¿Por qué en una relación N:M se usan las dos FK como clave primaria compuesta?

Porque **la tabla intermedia no representa un objeto nuevo**, sino **la relación entre dos objetos**.

EJEMPLO: Alumnos y Cursos

Un alumno puede estar en muchos cursos.
Un curso puede tener muchos alumnos.
Eso es N:M.

SQL te obliga a crear una tabla intermedia:

Matricula
id_alumno (FK)
id_curso (FK)

LA IDENTIDAD DE ESA FILA ES LA COMBINACIÓN DE ESOS DOS CAMPOS

Pregúntate: ¿Qué es lo que hace que una matrícula sea única?
No es un número inventado. (id_matricula)

Es:

- **Qué alumno**
- **En qué curso**

Eso **define la matrícula**.

Por tanto:

- Esa combinación **ES la identidad real de ese registro**
- Esa combinación **es única en el mundo real**

Así que SQL dice:
Si eso ya identifica de forma única la fila… ¿para qué inventar otra clave artificial?
Y hace esto:
PRIMARY KEY (id_alumno, id_curso) PK compuesta

¿Qué pasa si creas una PK artificial como id_matricula?

id_matricula ← número inventado
id_alumno
id_curso

Pasa esto:
Puedes insertar duplicados sin darte cuenta:
1 | 14 | 3
2 | 14 | 3 ← duplicado válido para SQL

SQL NO PROHÍBE esto porque la PK es el id, no la combinación de FKs.

Es decir:

- La base de datos **no sabría** que es la misma matrícula repetida
- Tendrías que crear igualmente una restricción UNIQUE en (id_alumno, id_curso)
- Entonces tendrías **dos índices en lugar de uno** (PK y UNIQUE)

Más espacio, más lentitud, más complejidad.

Piensa en esto como una pareja:

Una tabla N:M es como una lista de parejas de baile:

(pareja = personaA + personaB)

No necesitas decir:
pareja_1784928374 = personaA + personaB

La pareja **es** la combinación.
No le hace falta un ID artificial.

Razones resumidas

1. La combinación de las dos FKs ya identifica de forma única la fila
No hace falta otra PK artificial.

2. Evita duplicados automáticamente
SQL no dejará:
(id_alumno = 14, id_curso = 3) repetido.

3. Es más eficiente
1 índice compuesto en vez de 2.

4. Es más lógico
La tabla no representa un objeto propio, sino una relación.

5. Sigue la teoría de normalización
No introduces datos que no aportan nada.

¿CUÁNDO SÍ SE USA UNA PK ARTIFICIAL?
Solo cuando la tabla N:M tiene vida propia.
Ejemplo: detalle de pedido
(Precio, cantidad, IVA, descuentos…)

En ese caso:

- La relación tiene atributos propios importantes
- Cada registro SÍ tiene identidad real
- Conviene tener un id_detalle

Si la tabla solo relaciona dos cosas → PK compuesta
Si la tabla tiene datos propios → PK artificial

1) EJEMPLO CON PK COMPUESTA (las dos FK son la PK)

Tenemos:

```
CREATE TABLE alumno (
   id_alumno INT PRIMARY KEY,
   nombre    VARCHAR(50)
);

CREATE TABLE curso (
   id_curso INT PRIMARY KEY,
   nombre   VARCHAR(50)
);
```

Tabla intermedia **solo relación N:M**, sin datos propios:

```
CREATE TABLE matricula_compuesta (
   id_alumno INT,
   id_curso  INT,
   fecha     DATE,
   PRIMARY KEY (id_alumno, id_curso),      -- PK compuesta
   FOREIGN KEY (id_alumno) REFERENCES alumno(id_alumno),
   FOREIGN KEY (id_curso)  REFERENCES curso(id_curso)
);
```

Inserciones

```
INSERT INTO matricula_compuesta (id_alumno, id_curso, fecha)
VALUES (1, 10, '2024-09-01');   -- OK
```

Si intentas repetir la misma combinación:

```
INSERT INTO matricula_compuesta (id_alumno, id_curso, fecha)
VALUES (1, 10, '2024-09-15');   -- ERROR: viola la PK (duplicado)
```

La BD no permite que el mismo alumno esté dos veces en el mismo curso.
La combinación (id_alumno, id_curso) **ya es la identidad**.

2) EJEMPLO CON PK ARTIFICIAL (id propio) Y SIN CONTROL

Mismas tablas alumno y curso, pero otra tabla intermedia:

```
CREATE TABLE matricula_artificial_mal (
   id_matricula INT PRIMARY KEY,       -- PK artificial
   id_alumno    INT,
   id_curso     INT,
   fecha        DATE,
   FOREIGN KEY (id_alumno) REFERENCES alumno(id_alumno),
   FOREIGN KEY (id_curso)  REFERENCES curso(id_curso)
);
```

Inserciones

```
INSERT INTO matricula_artificial_mal (id_matricula, id_alumno, id_curso, fecha)
VALUES (1, 1, 10, '2024-09-01');   -- OK

INSERT INTO matricula_artificial_mal (id_matricula, id_alumno, id_curso, fecha)
VALUES (2, 1, 10, '2024-09-15');   -- también OK
```

Aquí **no hay ningún error** aunque el alumno 1 esté dos veces en el curso 10.

La PK es solo id_matricula, así que la BD **no ve el duplicado lógico**.

Para corregirlo, tendrías que hacer:

```
ALTER TABLE matricula_artificial_mal
ADD CONSTRAINT uq_alumno_curso UNIQUE (id_alumno, id_curso);
```

Al final tienes:

- Una **PK artificial** (id_matricula)
- Y además un **índice UNIQUE** sobre (id_alumno, id_curso)

Más columnas, más índices, más complejidad… para acabar como el caso 1.

3) CUÁNDO SÍ TIENE SENTIDO LA PK ARTIFICIAL

Si la tabla tiene **datos propios importantes**, por ejemplo:

```
CREATE TABLE detalle_pedido (
   id_detalle  INT PRIMARY KEY,       -- aquí sí tiene sentido
   id_pedido   INT,
   id_producto INT,
   cantidad    INT,
   precio_unit NUMERIC(10,2),
   FOREIGN KEY (id_pedido)   REFERENCES pedido(id_pedido),
   FOREIGN KEY (id_producto) REFERENCES producto(id_producto)
```

);

Aquí cada fila de detalle_pedido **sí es una entidad en sí misma**
(cantidad, precio, descuentos, etc.), por eso el id_detalle tiene lógica.

EJERCICIO: ¿PK COMPUESTA o PK ARTIFICIAL?
Di si cada tabla intermedia necesita:

- **PKC → Clave primaria compuesta (las dos FK)**
- **PKA → Clave primaria artificial (un id propio**

1. Alumno – Curso (Matricula)
Campos: id_alumno, id_curso, fecha_alta

Respuesta: PKC (clave primaria compuesta: id_alumno, id_curso)
Por qué:

- La tabla *Matricula* solo representa **que un alumno está en un curso**.
- La identidad real de la fila es: este alumno en este curso.
- No debería haber dos matrículas del mismo alumno al mismo curso (sería duplicado lógico).
- Usar PRIMARY KEY (id_alumno, id_curso):
  - Evita duplicados automáticamente
  - No necesitas un id_matricula inventado

Caso típico de **tabla puramente relacional N:M** → PK compuesta.

2. Actor – Película (Reparto)
Campos: id_actor, id_pelicula, rol

Respuesta (caso estándar): PKC (id_actor, id_pelicula)

Por qué:

- Normalmente un actor aparece **una sola vez** en una película en el reparto (aunque haga varios papeles, muchas veces se representan como uno solo).
- La identidad lógica es: este actor participa en esta película
- Si rol es solo información extra (protagonista, secundario…) y no esperas **varias filas** para el mismo actor y película,
  → PRIMARY KEY (id_actor, id_pelicula) es suficiente.

Si quisieras guardar **varios roles distintos** del mismo actor en la misma película (ej: doblajes, personajes múltiples), entonces sí podría tener sentido un **id artificial** o añadir rol a la PK. Pero para modelo típico: **PK compuesta**.

3. Pedido – Producto (DetallePedido)
Campos: id_pedido, id_producto, cantidad, precio_unit, descuento

Respuesta: PKA (clave primaria artificial, por ejemplo id_detalle)
Por qué:
Aquí cambia el juego:

- Cada fila de DetallePedido **sí tiene identidad propia**:
  - Es una "línea de pedido" con cantidad, precio, descuento, impuestos, etc.
- No es solo "producto en pedido", sino **una línea de venta con atributos importantes**.
- Además:
  - A veces se puede repetir el mismo producto en el mismo pedido con condiciones distintas (descuento, packs, promociones...).
  - Una PK compuesta (id_pedido, id_producto) **bloquearía eso**.
- Con id_detalle como PK:
  - Puedes identificar **unívocamente** cada línea
  - Puedes referenciar esta línea desde otras tablas (devoluciones, albaranes, facturas…)

Aquí lo correcto en la mayoría de diseños reales: **PK artificial**.

4. Usuario – Rol (UsuarioRol)
Campos: id_usuario, id_rol

Respuesta: PKC (id_usuario, id_rol)
Por qué:

- Tabla puramente relacional: este usuario tiene este rol
- No tiene sentido que el mismo usuario tenga el mismo rol dos veces.
- La combinación:
  - id_usuario = 5
  - id_rol = ADMIN
    es la identidad real de la fila.
- PRIMARY KEY (id_usuario, id_rol):
  - Evita duplicados
  - Mantiene el modelo limpio
  - No necesitas un id artificial id_usuario_rol.

Es otro caso claro de **relación N:M pura** → PK compuesta.

5. Profesor – Departamento (Asignación)
Campos: id_profesor, id_departamento, fecha_inicio, fecha_fin

Respuesta (modelo serio con histórico): PKA (clave artificial, por ejemplo id_asignacion)
Por qué:

- Aquí casi seguro quieres guardar **histórico de asignaciones**:
  - Un profesor puede estar en el mismo departamento **varias veces** en su vida laboral (se va, vuelve, cambia, etc.).

- Eso significa que podría haber registros como:

id_profesor	id_departamento	fecha_inicio	fecha_fin
7	3	2023-09-01	2024-06-30
7	3	2025-09-01	NULL

- Si usaras PRIMARY KEY (id_profesor, id_departamento):
    - No podrías insertar la segunda fila (el sistema la vería duplicada).

- Además, la unidad significativa aquí es: una asignación concreta en un intervalo de tiempo.

- Eso **sí es una entidad propia**, por lo que tiene todo el sentido:

id_asignacion (PK)
id_profesor (FK)
id_departamento (FK)
fecha_inicio
fecha_fin

En tablas de relación **con histórico o múltiples períodos**, lo normal es **PK artificial**.

Resumen

- **Solo enlaza dos cosas y no tiene vida propia → PK compuesta**
    - Matricula (Alumno–Curso)
    - UsuarioRol (Usuario–Rol)
    - Reparto simple (Actor–Película)
- **Tiene datos propios importantes o puede repetirse con variaciones → PK artificial**
    - DetallePedido
    - Asignación Profesor–Departamento con fechas
    - Cualquier relación con histórico, precio, cantidad, estado, etc.