Bases de Datos I

Profesor: Gustavo Ramoscelli

DIAGRAMA DE ESQUEMA

Diagrama de Esquema:

- Cada relación aparece como un cuadro con el nombre de la relación en la parte superior en un recuadro gris y los atributos en el interior del recuadro blanco.
- Los atributos que son clave primaria se muestran subrayados.
- Las dependencias de clave externa aparecen como flechas desde los atributos de clave externa de la relación referenciante a la clave primaria de la relación referenciada.
- Las restricciones de integridad referencial distintas de las restricciones de clave externa no se muestran explícitamente en los diagramas de esquema.

DIAGRAMA DE ESQUEMA

Ejemplo:

Modelar una biblioteca pequeña donde se prestan libros a usuarios de la biblioteca. Además los usuarios pagan una cuota mensual.

- **Usuarios**. Almacena la información de los usuarios de la biblioteca.
- **Libros**. Almacena la información de los libros disponibles.
- **Prestamos**. Registra los préstamos de libros realizados por los usuarios.
- **Pagos**. Registra los pagos de cuotas de los usuarios.

DIAGRAMA DE ESQUEMA

Ejemplo:

- Descargar del aula virtual el script biblioteca.sql para generar la base de datos.
- Luego correr el script en una base de datos mysql para crear las tablas.
- Insertar algunos datos para popular las tablas
- Extraer el diagrama de esquema de base de datos usando *mysql workbench*

NOTA: en caso de no tener instalado el programa mysql workbench, puede descargar el archivo biblioteca.dbml del aula virtual y usar la página *https://dbdiagram.io/*

DIAGRAMA DE ESQUEMA

¿Qué es DBML?

- **DBML o Database Markup Language**, es un lenguaje simple y declarativo para definir bases de datos relacionales.
- Se usa para generar diagramas de bases de datos de manera rápida y eficiente.
- Popular en plataformas como dbdiagram.io para modelado visual.

DIAGRAMA DE ESQUEMA

¿Qué es DBML?

Soporte para:

- Tablas
- Claves primarias y foráneas
- Relaciones entre tablas
- Comentarios y descripciones

DIAGRAMA DE ESQUEMA

Ejemplo básico

- Este ejemplo define una tabla llamada **Usuarios** con cuatro columnas.
- La columna **id_usuario** es la clave primaria (**pk**) y se *autoincrementa*.

```
Table Usuarios {
  id_usuario INT [pk, increment]
  nombre VARCHAR
  email VARCHAR
  fecha_registro DATE
}
```

DIAGRAMA DE ESQUEMA

Relaciones entre tablas

- Definición de tablas **Prestamos** y **Pagos**.
- Relaciones de clave foránea (**ref**) hacia la tabla **Usuarios**.

```
Table Prestamos {
  id_prestamo INT [pk, increment]
  id_usuario INT [ref: > Usuarios.id_usuario]
  fecha_prestamo DATE
}

Table Pagos {
  id_pago INT [pk, increment]
  id_usuario INT [ref: > Usuarios.id_usuario]
  monto DECIMAL
  fecha_pago DATE
}
```

Claves foráneas: `ref: >` para establecer relaciones entre tablas, en general uno a muchos.

DIAGRAMA DE ESQUEMA

Relaciones entre tablas

- Para la relación muchos a muchos usamos una tabla de relación

```
Table Usuarios {
  id_usuario INT [pk, increment]
  nombre VARCHAR
}

Table Libros {
  id_libro INT [pk, increment]
  titulo VARCHAR
}

Table Usuarios_Libros {
  id_usuario INT [ref: > Usuarios.id_usuario]
  id_libro INT [ref: > Libros.id_libro]
}
```

DIAGRAMA DE ESQUEMA

Relaciones entre tablas

- Para la relación uno a uno hay que agregar el atributo unique

```
Table Usuarios {
  id_usuario INT [pk, increment]
  nombre VARCHAR
}

Table Libros {
  id_libro INT [pk, increment]
  titulo VARCHAR
}

Table Usuarios_Libros {
  id_usuario INT [ref: > Usuarios.id_usuario]
  id_libro INT [ref: > Libros.id_libro]
}
```

DIAGRAMA DE ESQUEMA

Se puede usar la página dbdiagram.io o la herramienta MySQL Workbench para realizar los diagramas de esquema de tablas.

Lenguaje SQL

Tipos básicos en SQL:

- **char(n).** Cadena de caracteres de longitud fija
- **varchar(n)**. Cadena de caracteres de longitud variable
- **int, smaillint**
- **numeric(p,d)**. número de coma fija: p dígitos (más el signo), y de esos p dígitos, d pertenecen a la parte decimal.
- **real, double precision.**
- **float**(n). número de coma flotante cuya precisión es, al menos, de n dígitos.

Lenguaje SQL

Atributo Multivaluado

Un atributo multivaluado es un atributo que puede tener más de un valor asociado a una sola entidad. En otras palabras, una instancia de una entidad puede tener múltiples valores para un atributo específico. Este tipo de atributo se representa mediante un óvalo doble en los diagramas ER.

Ejemplo: Atributo multivaluado: Teléfonos

- Un estudiante (entidad) puede tener múltiples números de teléfono.
- El atributo teléfonos es multivaluado porque un estudiante puede tener más de un número de teléfono (móvil, fijo, etc.).

Lenguaje SQL

Atributo Multivaluado

Este tipo de atributo no es directamente soportado por los sistemas relacionales (como las bases de datos SQL), por lo que, en la práctica, se desnormaliza a través de una entidad adicional para mantener la relación.

```sql
CREATE TABLE Estudiantes (
    id_estudiante INT PRIMARY KEY,
    nombre VARCHAR(100),
    edad INT
);

CREATE TABLE Telefonos (
    id_telefono INT PRIMARY KEY,
    id_estudiante INT,
    telefono VARCHAR(20),
    CONSTRAINT fk_estudiante FOREIGN KEY
        (id_estudiante) REFERENCES
            Estudiantes(id_estudiante)
);
```

Lenguaje SQL

Obtención de datos con select:

- **SELECT DISTINCT:** No se muestran los duplicados

```
select distinct nombre_dept from profesor;
```

- **SELECT ALL:** Se muestran todos los datos, incluso los duplicados

```
select all nombre_dept from profesor;
```

Lenguaje SQL

Funciones de Agregación

Las funciones de agregación resumen datos en conjuntos.

1. **COUNT():** Cuenta el número de filas.
2. **SUM():** Suma los valores de una columna.
3. **AVG():** Calcula el promedio de una columna.
4. **MAX():** Encuentra el valor máximo.
5. **MIN():** Encuentra el valor mínimo.

```sql
SELECT COUNT(*) FROM empleados;
SELECT SUM(salario) FROM empleados;
SELECT AVG(salario) FROM empleados;
SELECT MAX(salario) FROM empleados;
SELECT MIN(salario) FROM empleados;
```

Lenguaje SQL

Funciones de Texto

Las funciones de texto manipulan cadenas de caracteres.

1. **UPPER():** Convierte a mayúsculas.
2. **LOWER():** Convierte a minúsculas.
3. **CONCAT():** Combina dos o más cadenas
4. **LENGTH():** Devuelve la longitud de una cadena.
5. **SUBSTRING():** Extrae una subcadena.

Lenguaje SQL

Funciones de Fecha y Hora

Las funciones de fecha y hora permiten trabajar con valores temporales.

1. **NOW():** Devuelve la fecha y hora actual.
2. **CURDATE():** Devuelve la fecha actual.
3. **YEAR():** Extrae el año de una fecha.
4. **MONTH():** Extrae el mes de una fecha.
5. **DATEDIFF():** Calcula la diferencia entre dos fechas.

Formato de fecha: ‘YYYY-MM-DD’. Por ejemplo: ‘2024-09-09’

Lenguaje SQL

Funciones de Fecha y Hora

Ejemplo de DATE BETWEEN:

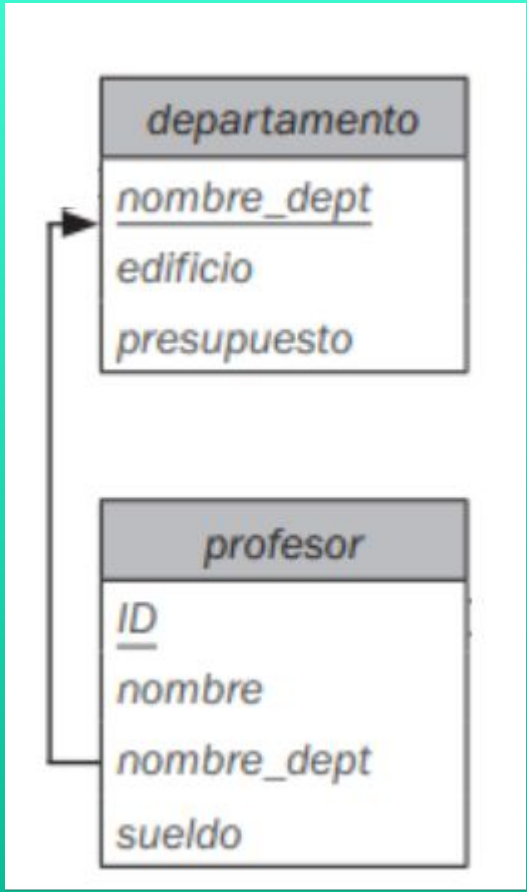
```
SELECT nombre, fecha_contrato
FROM empleados
WHERE fecha_contrato BETWEEN '2023-01-01' AND '2023-12-31';
```

Lenguaje SQL

Obtención de datos con select:

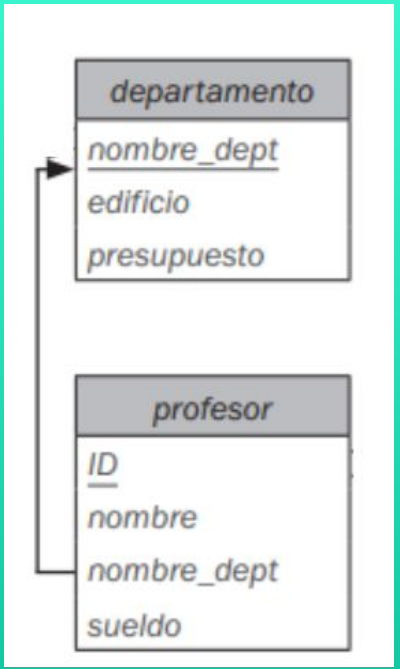
Cómo responder a la siguiente consulta:

«obtener los nombres de todos los profesores, junto con los nombres de sus departamentos y el nombre del edificio donde se encuentra el departamento».



Lenguaje SQL

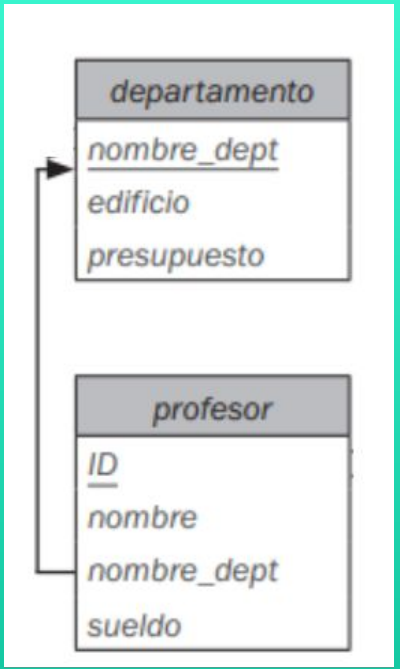
Obtención de datos con select:




```
select nombre,
    profesor.nombre_dept, edificio
    from profesor, departamento
    where profesor.nombre_dept = departamento.nombre_dept;
```

Lenguaje SQL

Obtención de datos con select:




```
select nombre,
    profesor.nombre_dept, edificio
    from profesor, departamento
    where profesor.nombre_dept = departamento.nombre_dept;
```

Lenguaje SQL

Creación de tablas:

```
create table departamento (
        nombre_dept varchar (20),
        edificio varchar (15),
        presupuesto numeric (12,2),
        primary key (nombre_dept));
```

RESTRICCIONES DE INTEGRIDAD

- Garantizan la exactitud, coherencia y validez de los datos.

- Permiten establecer reglas que los datos deben cumplir, evitando la entrada de datos incorrectos o inconsistentes.

RESTRICCIONES DE INTEGRIDAD

PRIMARY KEY: Garantiza que cada fila en una tabla tiene un valor único en la columna o combinación de columnas designadas como clave primaria. No permite valores duplicados ni nulos.

FOREIGN KEY: Asegura que los valores en una columna o conjunto de columnas coinciden con los valores en una columna de otra tabla (clave externa), manteniendo la relación entre tablas y asegurando la integridad referencial.

RESTRICCIONES DE INTEGRIDAD

UNIQUE: Obliga a que todos los valores en una columna o conjunto de columnas sean únicos, pero a diferencia de PRIMARY KEY, permite valores nulos.

NOT NULL: Evita que se inserten valores nulos en una columna específica, garantizando que siempre se proporcione un valor.

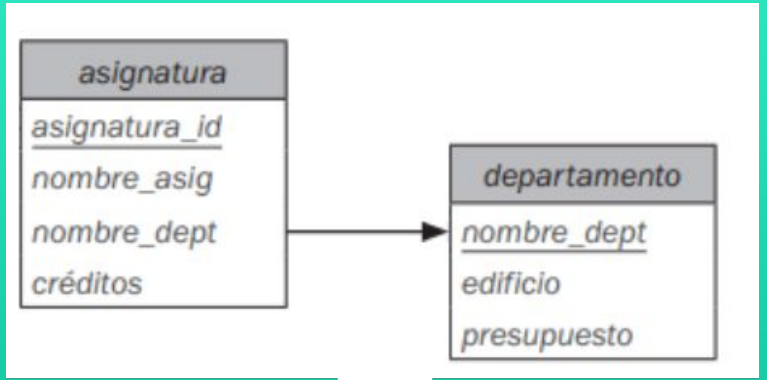
CHECK: Permite definir una condición que los datos deben cumplir. Por ejemplo, puedes asegurarte de que una columna de edad siempre contenga valores mayores que cero.

Lenguaje SQL

Restricciones de integridad

```
create table departamento
    (nombre_dept        varchar (20),
    edificio            varchar (15),
    presupuesto         numeric (12,2),
    primary key (nombre_dept));

create table asignatura
    (asignatura_id      varchar (7),
    nombre              varchar (50),
    nombre_dept         varchar (20),
    créditos            numeric (2,0),
    primary key (asignatura_id),
    foreign key (nombre_dept) references departamento);
```



Lenguaje SQL

Restricciones de integridad - Uso de CONSTRAINT

Cuando defines una restricción explícitamente usando la palabra clave CONSTRAINT, estás dando un nombre específico a esa restricción, lo que proporciona una manera más clara y controlada de gestionar las reglas de integridad.

```
CREATE TABLE Pedidos (
  id_pedido INT PRIMARY KEY,
  id_cliente INT,
  fecha_pedido DATE,
  CONSTRAINT fk_cliente FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);
```

Lenguaje SQL

Ventajas de usar CONSTRAINT:

1. **Asignar nombres a las restricciones:** Nombrar explícitamente la restricción (fk_cliente) permite identificar y modificar fácilmente la restricción en el futuro. Por ejemplo, si necesitas eliminar o cambiar esta restricción más tarde, puedes hacerlo por su nombre: `ALTER TABLE Pedidos DROP CONSTRAINT fk_cliente;`
2. **Mejor legibilidad:** Usar CONSTRAINT hace que el código sea más claro, ya que las restricciones están nombradas, lo que facilita la comprensión del propósito de la regla.
3. **Mantenimiento más fácil:** Si alguna vez tienes que depurar, modificar o gestionar la base de datos a largo plazo, tener nombres claros en las restricciones hace que sea mucho más fácil entender y trabajar con las reglas de integridad.
4. **Control explícito:** Usar CONSTRAINT también te permite definir de manera explícita no solo claves foráneas, sino también restricciones de unicidad, validaciones complejas, etc.

Lenguaje SQL

Ventajas de usar CONSTRAINT:

1. **Asignar nombres a las restricciones:** Nombrar explícitamente la restricción (fk_cliente) permite identificar y modificar fácilmente la restricción en el futuro. Por ejemplo, si necesitas eliminar o cambiar esta restricción más tarde, puedes hacerlo por su nombre: `ALTER TABLE Pedidos DROP CONSTRAINT fk_cliente;`
2. **Mejor legibilidad:** Usar CONSTRAINT hace que el código sea más claro, ya que las restricciones están nombradas, lo que facilita la comprensión del propósito de la regla.
3. **Mantenimiento más fácil:** Si alguna vez tienes que depurar, modificar o gestionar la base de datos a largo plazo, tener nombres claros en las restricciones hace que sea mucho más fácil entender y trabajar con las reglas de integridad.
4. **Control explícito:** Usar CONSTRAINT también te permite definir de manera explícita no solo claves foráneas, sino también restricciones de unicidad, validaciones complejas, etc.

Fin de la Clase...