MySQL

GROUP BY & Cardinalidad

Clausulas GROUP BY | SQL

En ciertas consultas nos puede interesar agrupar elementos, registros de una tabla. Por ejemplo, de nuestra tabla “articulos” nos podría interesar agrupar los registros por “marca”, o podríamos agruparlos por “empleado”. Para ello usaremos la clausula GROUP BY junto a COUNT:

```
SELECT COUNT(ID), marca FROM articulos GROUP BY marca;
```

Esta orden se utiliza para contar cuántos registros hay para cada marca en la tabla "articulos".

SELECT COUNT(ID), marca: Estamos seleccionando dos columnas, COUNT(ID) y marca.
COUNT(ID): cuenta el número de registros para cada marca y la columna "marca" nos muestra la marca correspondiente.
FROM articulos: Indicamos que estamos seleccionando datos de la tabla "articulos".
GROUP BY marca: Especificamos que queremos agrupar los registros según el valor de la columna "marca".

Clausulas GROUP BY | SQL

También podemos usar una clausula JOIN para, por ejemplo, agrupar cuantos productos tiene cada vendedor:

```
SELECT e.Nombre AS Comercial, COUNT(p.ID) AS Cantidad_Productos
FROM empleados e
LEFT JOIN articulos p ON e.ID = p.Vendedor
GROUP BY e.Nombre;
```

En esta consulta, estamos utilizando **LEFT JOIN** para unir la tabla de empleados con la tabla de productos. Esto nos permite incluir todos los empleados, incluso aquellos que no tienen productos asociados.
Luego, utilizamos la función de agregación **COUNT** para contar la cantidad de productos por vendedor. Usamos la columna **ID** de la tabla de productos para contar los registros.
Finalmente, utilizamos la cláusula **GROUP BY** para agrupar los resultados por el nombre del vendedor. Esto nos dará un resultado que muestra el nombre de cada vendedor junto con la cantidad de productos que tiene asignados. Los vendedores sin productos tendrán un valor de 0 en la columna de cantidad de productos.

Clausulas GROUP BY | SQL

Así mismo, puedo limitar las consultas, según diferentes criterios:

```
SELECT e.Nombre AS Comercial, COUNT(p.ID) AS Cantidad_Productos
FROM empleados e
LEFT JOIN articulos p ON e.ID = p.Vendedor
GROUP BY e.Nombre
HAVING COUNT(e.Nombre) >= 2;
```

En esta consulta, estamos utilizando la cláusula **HAVING** para filtrar los resultados. En este caso, estamos seleccionando solo aquellos vendedores que tengan una cantidad de productos igual o mayor a 2. Esto nos permite mostrar solo los vendedores que tienen al menos 2 productos asociados.

Clausulas GROUP BY | SQL

La clausula **COUNT** se puede usar de las siguientes formas:

Contar el número total de registros en una tabla:

```
SELECT COUNT(*) FROM articulos;
```

Contar el número de registros distintos en una columna:

```
SELECT COUNT(DISTINCT Vendedor) FROM articulos;
```

Contar el número de registros agrupados por una columna:

```
SELECT marca, COUNT(*) FROM articulos GROUP BY marca;
```

Clausulas GROUP BY | SQL

Por último, veremos la última instrucción que consideramos puede serte útil antes de pasar al moldeamiento de bases de datos:

```
DROP TABLE articulos;
DROP TABLE empleados;
DROP TABLE trabajadores;
DROP TABLE clientes;
DROP TABLE proveedores;
```

Evidentemente, el uso de **DROP TABLE** es la de borrar tablas completas.

Cardinalidad| SQL

La **cardinalidad** en el contexto de las bases de datos se refiere a la relación que existe entre dos tablas en términos de la cantidad de registros que pueden estar relacionados entre sí. Se utiliza para describir la cantidad de ocurrencias en una relación específica entre dos conjuntos de datos.

Existen tres tipos principales de **cardinalidad**:

1. Cardinalidad Uno a Uno (1↔1)
2. Cardinalidad Uno a Muchos (1 ↔ n)
3. Cardinalidad Muchos a Muchos (N ↔ n)

Cardinalidad| SQL

Existen tres tipos principales de cardinalidad:

1. **Cardinalidad Uno a Uno (1 ↔ 1):** En esta relación, un registro de una tabla está asociado con exactamente un registro de otra tabla, y viceversa. Es decir, cada registro en una tabla tiene una única correspondencia en la otra tabla.
2. **Cardinalidad Uno a Muchos (1 ↔ n):** En esta relación, un registro de una tabla puede estar asociado con varios registros de otra tabla, pero cada registro de la segunda tabla solo puede estar asociado con un único registro de la primera tabla. Es la relación más común en las bases de datos, donde un registro en una tabla principal se relaciona con varios registros en una tabla secundaria.
3. **Cardinalidad Muchos a Muchos (N ↔ n):** En esta relación, varios registros de una tabla pueden estar asociados con varios registros de otra tabla. Para representar esta relación en una base de datos relacional, se utiliza una tabla intermedia o de unión que vincula los registros de ambas tablas.

Cardinalidad| SQL

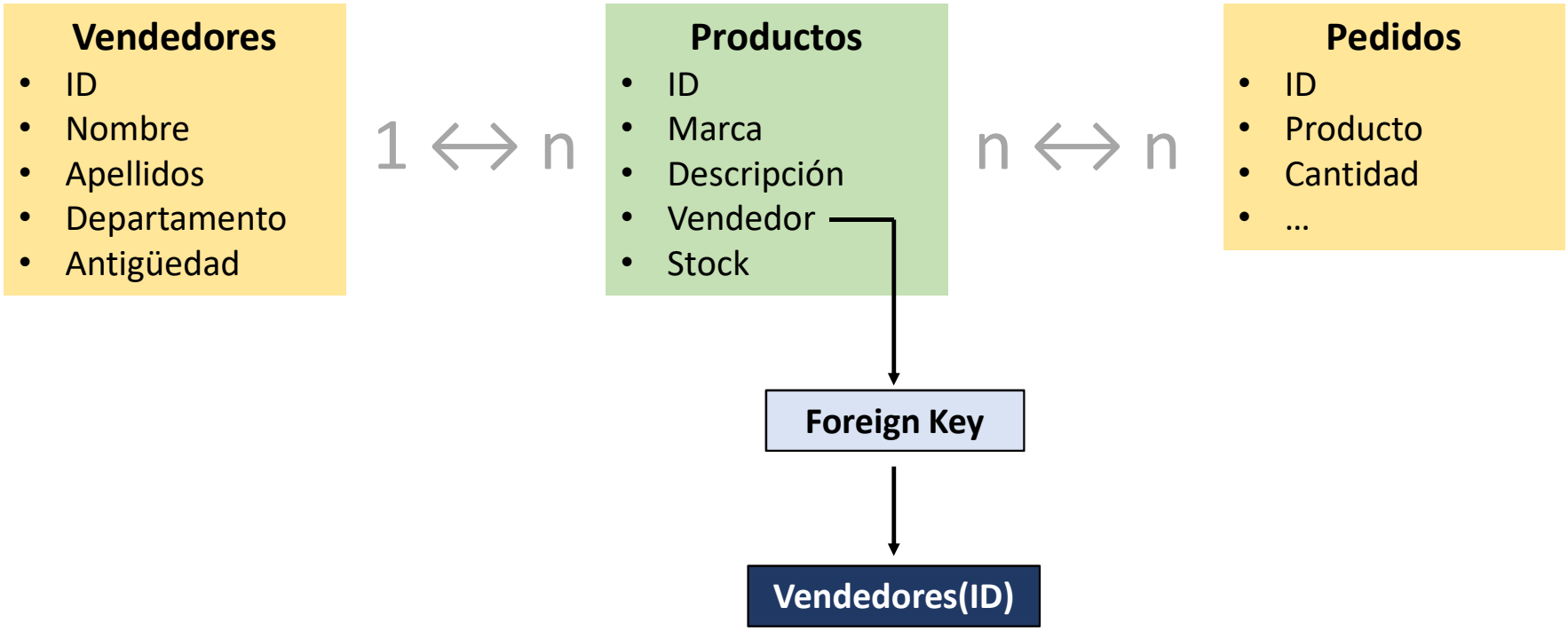
La **cardinalidad** es importante en el diseño de bases de datos ya que ayuda a definir las relaciones entre las tablas y determinar cómo se pueden unir los datos.

También influye en la eficiencia y el rendimiento de las consultas, ya que una mala gestión de la **cardinalidad** puede llevar a problemas como duplicidad de datos o consultas lentas.

Por lo tanto, comprender y definir correctamente la **cardinalidad** es fundamental para diseñar una estructura de base de datos eficiente y coherente.

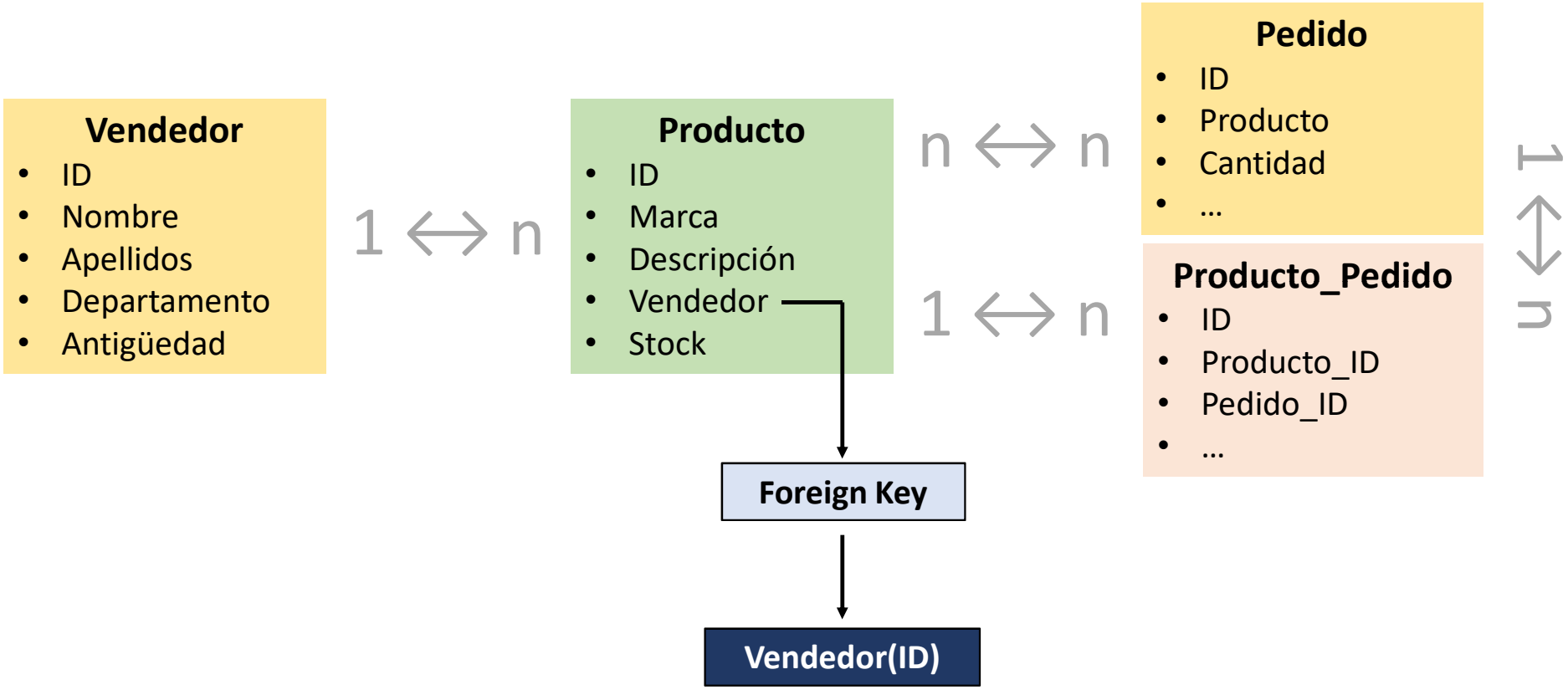
Cardinalidad| SQL

Veamos un ejemplo de la cardinalidad entre varias tablas en una base de datos.



Cardinalidad| SQL

¿Cómo resolvemos una cardinalidad de n↔n? Con una tabla intermedia. Así:



Cardinalidad| SQL

Es decir, al encontrarnos con una relación n↔n entre dos tablas, deberemos crear una tabla intermedia, que tendrá una relación, una cardinalidad con ambas tablas de 1↔n.

Ahora veamos como hacer esto en MySQL…