Relaciones One-to-One y One-to-Many en SQL

Píldora educativa para entender cómo las bases de datos organizan la información de manera eficiente y escalable.

Realizado por : Esther Tapias

¿Qué son las relaciones en SQL?

Las relaciones son el corazón de las bases de datos relacionales. Nos permiten conectar diferentes tablas, organizando la información de manera lógica y eficiente, evitando duplicidades y garantizando la integridad de los datos.

Conectan tablas

Relacionan información de diferentes conjuntos de datos.

Primary Key & Foreing Key

PK : campo que identifica de manera única a ca registro de una tabla

FK : columna en una tabla que dse refiere a la PK de otra tabla, relacionando entre sí ambas tablas

Tipos clave hoy

Exploraremos One-to-One (1:1) y One-to-Many (1:N).

Relación One-to-One (1:1)

En una relación uno a uno,un registro en una tabla se asocia exclusivamente con un solo registro en otra tabla, y viceversa. Es como si cada persona tuviera un único pasaporte.

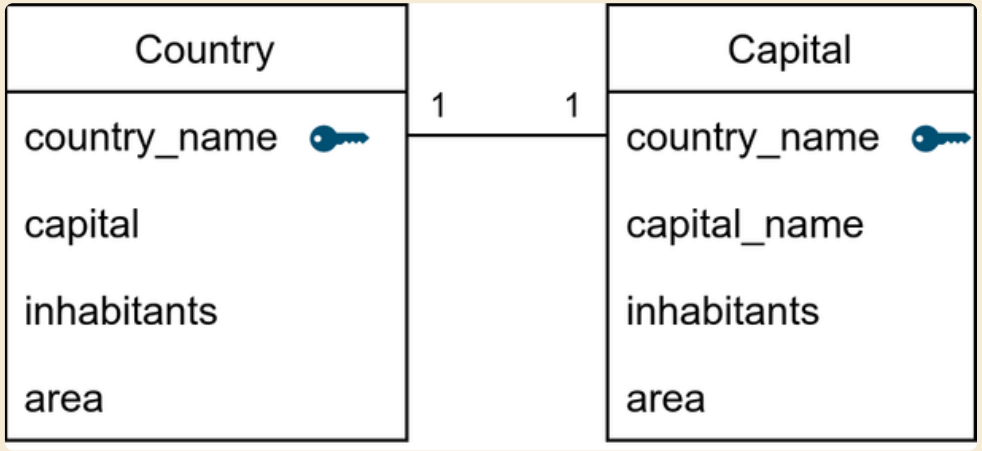
Definición

Cada registro de la Tabla A se relaciona con un único registro de la Tabla B.

Cada registro de la Tabla B se relaciona con un único registro de la Tabla A.

Implementación

Se usa una clave foránea en una de las tablas, que además debe ser **única** (UNIQUE constraint) para asegurar la exclusividad de la relación.



Ejemplo práctico

País y Capital: Un país tiene solo una capital y viceversa.

Ejemplo SQL One-to-One (1:1)

Una persona puede tener un único pasaporte, y cada pasaporte pertenece exclusivamente a una sola persona. Esta es una relación perfecta de uno a uno (1:1).

```sql
-- TABLA PRINCIPAL
CREATE TABLE personas (
    id INTEGER PRIMARY KEY,   -- Primary Key
    nombre VARCHAR(100)
);

-- TABLA RELACIONADA (1:1)
CREATE TABLE pasaportes (
    id INTEGER PRIMARY KEY,         -- Su propia Primary Key
    persona_id INTEGER UNIQUE,      -- Foreign Key + UNIQUE ← CLAVE!
    numero VARCHAR(20),
    FOREIGN KEY (persona_id) REFERENCES personas(id)
);
```

Primary Key:

Cada tabla tiene su propia PK para identificar registros únicos

Foreign Key + UNIQUE:

persona_id es la foreign key (apunta a personas.id)
UNIQUE garantiza que no se repita → solo un pasaporte por persona
Sin *UNIQUE* sería One-to-Many (una persona podría tener varios pasaportes)

Relación SQL One-to-Many (1:N)

Esta es la relación más común yflexible. Un registro en la Tabla A puede estar relacionado con múltiples registros en la Tabla B, pero un registro en la Tabla B solo puede estar relacionado con un registro de la Tabla A.

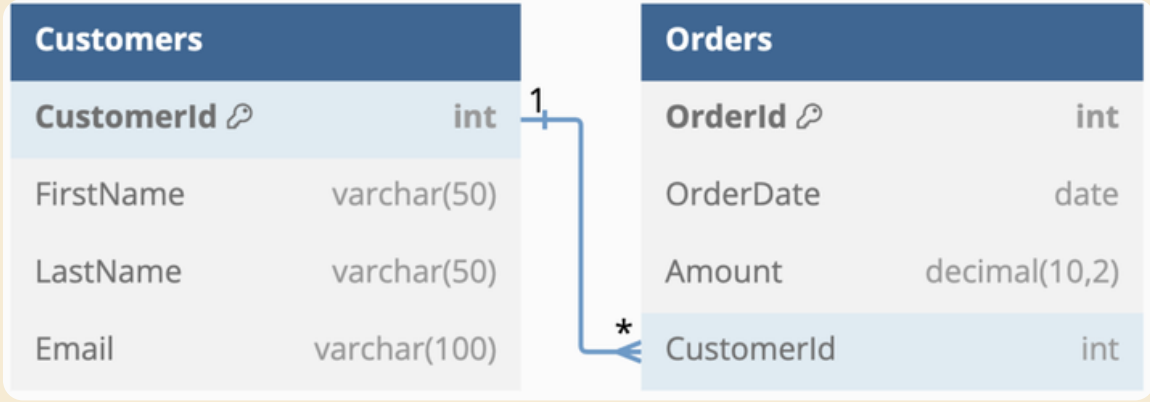
Definición

Un registro de la Tabla A se relaciona con **varios** registros de la Tabla B.

Un registro de la Tabla B se relaciona con **solo uno** de la Tabla A.

Implementación

La clave foránea se coloca en el lado "Many" de la relación, es decir, en la Tabla B.



Ejemplo práctico

Comprador y pedido: Un comprador puede hacer muchos lpedidos, pero cada pedido es comprado por una unica persona.

Ejemplo SQL One-to-Many (1:N)

Consideraunabasededatosparaunatiendaonline.Unclientepuede realizar muchos pedidos, pero cada pedido es hecho por un solo cliente.

```sql
-- Tabla principal (el "UNO")
CREATE TABLE clientes (
    id INTEGER PRIMARY KEY,
    nombre VARCHAR(100),
    email VARCHAR(100),
    telefono VARCHAR(15)
);

-- Tabla dependiente (los "MUCHOS")
CREATE TABLE pedidos (
    id INTEGER PRIMARY KEY,
    cliente_id INTEGER,                -- Foreign Key (SIN UNIQUE)
    fecha_pedido DATE,
    total DECIMAL(10,2),
    estado VARCHAR(20),
    FOREIGN KEY (cliente_id) REFERENCES clientes(id)
);
```

Aquí, el **cliente_id** en la tabla **Pedidos** permite que un mismo cliente tenga múltiples pedidos asociados, pero cada pedido está vinculado a un único cliente.

¿Qué más relaciones hay entre tablas?

Many-to-Many (N:M)

La más común después de 1:N.
Requiere una tabla intermedia:

```sql
-- Estudiantes pueden tener múltiples cursos
-- Cursos pueden tener múltiples estudiantes
CREATE TABLE estudiante_curso (
    estudiante_id INT,
    curso_id INT,
    PRIMARY KEY (estudiante_id, curso_id)
);
```

Self-Referencing (Auto-referencial)

Una tabla que se relaciona consigo misma:

```sql
-- Empleados con jefes (que también son empleados)
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    jefe_id INT,
    FOREIGN KEY (jefe_id) REFERENCES empleados(id)
);
```

¿Qué más relaciones hay entre tablas?

Hierarchical (Jerárquica)

Similar a auto-referencial pero específica para jerarquías:

```sql
-- Categorías con subcategorías
CREATE TABLE categorias (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    categoria_padre_id INT,
    FOREIGN KEY (categoria_padre_id) REFERENCES categorias(id)
);
```

Conditional/Optional Relationships

Relaciones que pueden existir o no dependiendo de condiciones:

```sql
-- Un pedido PUEDE tener un descuento aplicado
CREATE TABLE pedidos (
    id INT PRIMARY KEY,
    total DECIMAL(10,2),
    descuento_id INT NULL, -- Puede ser NULL
    FOREIGN KEY (descuento_id) REFERENCES descuentos(id)
);
```

¿Qué más relaciones hay entre tablas?

Polymorphic Relationships

Una tabla que puede relacionarse con múltiples tipos de entidades:

```sql
-- Comentarios que pueden ser para posts o para fotos
CREATE TABLE comentarios (
    id INT PRIMARY KEY,
    contenido TEXT,
    comentable_id INT,
    comentable_type VARCHAR(50), -- 'post' o 'foto'
    -- No hay FK tradicional, se maneja en aplicación
);
```

Ternary Relationships (N:M:P)

Relaciones entre tres o más entidades:

```sql
-- Estudiante + Curso + Profesor (todos relacionados)
CREATE TABLE asignaciones (
    estudiante_id INT,
    curso_id INT,
    profesor_id INT,
    semestre VARCHAR(20),
    PRIMARY KEY (estudiante_id, curso_id, profesor_id),
    FOREIGN KEY (estudiante_id) REFERENCES estudiantes(id),
    FOREIGN KEY (curso_id) REFERENCES cursos(id),
    FOREIGN KEY (profesor_id) REFERENCES profesores(id)
);
```

¿Qué más relaciones hay entre tablas?

Composition (Composición)

Relación fuerte donde las entidades hijas no pueden existir sin la padre:

```sql
-- Pedido y sus líneas de detalle
CREATE TABLE pedidos (
    id INT PRIMARY KEY,
    fecha DATE
);

CREATE TABLE detalle_pedidos (
    id INT PRIMARY KEY,
    pedido_id INT NOT NULL, -- No puede ser NULL
    producto VARCHAR(50),
    FOREIGN KEY (pedido_id) REFERENCES pedidos(id) ON DELETE CASCADE
);
```

Aggregation (Agregación)

Relación más débil donde las entidades pueden existir independientemente:

```sql
-- Departamento y empleados (empleados pueden existir sin departamento)
CREATE TABLE departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    departamento_id INT, -- Puede ser NULL
    FOREIGN KEY (departamento_id) REFERENCES departamentos(id)
);
```

Antes de la práctica... ¿Qué es JOIN?

JOIN es como un "pegamento" que une información de dos tablas diferentes,es decir, JOIN combina filas de dos o más tablas basándose en una condición de relación entre ellas.

Estructura básica de JOIN

```sql
SELECT columnas_que_quiero_ver
FROM tabla_principal
JOIN tabla_secundaria ON condición_de_unión;
```

La clave está en ON

```sql
ON l.autor_id = a.id
   ↑          ↑
   Foreign    Primary
   Key        Key
```

¿Por qué necesitamos JOIN?

Imagina que tienes esta información:

Tabla autores

id	nombre
1	García Márquez
2	Isabel Allende

Tabla libros

id	titulo	autor_id
1	Cien años de soledad	1
2	La casa de espíritus	2

¿Cómo mostramos el título del libro CON el nombre del autor?

```sql
SELECT libros.titulo, autores.nombre
FROM libros
JOIN autores ON libros.autor_id = autores.id;
```

resultado

titulo	nombre
Cien años de soledad	García Márquez
La casa de espíritus	Isabel Allende

¿Qué es LEFT/RIGHT JOIN?

Muestra TODO de la tabla izquierda /derecha
La tabla "izquierda" es la que escribes DESPUÉS de FROM y la de la “derecha” es la que escribes DESPUÉS de JOIN

```sql
SELECT a.nombre, l.titulo
FROM autores a          ← Esta es la tabla IZQUIERDA
LEFT JOIN libros l      ← Esta es la tabla DERECHA
```

¿Por qué lo necesitamos?

Si queremos ver TODOS los autores (aunque no tengan libros):

```sql
SELECT autores.nombre, libros.titulo
FROM autores
LEFT JOIN libros ON autores.id = libros.autor_id;
```

resultado

```
nombre          | titulo
García Márquez  | NULL
Isabel Allende  | La casa de espíritus
```

Aliases (Atajos)

En lugar de escribir autores.nombre, usamos:

```sql
SELECT a.nombre, l.titulo
FROM autores a
JOIN libros l ON a.id = l.autor_id;
```

a = alias para autores
l = alias para libros

GROUP BY

Agrupa registros iguales

¿Cuántos libros tiene cada autor?

```sql
SELECT autores.nombre, COUNT(libros.id) as total_libros
FROM autores
LEFT JOIN libros ON autores.id = libros.autor_id
GROUP BY autores.nombre;
```

resultado

```
nombre          | total_libros
García Márquez  | 2
Isabel Allende  | 2
```

Comandos Esenciales

CREAR RELACIÓN :

FOREIGNKEY(campo_fk) **REFERENCES** tabla_padre(campo_pk)+ agregar **UNIQUE** al campo_fk

CREAR RELACIÓN :N

FOREIGNKEY (campo_fk) **REFERENCES** tabla_padre (campo_pk) (sin UNIQUE en campo_fk)

CONSULTAS CON JOINS

INNER JOIN: Solo registrosque coinciden en ambas tablas

LEFT JOIN: Todos de la izquierda + coincidencias de la derecha

RIGHT JOIN: Todos de la derecha + coincidencias de la izquierda

OPERACIONES ÚTILES

COUNT(): Contarregistros

GROUP BY: Agrupar resultados

HAVING: Filtrar grupos

ORDER BY: Ordenar resultados

Caso práctico:

1-CREAR TABLAS

```sql
-- BIBLIOTECA DIGITAL

-- Tabla Autores (Uno)
CREATE TABLE autores (
    id INTEGER PRIMARY KEY,
    nombre VARCHAR(100),
    nacionalidad VARCHAR(50)
);

-- Tabla Libros (Muchos) - One-to-Many
CREATE TABLE libros (
    id INTEGER PRIMARY KEY,
    titulo VARCHAR(200),
    autor_id INTEGER,
    año_publicacion INTEGER,
    FOREIGN KEY (autor_id) REFERENCES autores(id)
);

-- Tabla Perfiles_Autor (Uno) - One-to-One
CREATE TABLE perfiles_autor (
    id INTEGER PRIMARY KEY,
    autor_id INTEGER UNIQUE,
    biografia TEXT,
    foto_url VARCHAR(300),
    FOREIGN KEY (autor_id) REFERENCES autores(id)
);
```

2-INSERTAR DATOS

```sql
-- Insertar autores
INSERT INTO autores VALUES
(1, 'Gabriel García Márquez', 'Colombiana'),
(2, 'Isabel Allende', 'Chilena'),
(3, 'Mario Vargas Llosa', 'Peruana');

-- Insertar libros (One-to-Many: Un autor puede tener varios libros)
INSERT INTO libros VALUES
(1, 'Cien años de soledad', 1, 1967),
(2, 'El amor en los tiempos del cólera', 1, 1985),
(3, 'La casa de los espíritus', 2, 1982),
(4, 'Eva Luna', 2, 1987),
(5, 'La ciudad y los perros', 3, 1963);

-- Insertar perfiles (One-to-One: Un autor tiene un solo perfil)
INSERT INTO perfiles_autor VALUES
(1, 1, 'Escritor colombiano, Premio Nobel de Literatura 1982', 'foto1.jpg'),
(2, 2, 'Escritora chilena, una de las más leídas del mundo', 'foto2.jpg');
```

Caso práctico:

3-Preguntas:

- ¿Qué tipo de relación existe entre autores y libros?
- ¿Qué tipo de relación existe entre autores y perfiles_autor?
- ¿En qué tabla está la foreign key en cada caso y por qué?

4-Consultas 1: n

```sql
-- 1. Mostrar todos los libros con el nombre de su autor
SELECT l.titulo, a.nombre AS autor
FROM libros l
JOIN autores a ON l.autor_id = a.id;

-- 2. ¿Cuántos libros ha escrito cada autor?
SELECT a.nombre, COUNT(l.id) AS total_libros
FROM autores a
LEFT JOIN libros l ON a.id = l.autor_id
GROUP BY a.id, a.nombre;

-- 3. Mostrar solo autores que tienen más de 1 libro
-- TU TURNO: Escribe la consulta
```

5-Consultas 1: 1

```sql
-- 1. Mostrar autores con su biografía
SELECT a.nombre, p.biografia
FROM autores a
JOIN perfiles_autor p ON a.id = p.autor_id;

-- 2. Mostrar TODOS los autores (tengan o no perfil)
-- TU TURNO: Escribe la consulta con LEFT JOIN
```

Solución Caso Práctico:

3-Preguntas:

- ¿Qué tipo de relación existe entre autores y libros? **One-to-Many (1:N)**

  Un autor puede escribir varios libros, pero cada libro tiene un solo autor principal.
  Gabriel García Márquez → "Cien años de soledad" + "El amor en tiempos del cólera"
  Un libro no puede tener múltiples autores principales (en este modelo)

- ¿Qué tipo de relación existe entre autores y perfiles_autor? **One-to-One (1:1)**

  Cada autor tiene un único perfil biográfico, y cada perfil pertenece a un solo autor.
  Gabriel García Márquez → 1 perfil con su biografía
  No puede tener varios perfiles, ni un perfil puede ser de varios autores

- ¿En qué tabla está la foreign key en cada caso y por qué?
  Foreign Key: En la tabla libros (columna autor_id)
  ¿Por qué? La FK siempre va en la tabla "muchos". Como un autor puede tener muchos libros, cada libro necesita "recordar" quién es su autor.
  Caso 2 (One-to-One):
  Foreign Key: En la tabla perfiles_autor (columna autor_id)
  ¿Por qué? En relaciones 1:1 puede ir en cualquiera, pero la pusimos en perfiles_autor porque el perfil "depende" del autor. Además tiene UNIQUE para garantizar que cada autor solo tenga un perfil.

💡 ***Regla práctica: En 1:N la FK va en "muchos", en 1:1 va en la tabla "dependiente" o menos importante.***

4-Consultas 1: n

```sql
SELECT a.nombre, COUNT(l.id) AS total_libros
FROM autores a
LEFT JOIN libros l ON a.id = l.autor_id
GROUP BY a.id, a.nombre
HAVING COUNT(l.id) > 1;
```

5-Consultas 1: 1

```sql
SELECT a.nombre, p.biografia
FROM autores a
LEFT JOIN perfiles_autor p ON a.id = p.autor_id;
```

¡Gracias por la atención!

Recursos:

- Desglose Ejercicios
- Repositorio Github