

Los operadores de agregación en álgebra relacional, como **SUM**, **COUNT**, **AVG**, **MAX**, y **MIN**, permiten realizar cálculos sobre conjuntos de valores en una base de datos relacional. Estos operadores no son fundamentales en álgebra relacional clásica, pero se derivan y se utilizan ampliamente en sistemas de bases de datos para resumir y analizar datos.

A continuación, te explico cada uno de estos operadores y cómo se derivan, junto con un ejemplo de cómo se utilizan en SQL.

1. SUM

El operador **SUM** calcula la suma total de un conjunto de valores numéricos en una columna.

- **Definición:**

$$SUM(A) = \sum_{i=1}^n A_i$$

Donde A es la columna de la cual se suman los valores.

- **Ejemplo en SQL:**

```
SELECT SUM(salario) FROM empleados;
```

Esto calcula la suma total de los salarios de todos los empleados.

2. COUNT

El operador **COUNT** cuenta el número total de tuplas en una relación o el número de valores no nulos en una columna específica.

- **Definición:**

COUNT(*)=Número total de tuplas

COUNT(A)=Número de valores no nulos en la columna A

- **Ejemplo en SQL:**

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

Esto cuenta el número total de empleados en la tabla.

3. AVG

El operador **AVG** calcula el valor promedio de un conjunto de valores numéricos en una columna.

- **Definición:**

$$AVG(A) = \frac{\sum_{i=1}^n A_i}{n}$$

Donde A es la columna de la cual se calcula el promedio y n es el número de valores en A.

- **Ejemplo en SQL:**

```
SELECT AVG(salario) FROM empleados;
```

Esto calcula el salario promedio de todos los empleados.

4. MAX

El operador **MAX** retorna el valor máximo en una columna.

- **Definición:**

$$MAX(A) = \max_{i=1}^n A_i$$

- **Ejemplo en SQL:**

```
SELECT MAX(salario) FROM empleados;
```

Esto retorna el salario máximo entre todos los empleados.

5. MIN

El operador **MIN** retorna el valor mínimo en una columna.

- **Definición:**

$$MIN(A) = \min_{i=1}^n A_i$$

- **Ejemplo en SQL:**

```
SELECT MIN(salario) FROM empleados;
```

Esto retorna el salario mínimo entre todos los empleados.

Derivación y Uso:

Estos operadores de agregación no son operadores básicos en álgebra relacional, pero se pueden ver como funciones derivadas que aplican un cálculo a un conjunto de valores extraídos de las tuplas que cumplen con ciertas condiciones en la consulta.

En términos de álgebra relacional:

- **SUM, AVG, MAX, MIN** se aplican a una sola columna de una relación.
- **COUNT** puede aplicarse a cualquier conjunto de columnas o a la relación completa.

Propiedades:

1. **SUM:**
 - **Asociativa:** SUM(A) sobre diferentes grupos puede sumarse para obtener un total.
 - **Conmutativa:** El orden de los elementos no afecta el resultado.
2. **COUNT:**
 - **Conmutativa:** El orden no afecta el conteo.
 - **Asociativa:** Puede contar en diferentes niveles y sumarse.
3. **AVG:**
 - **Asociativa** solo cuando se tienen en cuenta los pesos o los tamaños de los subconjuntos.
4. **MAX/MIN:**
 - **Idempotente:** Aplicar MAX o MIN repetidamente no cambia el resultado.
 - **Conmutativa y Asociativa:** El orden no importa.

Estas operaciones son cruciales para análisis y reportes en bases de datos, permitiendo a los usuarios obtener resúmenes de datos clave de manera rápida y eficiente.

Los operadores de totales, también conocidos como operadores de agregación, son operaciones utilizadas para realizar cálculos de resumen en conjuntos de datos. Estos operadores son fundamentales para calcular estadísticas sobre conjuntos de datos y obtener información resumida. Algunos de los operadores de totales más comunes son:

1. **SUM (Suma):** Este operador calcula la suma de los valores en una columna específica de un conjunto de datos.
2. **AVG (Promedio):** Calcula el promedio de los valores en una columna específica.
3. **COUNT (Conteo):** Cuenta el número total de tuplas en un conjunto de datos o el número de tuplas que satisfacen ciertas condiciones.
4. **MIN (Mínimo):** Encuentra el valor mínimo en una columna específica.
5. **MAX (Máximo):** Encuentra el valor máximo en una columna específica.

Estos operadores de totales son ampliamente utilizados en consultas de bases de datos para obtener información resumida sobre conjuntos de datos. Por ejemplo, pueden ser utilizados para calcular el total de ventas de una empresa, el promedio de edad de los clientes, el número de productos en inventario, entre otros. Los resultados de estos operadores proporcionan una visión general y estadísticas importantes sobre los datos almacenados en una base de datos relacional.

Operadores de Agregación con Nomenclatura Simplificada

Ejemplos Prácticos

Supongamos que tenemos la siguiente relación VENTAS:

VENTAS

| sale_id | ITEM | CANTIDAD | PRECIO |
|---------|------|----------|--------|
| 1 | 101 | 2 | 10 |
| 2 | 102 | 1 | 20 |
| 3 | 101 | 3 | 10 |
| 4 | 103 | 5 | 15 |

1. SUM: Suma de las cantidades vendidas por producto

γ ITEM,SUM(CANTIDAD)(VENTAS)

Resultado:

ITEM SUM(CANTIDAD)

101 5

102 1

103 5

2. COUNT: Contar el número de ventas por producto

γ ITEM,COUNT(*) (VENTAS)

Resultado:

ITEM COUNT(*)

101 2

ITEM COUNT(*)

| | |
|-----|---|
| 102 | 1 |
| 103 | 1 |

3. AVG: Promedio del precio de venta por producto

γ ITEM,AVG(PRECIO)(VENTAS)

Resultado:

ITEM AVG(PRECIO)

| | |
|-----|----|
| 101 | 10 |
| 102 | 20 |
| 103 | 15 |

4. MAX: Precio máximo de los productos

γ ITEM,MAX(PRECIO)(VENTAS)

Resultado:

ITEM MAX(PRECIO)

| | |
|-----|----|
| 101 | 10 |
| 102 | 20 |
| 103 | 15 |

5. MIN: Precio mínimo de los productos

γ ITEM,MIN(PRECIO)(VENTAS)

Resultado:

ITEM MIN(PRECIO)

| | |
|-----|----|
| 101 | 10 |
| 102 | 20 |
| 103 | 15 |

Ejercicio 1: Operaciones Básicas (Selección, Proyección)

Considera la siguiente relación ALUMNOS:

| id_alumno | nombre | edad | carrera |
|-----------|--------|------|----------------|
| 1 | Juan | 20 | Informática |
| 2 | María | 22 | Administración |
| 3 | Luis | 21 | Informática |
| 4 | Ana | 23 | Ingeniería |
| 5 | Pedro | 22 | Administración |

1. **Selección:** Selecciona los alumnos que estudian "Informática".

$\sigma_{\text{carrera}='Informática'}(\text{ALUMNOS})$

2. **Proyección:** Obtén una lista de los nombres y edades de todos los alumnos.

$\pi_{\text{nombre,edad}}(\text{ALUMNOS})$

Ejercicio 2: Unión, Intersección y Diferencia

Considera las siguientes dos relaciones EMPLEADOS_TECNOLOGIA y EMPLEADOS_VENTAS:

EMPLEADOS_TECNOLOGIA:

| id_empleado | nombre | salario |
|-------------|--------|---------|
| 1 | Carlos | 3000 |
| 2 | Laura | 3500 |
| 3 | José | 3200 |

EMPLEADOS_VENTAS:

| id_empleado | nombre | salario |
|-------------|--------|---------|
| 3 | José | 3200 |
| 4 | Ana | 2800 |
| 5 | María | 2700 |

1. **Unión:** Encuentra todos los empleados, tanto de Tecnología como de Ventas, sin duplicados.

$\text{EMPLEADOS_TECNOLOGIA} \cup \text{EMPLEADOS_VENTAS}$

2. **Intersección:** Encuentra los empleados que trabajan tanto en Tecnología como en Ventas.

$EMPLEADOS_TECNOLOGIA \cap EMPLEADOS_VENTAS$

3. **Diferencia:** Encuentra los empleados que trabajan solo en Tecnología y no en Ventas.

$EMPLEADOS_TECNOLOGIA - EMPLEADOS_VENTAS$

Ejercicio 3: Joins

Considera las siguientes relaciones PEDIDOS y CLIENTES:

PEDIDOS:

| id_pedido | id_cliente | fecha |
|-----------|------------|------------|
| 1 | 101 | 2023-01-05 |
| 2 | 102 | 2023-01-07 |
| 3 | 101 | 2023-01-10 |

CLIENTES:

| id_cliente | nombre | ciudad |
|------------|------------|-----------|
| 101 | Juan Pérez | Madrid |
| 102 | Ana García | Barcelona |

1. **Join Natural:** Realiza una combinación natural entre las tablas PEDIDOS y CLIENTES para obtener información completa de los pedidos realizados.

$PEDIDOS \bowtie CLIENTES$

2. **Join Theta:** Realiza una combinación theta entre PEDIDOS y CLIENTES que muestre los pedidos hechos por clientes que viven en "Madrid".

$PEDIDOS \bowtie_{ciudad='Madrid'} CLIENTES$

Ejercicio 4: Agregación y Agrupación

Considera la relación VENTAS:

| id_venta | id_producto | cantidad | precio |
|----------|-------------|----------|--------|
| 1 | 101 | 5 | 20 |
| 2 | 102 | 2 | 30 |
| 3 | 101 | 3 | 20 |

id_venta id_producto cantidad precio

4 103 1 50

1. **SUM:** Suma la cantidad total vendida por producto.

$\gamma_{id_producto, SUM(cantidad)}(VENTAS)$

2. **AVG:** Calcula el precio promedio de venta por producto.

$\gamma_{id_producto, AVG(precio)}(VENTAS)$

Ejercicio 5: División

Considera las siguientes relaciones EMPLEADOS y PROYECTOS:

EMPLEADOS:

id_empleado nombre

1 Pedro
2 Juan
3 Ana

PROYECTOS:

id_empleado id_proyecto

1 A
1 B
2 A
3 A
3 B

1. **División:** Encuentra los empleados que han trabajado en todos los proyectos.

$EMPLEADOS \div \pi_{id_proyecto}(PROYECTOS)EMPLEADOS$

Ejercicio Combinado: Sistema de Gestión de Biblioteca

Imagina que gestionas un sistema de base de datos para una biblioteca. La biblioteca tiene varios libros, autores, préstamos y clientes. Se desea realizar consultas sobre la información almacenada en el sistema.

Tablas:

1. **LIBROS:**

| id_libro | título | id_autor | año_publicación |
|----------|------------------------|----------|-----------------|
| 1 | El Quijote | 1 | 1605 |
| 2 | Cien años de soledad | 2 | 1967 |
| 3 | La ciudad y los perros | 3 | 1963 |
| 4 | Rayuela | 4 | 1963 |

2. **AUTORES:**

| id_autor | nombre_autor |
|----------|------------------------|
| 1 | Miguel de Cervantes |
| 2 | Gabriel García Márquez |
| 3 | Mario Vargas Llosa |
| 4 | Julio Cortázar |

3. **PRÉSTAMOS:**

| id_préstamo | id_libro | id_cliente | fecha_préstamo |
|-------------|----------|------------|----------------|
| 1 | 1 | 101 | 2023-08-01 |
| 2 | 2 | 102 | 2023-08-02 |
| 3 | 3 | 103 | 2023-08-05 |
| 4 | 2 | 101 | 2023-08-10 |

4. **CLIENTES:**

| id_cliente | nombre_cliente | dirección |
|------------|----------------|-----------|
| 101 | Ana Pérez | Madrid |
| 102 | Juan García | Barcelona |
| 103 | Luis Fernández | Valencia |

Preguntas a Resolver:

1. **Selección:** Encuentra los libros que fueron publicados en el año 1963.

$\sigma_{\text{año_publicación}=1963}(\text{LIBROS})$

2. **Proyección:** Muestra solo los títulos de todos los libros disponibles.

$\pi_{\text{título}}(\text{LIBROS})$

3. **Join (Combinación Natural):** Encuentra los préstamos de libros realizados por "Ana Pérez", mostrando el título del libro y la fecha del préstamo.

$\pi_{\text{título}, \text{fecha_préstamo}}(\text{PRESTAMOS} \bowtie \text{LIBROS} \bowtie \text{CLIENTES})$

Condición: CLIENTES.nombre_cliente = 'Ana Pérez'

4. **Unión:** Obtén la unión de los libros que han sido prestados y aquellos que no lo han sido. Aquí debes combinar los préstamos con los libros en un listado.

$(\pi_{\text{título}}(\text{PRESTAMOS} \bowtie \text{LIBROS})) \cup (\pi_{\text{título}}(\text{LIBROS})) -$

$\pi_{\text{título}}(\text{PRESTAMOS} \bowtie \text{LIBROS}))$

5. **Intersección:** Encuentra los autores cuyos libros han sido prestados a más de un cliente.

$\pi_{\text{nombre_autor}}(\text{LIBROS} \bowtie \text{PRESTAMOS} \bowtie \text{CLIENTES}) \cap$

$\pi_{\text{nombre_autor}}(\text{LIBROS} \bowtie \text{PRESTAMOS})$

6. **Diferencia:** Encuentra los libros que aún no han sido prestados.

$\pi_{\text{título}}(\text{LIBROS}) - \pi_{\text{título}}(\text{PRESTAMOS} \bowtie \text{LIBROS})$

7. **Agregación (SUM):** Calcula cuántos préstamos se han realizado para cada libro.

$\gamma_{\text{id_libro}, \text{COUNT}(*)(\text{PRESTAMOS})}$

8. **División:** Encuentra los clientes que han pedido prestados todos los libros de la biblioteca.

$\text{CLIENTES} \div \pi_{\text{id_libro}}(\text{LIBROS})\text{CLIENTES}$

Las **entidades débiles** son un tipo de entidad en los modelos de bases de datos que dependen de otra entidad (llamada **entidad fuerte** o **entidad propietaria**) para existir. A diferencia de las entidades fuertes, las entidades débiles no pueden ser identificadas únicamente por sus atributos propios; necesitan una clave parcial combinada con una clave de la entidad fuerte a la que están asociadas.

Características principales de las entidades débiles:

1. **Dependencia de una entidad fuerte:** Una entidad débil no puede existir sin estar asociada a una entidad fuerte. La existencia de una entidad débil depende de una relación con la entidad fuerte.
2. **Clave primaria derivada:** Las entidades débiles no tienen una clave primaria propia. Para ser identificadas de manera única, dependen de una **clave foránea** de la entidad fuerte y una **clave parcial** que distingue entre las múltiples ocurrencias de la entidad débil dentro de la misma entidad fuerte.
3. **Relación identificativa:** La relación entre una entidad fuerte y una entidad débil es una relación de tipo identificativa (denotada usualmente como una relación 1 o uno a muchos). Esta relación es crucial para la existencia de la entidad débil.
4. **No tienen suficiente significado por sí mismas:** La entidad débil no tiene un sentido claro sin la entidad fuerte que la soporta.

Ejemplo de entidad débil:

Un ejemplo clásico es el de una **Factura** (entidad fuerte) y sus **Líneas de Factura** (entidad débil). Una línea de factura no tiene sentido sin la factura a la que pertenece, y la clave de la línea de factura podría estar formada por el número de factura (clave primaria de Factura) y un número de línea de factura (clave parcial).

Diagrama Entidad-Relación (ER) de una entidad débil:

- **Factura ----- Tiene (1:N) ----- Línea de Factura**
 - Factura: id_factura (clave primaria).
 - Línea de Factura: número de línea (clave parcial) + id_factura (clave foránea).

Uso de entidades débiles:

Las entidades débiles son útiles cuando es necesario modelar situaciones en las que los datos dependen de otra entidad para su existencia y no tienen identidad por sí mismos fuera de ese contexto.

Tipos de entidades débiles:

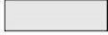
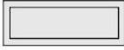





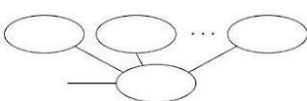

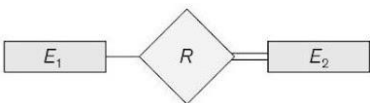
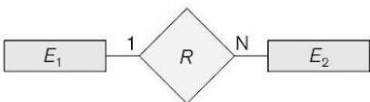
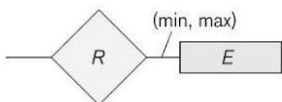
- **Entidades débiles identificativas:** Estas entidades dependen directamente de la entidad fuerte para su identificación, como el ejemplo de líneas de factura.

- **Entidades débiles de especialización:** Se utilizan en modelos de herencia cuando una subentidad (especialización) depende completamente de una entidad genérica.

Diagramas de entidad-relación

El modelo entidad-relación es un modelo conceptual mediante el cual se puede concebir la estructura de la base de datos, mantener la relación entre los diferentes componentes e identificar las restricciones que se dan en el diseño integrado del sistema de base de datos. El modelo entidad-relación o modelo ER describe los datos como entidades, atributos y relaciones.

Una entidad es un objeto del mundo real con una existencia independiente; es una cosa o un objeto con su propia identidad. Puede ser un objeto con una existencia física como una casa, una persona, un empleado, etc., o puede ser un objeto con una existencia conceptual como un curso, una carrera, un trabajo, etc.

| Symbol | Meaning |
|---|---|
|  | Entity |
|  | Weak Entity |
|  | Relationship |
|  | Identifying Relationship |
|  | Attribute |
|  | Key Attribute |
|  | Multivalued Attribute |
|  | Composite Attribute |
|  | Derived Attribute |
|  | Total Participation of E_2 in R |
|  | Cardinality Ratio 1: N for $E_1:E_2$ in R |
|  | Structural Constraint (min, max) on Participation of E in R |

Fuente: 1 Fundamentos de Sistemas de Bases de datos - Elmasri-Navathe - 5ª Ed 2007 Pearson

En el contexto de bases de datos y modelado de datos, los conceptos de **subclases**, **superclases** y **herencia** son útiles para representar relaciones jerárquicas y modelar estructuras más complejas. Estos conceptos son análogos a los que se encuentran en programación orientada a objetos, pero aplicados al diseño de bases de datos, especialmente en el **modelo entidad-relación extendido (EER)**.

1. Superclase

Una **superclase** es una entidad que contiene atributos y relaciones comunes a varias subentidades o subclases. La superclase generaliza las características que pueden ser compartidas por varias subclases. Es la entidad más "general" de una jerarquía.

Por ejemplo, si tienes una entidad **Persona**, esta puede ser una **superclase** de entidades como **Empleado** y **Cliente**. La entidad **Persona** contiene los atributos generales como nombre, dirección, y fecha de nacimiento, que son comunes a ambos tipos de personas, mientras que **Empleado** y **Cliente** pueden tener atributos específicos adicionales.

2. Subclase

Una **subclase** es una entidad que hereda los atributos y relaciones de una superclase, pero también puede tener atributos y relaciones específicas que no se aplican a todas las entidades de la superclase. Es la entidad más "específica" en la jerarquía.

En el ejemplo anterior, **Empleado** y **Cliente** son **subclases** de la entidad **Persona**. **Empleado** puede tener atributos específicos como **puesto** o **salario**, mientras que **Cliente** podría tener un atributo como **historial de compras**.

3. Herencia

La **herencia** es el mecanismo por el cual una subclase hereda atributos y relaciones de una superclase. Permite que una subclase tome las características de su superclase sin necesidad de redefinirlas.

En el diseño de bases de datos, esto implica que una subclase hereda automáticamente los atributos de la superclase, además de agregar atributos y relaciones que son específicos para esa subclase. La herencia ayuda a evitar redundancias y a mantener un diseño más organizado.

Ejemplo gráfico:

Si tenemos una base de datos para una empresa, podríamos tener la siguiente jerarquía:

- **Superclase: Persona**
 - Atributos: nombre, dirección, fecha de nacimiento
- **Subclase: Empleado**
 - Atributos adicionales: puesto, salario
- **Subclase: Cliente**
 - Atributos adicionales: historial de compras, número de cliente

En este ejemplo, tanto **Empleado** como **Cliente** heredan los atributos comunes de **Persona**, pero también tienen sus propios atributos específicos.

Tipos de herencia

En el modelado entidad-relación extendido, la herencia puede ser de dos tipos:

- **Herencia total:** Cuando todos los elementos de la superclase deben pertenecer a una subclase.
- **Herencia parcial:** Cuando algunos elementos de la superclase no pertenecen a ninguna subclase.

Aplicación en Bases de Datos

La herencia en el modelo entidad-relación extendido se traduce en la organización eficiente de los datos. En algunos sistemas de bases de datos, se puede representar con tablas donde las subclases tienen una referencia a la tabla de la superclase, o bien con estrategias de diseño como la **herencia de tabla por jerarquía** o **tabla por clase concreta**.

En resumen:

- **Superclase:** entidad general que agrupa características comunes.
- **Subclase:** entidad específica que hereda de la superclase.
- **Herencia:** mecanismo que permite a las subclases heredar atributos y relaciones de la superclase.

Restricciones del modelo relacional

En el **modelo relacional**, las **restricciones** son condiciones o reglas que definen la validez de los datos almacenados en una base de datos. Estas restricciones garantizan la **integridad de los datos**, es decir, que los datos sean correctos, consistentes y reflejen correctamente la realidad que representan. A continuación se describen las principales restricciones del modelo relacional:

1. Restricción de Dominio

Cada atributo (columna) de una relación (tabla) debe tener un **dominio** de valores permitidos, lo que significa que los valores de un atributo deben pertenecer a un tipo de dato específico o a un conjunto de valores permitidos.

- **Ejemplo:** Si el atributo *edad* tiene un dominio de números enteros positivos, entonces no se permitirá almacenar un valor no entero o negativo en ese campo.

2. Restricción de Clave (Clave Primaria)

Una **clave primaria** es un conjunto de uno o más atributos que identifica de manera **única** cada tupla (fila) en una relación (tabla). Las restricciones de clave aseguran que:

- Ningún valor en los atributos que forman la clave primaria puede ser **NULL**.
- No puede haber duplicados en los valores de la clave primaria.

- **Ejemplo:** Si una tabla `Empleados` tiene una clave primaria en el atributo `id_empleado`, cada empleado debe tener un valor único en `id_empleado`.

3. Restricción de Clave Foránea (Integridad Referencial)

Una **clave foránea** es un conjunto de atributos en una tabla que referencia la clave primaria de otra tabla. Las **restricciones de integridad referencial** aseguran que los valores de la clave foránea deben coincidir con los valores existentes en la clave primaria de la tabla referenciada, o pueden ser `NULL` si está permitido.

- **Ejemplo:** Si una tabla `Ventas` tiene una clave foránea `id_cliente` que hace referencia a la clave primaria `id_cliente` en la tabla `Clientes`, entonces no se puede registrar una venta para un cliente que no exista en la tabla `Clientes`.

4. Restricción de Integridad de Entidad

Esta restricción asegura que **ningún valor en la clave primaria** de una tabla puede ser `NULL`. Es decir, cada fila debe estar correctamente identificada por un valor de clave primaria no nulo.

- **Ejemplo:** En una tabla de `Pedidos`, la clave primaria `id_pedido` no puede tener valores `NULL`, ya que cada pedido debe tener un identificador único.

5. Restricción de Integridad Referencial

La **integridad referencial** asegura que las relaciones entre las tablas sean coherentes. Cuando se crea una relación entre dos tablas mediante una clave foránea, se debe garantizar que:

- No se puede insertar una tupla con un valor de clave foránea que no exista en la tabla referenciada.
- Si se elimina o actualiza una tupla en la tabla referenciada, las tuplas dependientes en otras tablas deben ser actualizadas o eliminadas de manera consistente (con opciones como `CASCADE` o `SET NULL`).

6. Restricción de Unicidad

Asegura que los valores de un conjunto de atributos en una tabla sean **únicos**, pero a diferencia de la clave primaria, permite valores `NULL` en los atributos.

- **Ejemplo:** Si una tabla `Empleados` tiene un campo `email` que se establece como único, ningún empleado puede tener el mismo correo electrónico, pero se podría permitir un `NULL` si no tiene correo.

7. Restricción CHECK

La restricción **CHECK** impone una condición lógica sobre los valores que se pueden insertar en una columna. Si la condición no se cumple, el valor no se puede insertar o actualizar.

- **Ejemplo:** Si se establece una restricción **CHECK** en una columna `edad` de que `edad > 18`, no se podrán insertar valores menores o iguales a 18.

8. Restricción NOT NULL

La restricción **NOT NULL** asegura que una columna no puede tener valores **NULL**. Es decir, siempre debe almacenarse un valor en esa columna cuando se inserta una nueva tupla.

- **Ejemplo:** Si la columna `nombre` en la tabla `Clientes` tiene una restricción **NOT NULL**, no se podrá insertar una fila sin proporcionar un nombre.

9. Restricción por Defecto (DEFAULT)

La restricción **DEFAULT** define un valor por defecto para una columna si no se especifica un valor explícito en una operación **INSERT**.

- **Ejemplo:** En una columna `estado_pedido` en una tabla de `Pedidos`, se puede establecer un valor por defecto como `'pendiente'`. Si no se proporciona un valor para `estado_pedido` al insertar una nueva fila, automáticamente se asignará `'pendiente'`.

Resumen de las Principales Restricciones:

1. **Dominio:** Controla los tipos de datos permitidos.
2. **Clave Primaria:** Asegura que cada fila sea identificada de manera única.
3. **Clave Foránea:** Mantiene la integridad referencial entre tablas.
4. **Integridad de Entidad:** La clave primaria no puede ser **NULL**.
5. **Integridad Referencial:** Mantiene la consistencia en las relaciones entre tablas.
6. **Unicidad:** Asegura valores únicos en un conjunto de atributos.
7. **CHECK:** Impone condiciones sobre los valores permitidos.
8. **NOT NULL:** Evita que una columna tenga valores **NULL**.
9. **DEFAULT:** Define valores por defecto para columnas.

Estas restricciones son fundamentales para mantener la integridad y consistencia de los datos en una base de datos relacional.

En álgebra relacional, las **operaciones de cierre recursivo** permiten manejar relaciones que contienen información jerárquica o de tipo recursiva, es decir, cuando los datos en una relación tienen una estructura que requiere varios niveles de relación entre sí. Un ejemplo típico es una tabla que representa una jerarquía de empleados donde cada empleado puede tener un supervisor que también es un empleado.

1. Cierre Transitivo o Cierre Recursivo de una Relación

El **cierre transitivo** es una operación que se utiliza para encontrar todas las relaciones indirectas que se derivan de una relación base. Es común en problemas jerárquicos o de grafos, como encontrar todos los empleados que dependen de un supervisor en diferentes niveles, o todas las rutas posibles entre ciudades en una red de transporte.

Ejemplo clásico:

En un grafo dirigido donde se representan relaciones jerárquicas o conexiones directas entre nodos (entidades), el cierre transitivo permite encontrar todos los caminos posibles desde un nodo de partida a otros nodos, no solo las conexiones directas.

Supongamos una tabla **Empleados** con una estructura jerárquica donde cada empleado tiene un supervisor (también empleado). El objetivo es encontrar todos los empleados que dependen jerárquicamente de un supervisor, incluso si son supervisores de otros empleados.

Empleado Supervisor

| | |
|----|----|
| E1 | E2 |
| E2 | E3 |
| E3 | E4 |

El cierre recursivo nos permitiría encontrar que **E1 depende de E2, E3, y E4**, y así sucesivamente.

2. Explicación Formal del Cierre Recursivo

Dada una relación $R(A,B)$, que representa que el elemento AAA está relacionado directamente con BBB, el **cierre transitivo** de R es una relación R^+ tal que si AAA está relacionado con BBB y BBB está relacionado con CCC, entonces AAA también está relacionado con CCC.

En álgebra relacional, el cierre transitivo se define mediante una secuencia de uniones (operación de unión repetida) hasta que no se puedan generar nuevas relaciones.

3. Operación de Renombrar

En álgebra relacional, la **operación de renombrar** se utiliza para cambiar los nombres de los atributos de una relación, lo que es útil en situaciones donde se requieren

múltiples relaciones con atributos que deben ser diferenciados en una operación conjunta, como en una **auto unión** (self-join) o cuando se quiere mantener un código más legible.

Notación:

La operación de renombrar se expresa como $\rho(\text{nuevo_nombre})(\text{relacion})$, donde `nuevo_nombre` es el nombre que se le quiere asignar al atributo o relación, y `relación` es el nombre original de la relación.

Ejemplo:

Supongamos que tenemos una relación `Empleados(id,nombre,supervisor)` y queremos hacer una auto unión de esta tabla para encontrar los empleados y sus respectivos supervisores. Para diferenciar entre el empleado y su supervisor, usamos la operación de renombrar:

$$\rho E1(\text{Empleados}) \bowtie \rho E2(\text{Empleados})$$

Esto nos permite cambiar los nombres de los atributos para diferenciarlos en la operación de **join**.

Resumen:

- **Cierre recursivo o cierre transitivo:** Se usa para encontrar relaciones indirectas en estructuras jerárquicas o transitivas.
- **Operación de renombrar:** Se utiliza para cambiar los nombres de atributos o relaciones, facilitando operaciones complejas como las auto uniones.

Estas operaciones son importantes para manipular datos jerárquicos o recursivos y para gestionar relaciones complejas en álgebra relacional.

Convención de nombres en el modelo relacional

En el **modelo relacional**, las convenciones de nombres para entidades y atributos son importantes para garantizar que los esquemas de bases de datos sean legibles, comprensibles y mantenibles. Aunque no existe un estándar único o universal, hay ciertas **prácticas recomendadas** y convenciones que se suelen seguir para nombrar entidades y atributos.

1. Nombres de Entidades

Las **entidades** son representadas como **tablas** en el modelo relacional. El nombre de la entidad debe ser claro, representativo y estar en singular (aunque en algunos casos se prefiera el plural). A continuación se describen las principales convenciones:

Convenciones:

- **Nombre significativo y descriptivo:** El nombre de la entidad debe reflejar claramente el tipo de información que contiene. Por ejemplo, Empleado, Cliente, Producto, Factura.
- **Singular vs. Plural:** Se suele usar el **singular** para el nombre de la tabla (Empleado en lugar de Empleados), ya que cada fila representa un registro individual. Sin embargo, algunas convenciones permiten usar el plural (Empleados).
- **Mayúsculas y minúsculas:**
 - Convención 1: Usar **CamelCase** (primer letra de cada palabra en mayúscula, sin espacios). Ejemplo: Empleado, ClientePreferente.
 - Convención 2: Usar **snake_case** (palabras separadas por guiones bajos y en minúsculas). Ejemplo: empleado, cliente_preferente.
- **Evitar nombres reservados:** No utilizar palabras reservadas de SQL como nombres de entidades (por ejemplo, User, Order, Date).
- **Evitar abreviaciones innecesarias:** Siempre que sea posible, se deben evitar abreviaciones que puedan causar confusión. Por ejemplo, en lugar de Emp, usar Empleado.

Ejemplos:

- Correcto: Empleado, Producto, Pedido.
- Incorrecto: Emp, Pro, Ord.

2. Nombres de Atributos

Los **atributos** son las **columnas** de las tablas que representan las propiedades o características de las entidades. Las convenciones para los nombres de atributos también deben seguir criterios claros.

Convenciones:

- **Descriptivos y únicos:** Cada atributo debe tener un nombre que describa su contenido de manera clara y única dentro del contexto de la entidad. Ejemplo: nombre, apellido, precio_producto.
- **No incluir el nombre de la entidad en el atributo:** No es necesario repetir el nombre de la entidad en el nombre de cada atributo. Ejemplo:
 - Correcto: nombre, apellido.
 - Incorrecto: empleado_nombre, empleado_apellido.

Solo se incluye el nombre de la entidad si es absolutamente necesario, como en claves externas.

- **Mayúsculas y minúsculas:** Al igual que con las entidades, se puede utilizar **CamelCase** o **snake_case** para los nombres de los atributos, dependiendo de las convenciones establecidas:
 - CamelCase: FechaIngreso, CodigoPostal.
 - snake_case: fecha_ingreso, codigo_postal.

- **Prefijos para claves externas:** Para facilitar la identificación de claves externas, se puede utilizar el nombre de la entidad referenciada como prefijo. Ejemplo:
 - `id_cliente` como clave externa que referencia a la tabla `Cliente`.
- **Evitar nombres genéricos:** No usar nombres ambiguos o genéricos como `dato`, `valor`, `estado`. En su lugar, usar nombres descriptivos como `fecha_creacion`, `precio_producto`, `estado_pedido`.

Ejemplos:

- **Correcto:** `nombre`, `apellido`, `fecha_nacimiento`, `id_departamento`.
- **Incorrecto:** `dato1`, `estado`, `id_user`, `nombre_usuario_apellido`.

3. Convenciones para Claves Primarias

La clave primaria es un atributo (o conjunto de atributos) que identifica de manera única a cada fila en la tabla.

- **Nombre estándar:** Se recomienda usar nombres como `id` o `id_entidad` para las claves primarias. Ejemplo: `id_cliente`, `id_empleado`.
- **Singularidad:** La clave primaria debe ser única y no debe tener valores nulos.

4. Convenciones para Claves Externas

Las claves externas son atributos que hacen referencia a la clave primaria de otra tabla.

- **Nombres descriptivos:** El nombre de la clave externa debe reflejar la entidad que está referenciando. Ejemplo: `id_cliente` en la tabla `Pedidos` que hace referencia a `Cliente`.

Resumen de Convenciones:

1. **Nombres de entidades:**
 - Singulares, significativos y descriptivos.
 - Evitar abreviaturas y nombres reservados.
 - Preferencias: `Empleado`, `Producto` o `empleado`, `producto`.
2. **Nombres de atributos:**
 - Descriptivos y claros.
 - Sin incluir el nombre de la entidad (a menos que sea una clave externa).
 - Evitar nombres genéricos como `dato` o `valor`.
3. **Claves primarias y externas:**
 - Clave primaria: `id` o `id_entidad`.
 - Clave externa: `id_entidad` o `entidad_id`.

Estas convenciones son esenciales para garantizar la claridad y el mantenimiento de las bases de datos a lo largo del tiempo, especialmente en proyectos grandes donde participan múltiples desarrolladores.

Reglas de Precedencia:

1. **Selección (σ) y Proyección (π)** tienen **mayor precedencia** que el **join (\bowtie)** o producto cartesiano.
 - o Esto significa que, si no hay paréntesis, se aplican primero las operaciones de selección y proyección antes de realizar un join o un producto cartesiano entre dos relaciones.
2. **Join (\bowtie)** o producto cartesiano (\times) tiene **menor precedencia**, por lo que se realiza después de que se hayan aplicado las selecciones y proyecciones.

Ejemplo 1: Sin paréntesis (con selección y join)

σ condición ($R \bowtie S$)

Este se resuelve primero como el **join** entre R y S y luego se aplica la **selección** sobre el resultado del join.

Ejemplo 2: Sin paréntesis (con proyección y join)

π atributo ($R \bowtie S$)

En este caso, primero se realiza el **join** entre R y S , y luego se aplica la **proyección** sobre el resultado del join.

Ejemplo 3: Sin paréntesis (combinación de selección, proyección y join)

π atributo (σ condición ($R \bowtie S$))

Este se resuelve en tres pasos:

1. Primero se hace el **join** entre R y S .
2. Luego se aplica la **selección** (σ) sobre el resultado del join.
3. Finalmente, se aplica la **proyección** (π) sobre el resultado de la selección.

Ejemplo con las tres operaciones:

Supongamos que tenemos:

π nombre_paciente (σ especialidad = 'Cardiología' ($\text{Pacientes} \bowtie \text{Consultas} \bowtie \text{Doctores}$))

1. Primero se hace el **join** entre Pacientes , Consultas y Doctores .
2. Luego, se aplica la **selección** donde la especialidad del doctor es "Cardiología".
3. Finalmente, se aplica la **proyección** para obtener únicamente el atributo `nombre_paciente`.

Secuencia de Ejecución en ausencia de paréntesis:

1. Primero, **selección** (σ) y **proyección** (π) si están presentes, ya que tienen mayor precedencia.
2. Después, se ejecutan los **joins** (\bowtie) o productos cartesianos.

Resumen:

- Sin paréntesis, el álgebra relacional sigue la precedencia estándar:
 1. **Selección** (σ) y **Proyección** (π) se ejecutan primero.
 2. Luego, se ejecutan los **joins** (\bowtie) o productos cartesianos.
- Si quieres controlar el orden de evaluación de manera explícita, es recomendable usar paréntesis para agrupar operaciones.