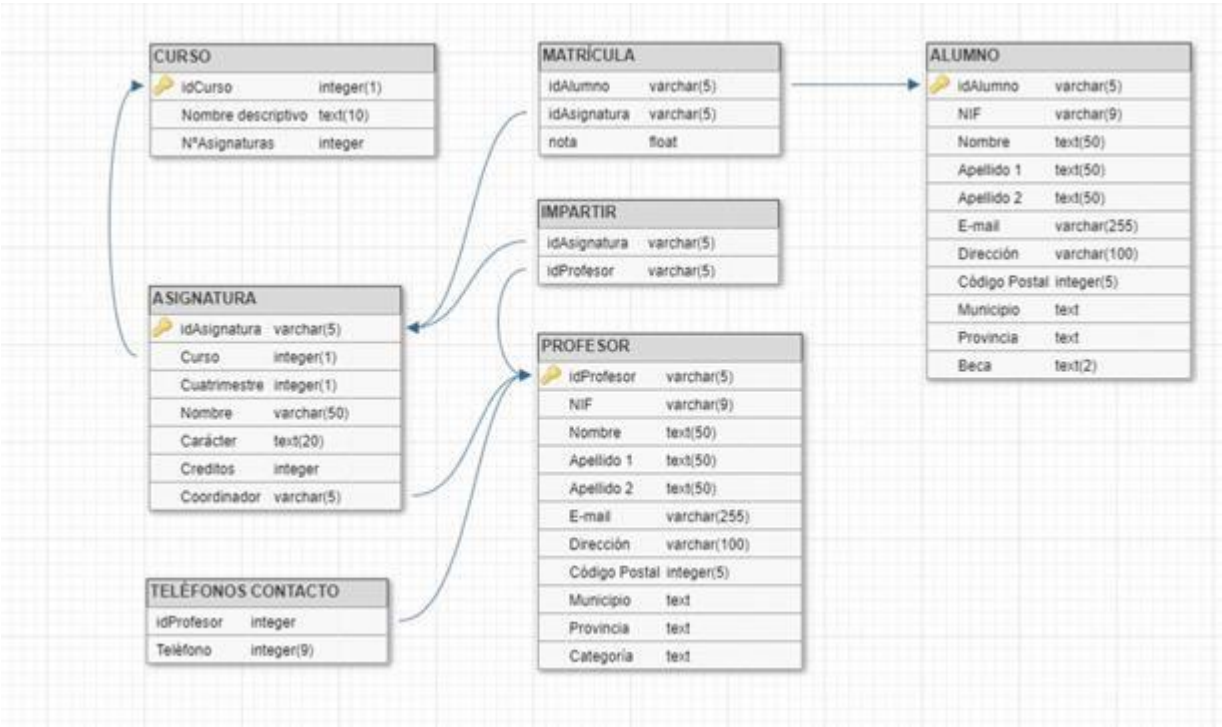


Ilustración 1. Modelo relacional

Rescato la ilustración 2 para la explicación:

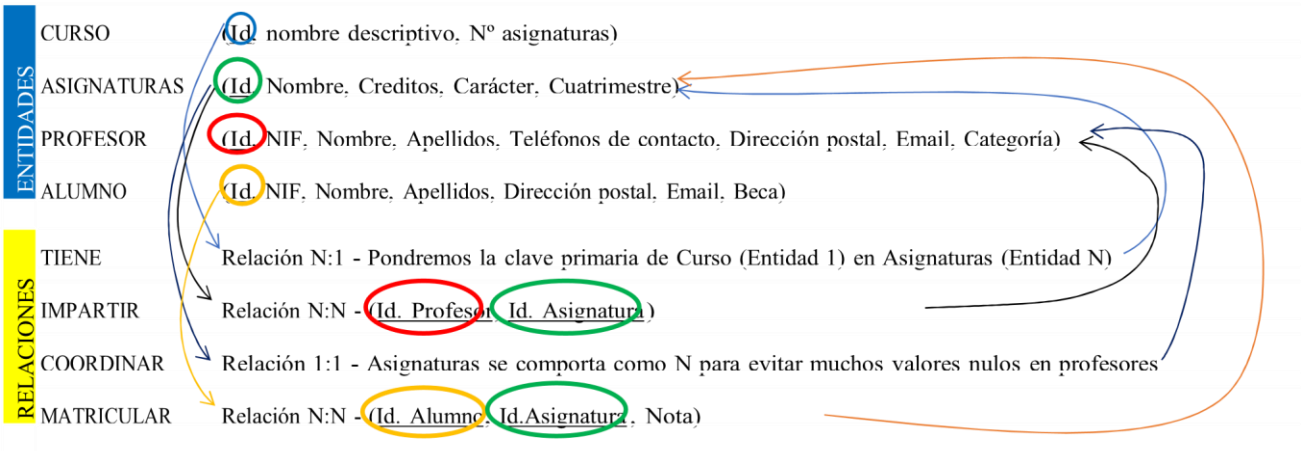


Ilustración 2. Tabla de entidades y relaciones

Hemos creado una tabla por entidad con sus atributos como columnas. El único atributo con tabla propia es "teléfono de contacto" de profesor al ser un atributo multivalorado. En ella encontraremos el idProfesor y cada uno de sus número de contacto como un registro.
Las relaciones que se establecen en este modelo son:

- Tiene: Relaciona la entidad curso y asignatura. Al ser una relación de N asignaturas en 1 curso, añadiremos una columna nueva en asignaturas que contendrá la clave foránea idCurso. De esta manera relacionaremos las dos tablas.
- Impartir: Relaciona profesor y asignatura. Al ser una relación varios a varios, crearemos una nueva tabla que contenga idProfesor e idAsignatura. De esta manera conectamos las dos entidades.
- Coordinar: Relaciona profesor y asignatura con una relación 1:1. Dado que no todos los profesores son coordinadores, para evitar valores nulos, trataremos asignaturas como entidad N de la relación y le añadiremos la columna "coordinador" con idProfesor del coordinador como clave foránea.
- Matricular: Relaciona alumno y asignatura. Además, cuenta con el atributo nota que ha sacado el alumno en la asignatura matriculada. Como se establece una relación varios a varios crearemos una tabla con idAlumno, idAsignatura y nota que relacione ambas entidades.

DISEÑO CONCEPTUAL

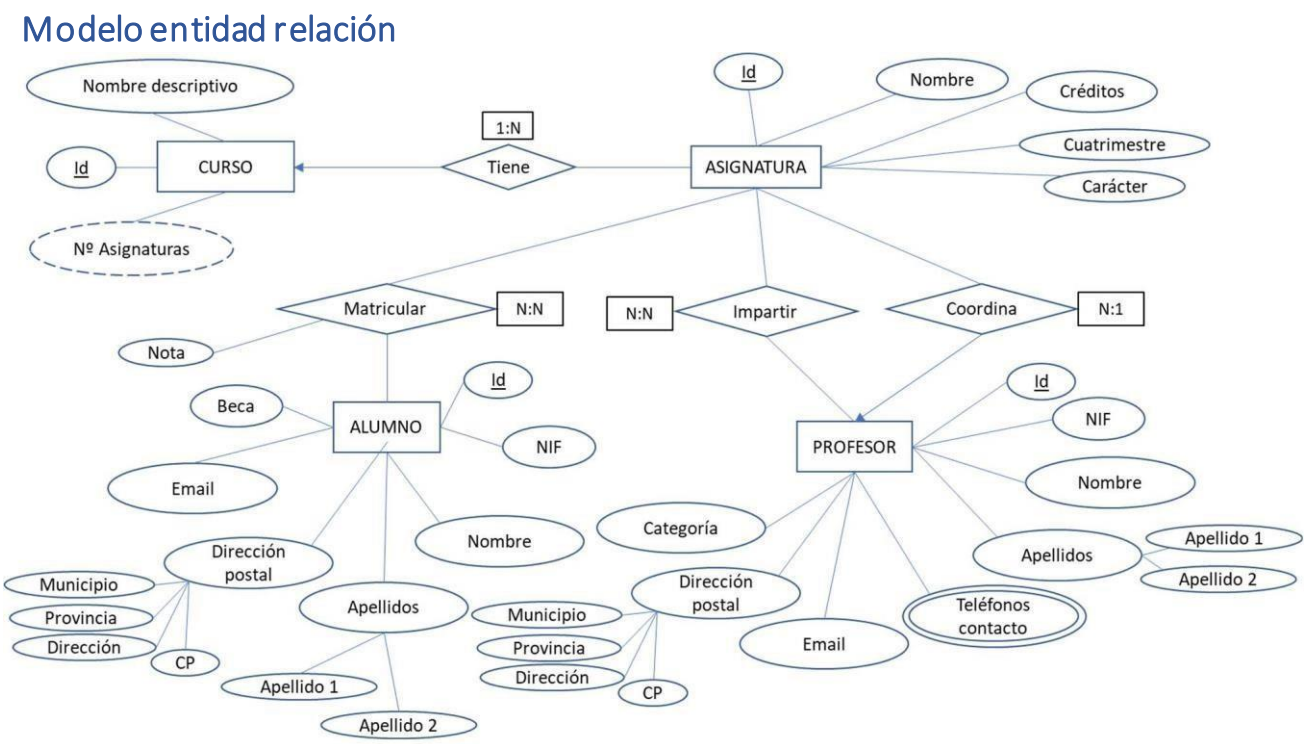


Ilustración 1. Esquema Entidad-Relación

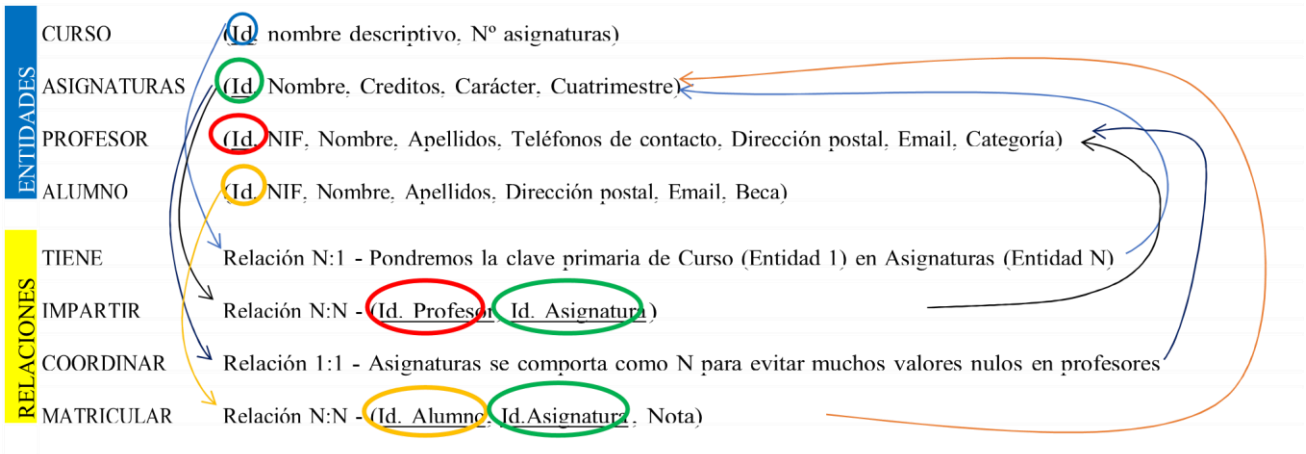


Ilustración 1.Tabla de entidades y relaciones

Entidades

Curso

La entidad curso cuenta con 3 atributos:

Id: Atributo simple y univalorado. Será la clave primaria identificativa, un número único del 1 al 6.

Nombre descriptivo: Atributo simple y univalorado. Permite identificar el curso. Tendrá formato texto y tomará los valores: primero, segundo, tercero, cuarto, máster y doctorado.

Nº Asignaturas: Atributo derivado.

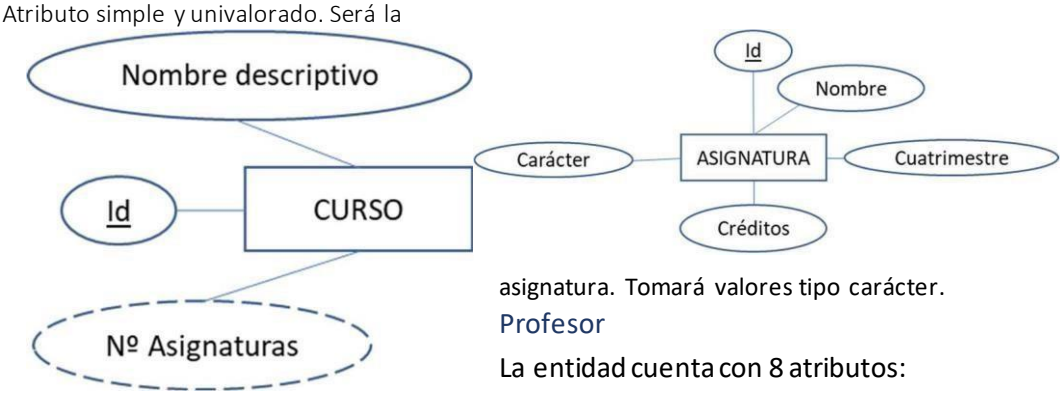
clave primaria identificativa, un varchar compuesto por "AS" y 3 números.

Nombre: Atributo simple y univalorado. Permite identificar la Dependerá de las asignaturas de la

Asignatura

La entidad cuenta con 5 atributos:

Id: Atributo simple y univalorado. Será la



asignatura. Tomará valores tipo carácter.

Profesor

La entidad cuenta con 8 atributos:

Cuatrimestre: Atributo simple y

Identifica el cuatrimestre de impartición de la asignatura. Tendrá formato numérico. Creditos: Atributo simple y univalorado. Permite identificar el número de créditos de una asignatura. Tomará valores tipo numérico real.
Carácter: Atributo simple y univalorado. Permite identificar la obligatoriedad de una asignatura, siendo las posibles opciones: Obligatoria y Optativa. Tomará valores tipo carácter.

Id: Atributo simple y univalorado. Será la clave primaria al tomar un valor único. Es el identificador único del profeso dentro de la universidad. Tomará valores un varchar

compuesto por "PR" y 3 números.

NIF: Atributo simple y univalorado.
Lo tomaremos como clave alternativa al ser un valor de identificación única. Tomará valores tipo caracter.

Nombre: Atributo simple y univalorado. Tomará valores tipo carácter.

Apellidos: Atributo compuesto y univalorado. Lo podemos descomponer en Apellido 1 y Apellido 2. Tomará valores tipo carácter de longitud fija.

Teléfonos contacto: Atributo simple y multivalorado. Un profeso puede tener varios teléfonos (fijo y móvil). Tomará valores tipo numérico y de longitud fija. Al ser multivalorado implicará la creación de una nueva tabla específica de este atributo junto con la id del profesor.

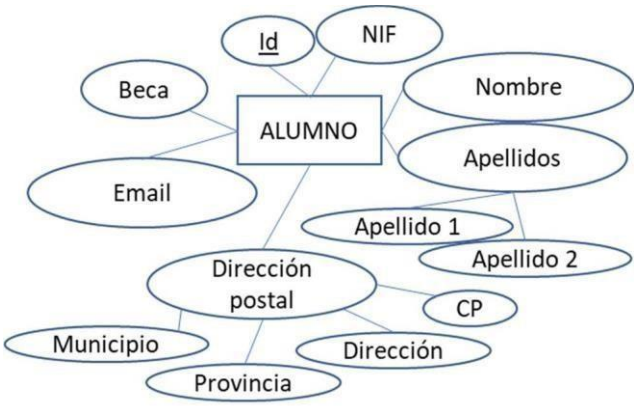
Email: Atributo simple y univalorado. Lo tomaremos como clave alternativa al ser un valor de identificación única.

Dirección postal: Atributo compuesto y univalorado. Lo podemos descomponer en: Dirección, código postal, municipio y provincia. Tomará valores alfanuméricos.

Categoría: Atributo simple y univalorado. Tomará valores numéricos dentro de un rango de opciones.

- Alumno

  La entidad cuenta con 5 atributos:



Id: Atributo simple y univalorado. Será la clave primaria al tomar un valor único. Es el identificador único del alumno dentro de la universidad. Tomará valores un varchar compuesto por "AL" y 3 números.

NIF: Atributo simple y univalorado.
Lo tomaremos como clave alternativa al ser un valor de identificación única. Tomará valores tipo numérico.

Nombre: Atributo simple y univalorado. Tomará valores tipo carácter.

Apellidos: Atributo compuesto y univalorado. Lo podemos descomponer
Entidad alumno y sus atributos en Apellido 1 y Apellido 2. Tomará valores tipo carácter.
Email: Atributo simple y univalorado. Lo tomaremos como clave alternativa al ser un valor de identificación única.

Dirección postal: Atributo compuesto y univalorado. Lo podemos descomponer en: Dirección, código postal, municipio y provincia. Tomará valores alfanuméricos.

Beca: Es un atributo simple y univalorado. Tomará valores de tipo carácter: 'si' o 'no'.

- Relaciones

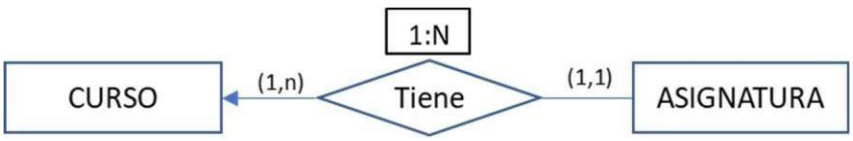


Ilustración 6. Relación tiene

Un curso como mínimo tiene 1 asignatura y como máximo varias. Una asignatura pertenece como mínimo y como máximo a 1 curso. Se establece una relación uno a varios, por ello se incluirá el id de curso en la tabla asignatura como clave foránea.

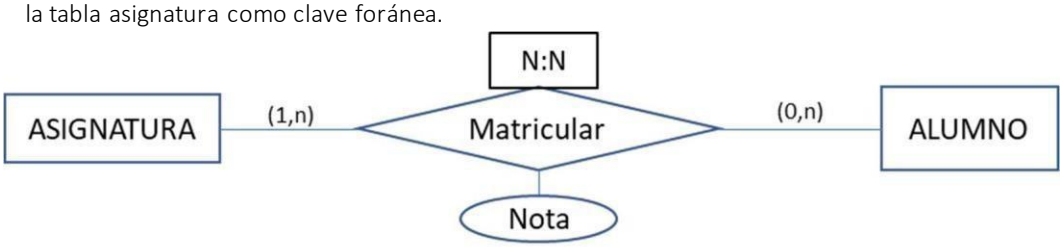


Ilustración 7. Relación matricular

Una asignatura puede tener 0 alumnos matriculados como mínimo (alguna optativa) o varios alumnos. Un alumno se matriculará al menos en 1 asignatura o varias. Además, matricular tiene un atributo que define la nota del alumno en la asignatura en la que se ha matriculado. Se establece una relación varios a varios, por ello crearemos una nueva tabla, "Matrículas", que contenga las claves primarias de las entidades que se relacionan y la nota correspondiente.

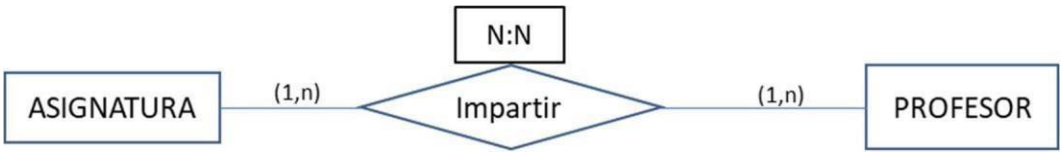


Ilustración 8. Relación impartir

Un profesor imparte al menos 1 asignatura, aunque puede impartir varias. Una asignatura es impartida como mínimo por un profesor, aunque pueden impartirla varios. Se establece una relación varios a varios, por ello crearemos una nueva tabla, "Impartir", que contenga las claves primarias de las entidades que se relacionan.

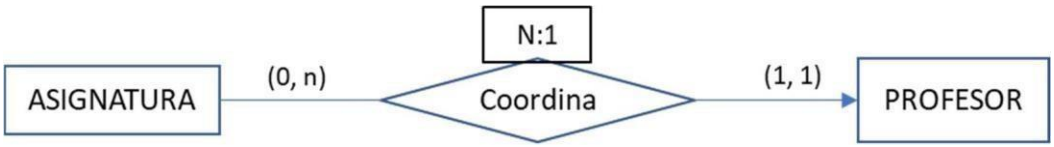


Ilustración 2. Relación coordinar

Un profesor podría coordinar ninguna 0 o N asignatura (por ejemplo, todas las asignaturas de estadística) y una asignatura tiene un único profesor-coordinador. Se establece una relación varios a uno, por ello se incluirá en la tabla asignatura una columna con la id del coordinador.

Mantenimiento y Backup

Plan de Mantenimiento: Revisión periódica de índices y actualización de estadísticas. Backup: Estrategia de respaldo regular (diaria, semanal) con almacenamiento en un lugar seguro. Considerar respaldo completo y diferencial.