UF2176 - DEFINICIÓN Y MANIPULACIÓN DE DATOS

Unidad 1: Lenguajes relacionales

By: Sergi Faura Alsina

SOC Servei d'Ocupació de Catalunya
Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SEPE

Índice

1. Lenguajes Relacionales

Los lenguajes relacionales son fundamentales para la interacción con bases de datos en el modelo relacional, uno de los enfoques más extendidos en el diseño y manejo de bases de datos. Estos lenguajes permiten a los usuarios formular consultas para gestionar y extraer información de las bases de datos de manera eficiente. Se dividen en dos grandes categorías: el álgebra relacional y el cálculo relacional, cada uno con diferentes enfoques y operaciones que permiten la manipulación de los datos.

El **álgebra relacional** es un lenguaje procedimental que define un conjunto de operaciones que pueden realizarse sobre relaciones para obtener otras relaciones. Estas operaciones incluyen la **selección** (filtrar filas), **proyección** (seleccionar columnas), el **producto cartesiano** (combinación de dos relaciones), así como las operaciones **de unión**, **diferencia**, **intersección**, **join** y **división**, entre otras. Estas herramientas permiten transformar y derivar nuevas relaciones a partir de las ya existentes.

Por otro lado, el **cálculo relacional** es un lenguaje no procedimental que se centra en lo que se desea obtener, en lugar de cómo obtenerlo. Se divide en dos variantes: el **cálculo orientado a dominios** y el **cálculo orientado a tuplas**. Ambas técnicas se utilizan para especificar consultas de manera declarativa, es decir, el usuario describe las condiciones que deben cumplir los datos, y el sistema se encarga de encontrar los resultados.

Los lenguajes comerciales como **SQL** (Structured Query Language) y **QBE** (Query By Example) son implementaciones prácticas que permiten interactuar con bases de datos relacionales de manera sencilla y potente. SQL, en particular, ha evolucionado considerablemente desde sus inicios, convirtiéndose en el estándar de facto para la mayoría de los sistemas de gestión de bases de datos relacionales.

Este conjunto de herramientas permite no solo la gestión de los datos en una base de datos, sino también la optimización de consultas a través de la **transformación entre álgebra y cálculo relacional**, lo que proporciona a los desarrolladores y administradores de bases de datos una base sólida para manejar grandes volúmenes de información de manera eficiente y escalable.

1.1. Tipos de lenguajes relacionales.

Los lenguajes relacionales se utilizan para formular consultas y gestionar datos dentro del modelo relacional de bases de datos. Existen dos tipos principales de lenguajes relacionales:

1. **Álgebra Relacional**: Es un lenguaje **procedimental**, lo que significa que el usuario especifica una serie de operaciones que se deben realizar para obtener el resultado

deseado. Estas operaciones actúan sobre relaciones (tablas) y producen nuevas relaciones. Las operaciones más comunes incluyen:
    - Selección
    - Proyección
    - Unión
    - Diferencia
    - Producto cartesiano
    - Join
2. **Cálculo Relacional**: Es un lenguaje **no procedimental**, donde el usuario indica **qué** resultado quiere, sin especificar los pasos para obtenerlo. Se enfoca en describir las condiciones que deben cumplir los datos. Hay dos variantes del cálculo relacional:
    - **Cálculo Relacional Orientado a Tuplas**: Las consultas se centran en las tuplas (filas) de las relaciones.
    - **Cálculo Relacional Orientado a Dominios**: Las consultas se centran en los valores de los atributos (columnas).

Ambos tipos de lenguajes proporcionan diferentes enfoques para consultar y manipular los datos en un sistema de bases de datos relacional.

1.2. Operaciones en el modelo relacional:

Las operaciones en el modelo relacional son el conjunto de acciones que se pueden realizar sobre las relaciones (tablas) en una base de datos relacional para manipular y consultar los datos. Estas operaciones permiten extraer información relevante, combinar tablas y derivar nuevas relaciones a partir de las existentes. Las operaciones están basadas en el **álgebra relacional**, que define un conjunto de operadores que actúan sobre las tablas, permitiendo transformarlas de acuerdo con las necesidades del usuario.

Las operaciones en el modelo relacional se dividen en dos categorías:

- **Operaciones fundamentales (primitivas)**: Son las operaciones básicas que se aplican directamente a las relaciones. Estas operaciones incluyen la selección, proyección, producto cartesiano, unión y diferencia, y son esenciales para la manipulación de los datos.
- **Operaciones derivadas**: Son operaciones más complejas que pueden ser definidas a partir de las primitivas. Entre ellas se incluyen la intersección, el join (o combinación de tablas), y la división. Estas operaciones son útiles para realizar consultas más avanzadas y específicas.

El conjunto de estas operaciones proporciona una base sólida para la manipulación y consulta de datos en sistemas de bases de datos relacionales, asegurando una interacción eficiente y estructurada con los datos almacenados.

1.2.1. Álgebra relacional:

El **álgebra relacional** es un lenguaje procedimental utilizado para realizar consultas sobre una base de datos relacional. A través de un conjunto de operaciones, permite manipular las relaciones (tablas) para obtener resultados específicos. Cada operación toma una o más relaciones como entrada y genera una nueva relación como resultado. Este lenguaje es esencial para la teoría de bases de datos relacionales, ya que proporciona un enfoque formal y matemático para la manipulación de datos.

1.2.1.1. Clasificación de operaciones

Las operaciones del álgebra relacional se dividen en dos tipos principales: **operaciones fundamentales (primitivas)** y **operaciones derivadas**. Estas operaciones son las herramientas clave para realizar todo tipo de consultas dentro de una base de datos relacional.

- **Operaciones fundamentales (primitivas)**: Son las operaciones básicas necesarias para realizar cualquier consulta en una base de datos relacional. Estas operaciones incluyen:
    - **Selección (σ)**: Permite filtrar las filas de una tabla que cumplen una determinada condición.
    - **Proyección (π)**: Selecciona un subconjunto de columnas de una tabla.
    - **Producto cartesiano (×)**: Combina todas las tuplas posibles entre dos relaciones.
    - **Unión (∪)**: Combina las tuplas de dos relaciones eliminando duplicados.
    - **Diferencia (-)**: Devuelve las tuplas que están en una relación pero no en la otra.
- **Operaciones derivadas**: Estas operaciones se construyen a partir de las operaciones fundamentales y permiten realizar consultas más complejas:
    - **Intersección (∩)**: Devuelve las tuplas comunes a dos relaciones.
    - **Join (⋈)**: Combina dos relaciones basándose en una condición específica de igualdad entre atributos.
    - **División (÷)**: Utilizada para encontrar tuplas que se corresponden con todas las tuplas de otra relación.

El álgebra relacional es un lenguaje que permite la construcción de consultas mediante la combinación de estas operaciones, proporcionando una base sólida y estructurada para la manipulación de bases de datos relacionales.

1.2.1.2. Denominación de atributos.

En el **álgebra relacional**, los atributos (columnas) de una relación juegan un papel clave en la definición y manipulación de los datos. Los atributos deben tener nombres únicos dentro de una relación para evitar ambigüedades durante las consultas. Sin embargo, a medida que se realizan operaciones como el **producto cartesiano** o el **join** entre múltiples relaciones, pueden surgir conflictos de nombres entre atributos que tienen el mismo nombre pero provienen de diferentes tablas. Para resolver estos conflictos y evitar confusión, el álgebra relacional proporciona mecanismos para renombrar atributos.

Renombramiento de Atributos

El proceso de **renombrar** atributos es fundamental cuando se realizan operaciones que involucran múltiples relaciones o cuando se desea mejorar la claridad de los resultados de una consulta. El renombramiento permite asignar un nuevo nombre a uno o más atributos para evitar conflictos o hacer que los resultados sean más comprensibles. Esta operación de renombramiento se denota mediante el operador **ρ (rho)** y se utiliza de la siguiente manera:

- **ρ(nombre_nuevo, nombre_antiguo)**: Esta notación permite renombrar un atributo de una relación, reemplazando el `nombre_antiguo` con el `nombre_nuevo`.
- **ρ(R(nombre_nuevo1, nombre_nuevo2,...), Relación)**: También es posible renombrar múltiples atributos de una relación al mismo tiempo. Esta notación cambia los nombres de los atributos de la relación "Relación" por los nuevos nombres especificados.

El renombramiento no modifica la estructura de la relación original; simplemente proporciona un alias temporal para los atributos, lo que es útil en consultas complejas.

Importancia del Renombramiento

El renombramiento de atributos es especialmente útil en las siguientes situaciones:

- **Producto cartesiano o join**: Al realizar operaciones que combinan varias relaciones, es posible que existan atributos con el mismo nombre en las diferentes relaciones. El renombramiento permite diferenciar estos atributos para evitar confusiones.
- **Claridad en los resultados**: En ocasiones, es conveniente cambiar el nombre de los atributos para que los resultados de una consulta sean más fáciles de entender. Por ejemplo, si el resultado de una consulta tiene atributos con nombres técnicos o ambiguos, se pueden renombrar para darles nombres más intuitivos.

- **Evitar ambigüedades**: Cuando se trabaja con varias relaciones en una misma consulta, el uso de nombres claros y únicos para los atributos es crucial para evitar errores y ambigüedades en la interpretación de los resultados.

En resumen, la **denominación y renombramiento de atributos** es un aspecto crucial en el álgebra relacional que asegura la claridad y precisión en la manipulación de datos, especialmente cuando se combinan múltiples relaciones en una consulta.

1.2.1.3. Relaciones derivadas.

En el álgebra relacional, las **relaciones derivadas** son aquellas que no existen originalmente en la base de datos, sino que se generan a partir de la aplicación de una o más operaciones relacionales sobre relaciones ya existentes. Estas relaciones se producen como resultado de operaciones como selección, proyección, producto cartesiano, unión, diferencia, intersección, join, entre otras.

Las relaciones derivadas son temporales y no se almacenan de manera permanente en la base de datos a menos que se especifique. Sin embargo, son de gran utilidad, ya que permiten realizar consultas complejas y transformaciones sobre los datos en tiempo real, sin la necesidad de modificar las tablas subyacentes. A continuación, se detallan los aspectos más importantes de las relaciones derivadas en el álgebra relacional.

Generación de Relaciones Derivadas

Las relaciones derivadas surgen cuando aplicamos las operaciones del álgebra relacional sobre una o varias relaciones. A medida que se encadenan o combinan estas operaciones, se generan nuevas relaciones que pueden ser utilizadas en consultas posteriores. Ejemplos de operaciones que generan relaciones derivadas son:

- **Selección (σ)**: Extrae las filas de una relación que cumplen con una condición específica. La nueva relación derivada contiene solo las tuplas filtradas.
- **Proyección (π)**: Genera una nueva relación que incluye solo ciertos atributos (columnas) de una relación existente.
- **Unión (∪)**: Combina dos relaciones con el mismo esquema y genera una relación que contiene todas las tuplas de ambas relaciones, eliminando duplicados.
- **Join (⋈)**: Combina dos relaciones en función de una condición de igualdad entre sus atributos, generando una nueva relación que contiene tuplas combinadas de ambas.

Utilidad de las Relaciones Derivadas

Las relaciones derivadas permiten realizar operaciones complejas y transformar datos sin alterar la estructura original de las tablas. Algunas de las principales utilidades son:

- **Consulta de datos sin modificar las tablas originales**: Al aplicar operaciones relacionales, podemos obtener una vista derivada de los datos que se ajusta a las necesidades específicas de una consulta, sin cambiar los datos originales.
- **Facilitación de consultas complejas**: Las relaciones derivadas permiten descomponer consultas complejas en operaciones más simples, donde el resultado de una operación puede ser utilizado como entrada para otra.
- **Flexibilidad y reutilización**: Las relaciones derivadas pueden ser reutilizadas dentro de otras operaciones. Por ejemplo, es posible realizar una selección sobre una proyección, y el resultado será una relación derivada que puede utilizarse en nuevas operaciones.

Ejemplo de Relaciones Derivadas

Consideremos una base de datos con una relación **Empleado** que tiene los atributos **Nombre**, **Departamento** y **Salario**. Si queremos obtener una relación derivada que contenga solo los empleados del departamento "Ventas" con un salario mayor a 2000, podemos utilizar la siguiente consulta de selección:

$$\sigma\ (\text{Departamento} = \text{'Ventas'} \wedge \text{Salario} > 2000)\ (\text{Empleado})$$

El resultado será una **relación derivada** que contiene solo las tuplas de la relación original que cumplen con ambas condiciones.

En resumen, las **relaciones derivadas** son un resultado directo de aplicar operaciones del álgebra relacional sobre relaciones existentes, y son fundamentales para la manipulación temporal y transformación de los datos en bases de datos relacionales. Permiten construir consultas avanzadas, generar nuevas perspectivas de los datos y facilitar la realización de análisis complejos sin comprometer la integridad de las tablas originales.

1.2.1.4. Operaciones primitivas: selección, proyección, producto cartesiano, unión y diferencia.

Las **operaciones primitivas** en el álgebra relacional son las operaciones básicas y fundamentales que se pueden aplicar sobre las relaciones (tablas) en una base de datos relacional. Estas operaciones permiten manipular los datos de diversas maneras y son esenciales para la construcción de consultas. Todas las demás operaciones derivadas

pueden ser expresadas a partir de estas operaciones primitivas. A continuación, se describen en detalle las principales operaciones primitivas: **selección**, **proyección**, **producto cartesiano**, **unión** y **diferencia**.

1. Selección (σ)

La operación de **selección** se utiliza para filtrar las tuplas (filas) de una relación que cumplen con una condición específica. Esta operación devuelve una nueva relación que contiene solo aquellas tuplas que satisfacen la condición dada. La selección se denota mediante la letra griega **σ** seguida de la condición a evaluar y la relación sobre la que se aplica.

- Notación: `σ_condición(Relación)`
- Ejemplo: `σ_Salario > 2000 (Empleado)` devuelve las filas de la relación `Empleado` donde el salario es mayor a 2000.

La selección actúa sobre las filas de la tabla, pero no modifica los atributos (columnas).

2. Proyección (π)

La operación de **proyección** se utiliza para seleccionar columnas específicas de una relación, eliminando duplicados. Esta operación devuelve una nueva relación que contiene solo los atributos indicados por el usuario, y elimina cualquier repetición de tuplas resultante de la operación.

- Notación: `π_atributos(Relación)`
- Ejemplo: `π_Nombre, Salario (Empleado)` devuelve una relación con solo los atributos `Nombre` y `Salario` de la tabla `Empleado`.

La proyección actúa sobre los atributos de la relación, eliminando las columnas no deseadas y cualquier duplicado.

3. Producto Cartesiano (×)

El **producto cartesiano** es una operación que combina todas las tuplas posibles entre dos relaciones. El resultado es una nueva relación que contiene todas las combinaciones de filas de las dos relaciones originales. El número de filas en el producto cartesiano es el producto del número de filas de las dos relaciones de entrada.

- Notación: `Relación1 × Relación2`

- Ejemplo: Si `Empleado` tiene 5 filas y `Departamento` tiene 3 filas, entonces `Empleado × Departamento` tendrá 15 filas (5 x 3).

El producto cartesiano es útil como paso intermedio en otras operaciones como el **join**, pero por sí solo puede generar grandes volúmenes de datos irrelevantes si no se usa con condiciones de filtrado adicionales.

4. Unión (∪)

La operación de **unión** combina las tuplas de dos relaciones que tienen el mismo esquema (los mismos atributos) en una nueva relación. La unión elimina las tuplas duplicadas, devolviendo un conjunto único de tuplas que aparecen en cualquiera de las dos relaciones.

- Notación: `Relación1 ∪ Relación2`
- Ejemplo: Si `Empleado1` y `Empleado2` son relaciones con el mismo esquema, entonces `Empleado1 ∪ Empleado2` devuelve una relación con todas las tuplas de ambas relaciones, sin duplicados.

La unión solo puede aplicarse si las dos relaciones tienen el mismo número de atributos y los mismos tipos de datos para cada atributo.

5. Diferencia (-)

La operación de **diferencia** devuelve las tuplas que están en una relación pero no en la otra. Al igual que la unión, la diferencia solo se puede aplicar si ambas relaciones tienen el mismo esquema.

- Notación: `Relación1 - Relación2`
- Ejemplo: Si `Empleado1` y `Empleado2` tienen el mismo esquema, entonces `Empleado1 - Empleado2` devolverá una relación con las tuplas que están en `Empleado1` pero no en `Empleado2`.

La diferencia es útil para comparar conjuntos de datos y encontrar tuplas exclusivas de una relación en particular.

Resumen

Las **operaciones primitivas** del álgebra relacional (selección, proyección, producto cartesiano, unión y diferencia) son las herramientas fundamentales para la manipulación y consulta de datos en una base de datos relacional. Permiten filtrar, combinar y transformar

datos de manera flexible, proporcionando la base para consultas más complejas que se pueden construir a partir de estas operaciones.

1.2.1.5. Otras operaciones: intersección, join, división, etc.

Además de las operaciones primitivas del álgebra relacional, existen otras **operaciones derivadas** que proporcionan mayor flexibilidad y potencia para las consultas en bases de datos relacionales. Estas operaciones, aunque no son fundamentales, se construyen a partir de las primitivas y permiten realizar consultas más avanzadas y especializadas. Las más importantes entre estas operaciones son la **intersección**, el **join** y la **división**. El **join**, en particular, tiene varias variantes, cada una adaptada a diferentes tipos de consultas.

1. Intersección (∩)

La operación de **intersección** devuelve las tuplas que son comunes a dos relaciones. Es decir, el resultado incluye solo aquellas tuplas que aparecen en ambas relaciones. Las relaciones sobre las que se aplica la intersección deben tener el mismo esquema (mismo número de columnas y tipos de datos).

- Notación: `Relación1 ∩ Relación2`
- Ejemplo: Si `Empleado1` y `Empleado2` tienen el mismo esquema, `Empleado1 ∩ Empleado2` devolverá una relación con las tuplas que están presentes en ambas relaciones.

La intersección es útil cuando se necesita encontrar coincidencias entre dos conjuntos de datos.

2. Join (⋈)

La operación de **join** combina dos relaciones en función de una condición que establece una correspondencia entre sus atributos. Es una de las operaciones más poderosas del álgebra relacional, ya que permite unir datos relacionados que se encuentran en diferentes tablas. Hay varios tipos de **join**, cada uno con un propósito específico:

Tipos de Join:

1. **Natural Join (⋈)**

El **natural join** une dos relaciones sobre los atributos que tienen el mismo nombre en ambas relaciones, combinando las tuplas que coinciden en esos atributos y eliminando las columnas duplicadas.

- Notación: `Relación1 ⋈ Relación2`
- Ejemplo: Si tenemos una relación `Empleado` con el atributo `ID_Departamento` y otra relación `Departamento` con el mismo atributo, el natural join combinara las tuplas que tengan el mismo valor de `ID_Departamento`.

2. **Equijoin**

El **equijoin** es una operación de join donde se establece una condición de igualdad entre uno o más atributos de dos relaciones. A diferencia del natural join, el equijoin no elimina las columnas duplicadas.

- Notación: `Relación1 ⋈ Relación2 ON condición`
- Ejemplo: `Empleado ⋈ Departamento ON Empleado.ID_Departamento = Departamento.ID_Departamento` combina ambas tablas sin eliminar las columnas duplicadas.

3. **Theta Join (θ-join)**

El **theta join** generaliza el equijoin al permitir cualquier condición relacional (≤, ≥, <, >, ≠), no solo igualdad. Combina las tuplas de dos relaciones siempre que la condición especificada sea verdadera.

- Notación: `Relación1 ⋈θ condición Relación2`
- Ejemplo: `Empleado ⋈θ Salario > 3000 Relación2` devuelve las combinaciones de empleados cuyo salario sea mayor a 3000.

4. **Left Join (⟕)**

El **left join** o **left outer join** combina todas las tuplas de la relación de la izquierda con las correspondientes de la relación de la derecha. Si no hay una coincidencia en la relación de la derecha, las columnas correspondientes se rellenan con `NULL`. Esto es útil cuando queremos asegurarnos de que todas las tuplas de la relación izquierda se incluyan en el resultado, aunque no haya correspondencias en la relación derecha.

- **Notación**: `Relación1 ⟕ Relación2`
- **Ejemplo**: `Empleado ⟕ Departamento` devuelve todos los empleados, y si un empleado no tiene departamento asignado, las columnas de `Departamento` aparecerán como `NULL`.

El **left join** es útil en casos en los que se desea mantener toda la información de una relación, incluso si no hay datos correspondientes en la otra.

5. **Right Join (⟖)**

El **right join** o **right outer join** es similar al left join, pero devuelve todas las tuplas de la relación de la derecha, y si no hay coincidencia en la relación de la izquierda, las columnas de la relación izquierda se rellenan con `NULL`. Esto asegura que todas las tuplas de la relación derecha estén presentes en el resultado, aunque no haya correspondencia en la izquierda.

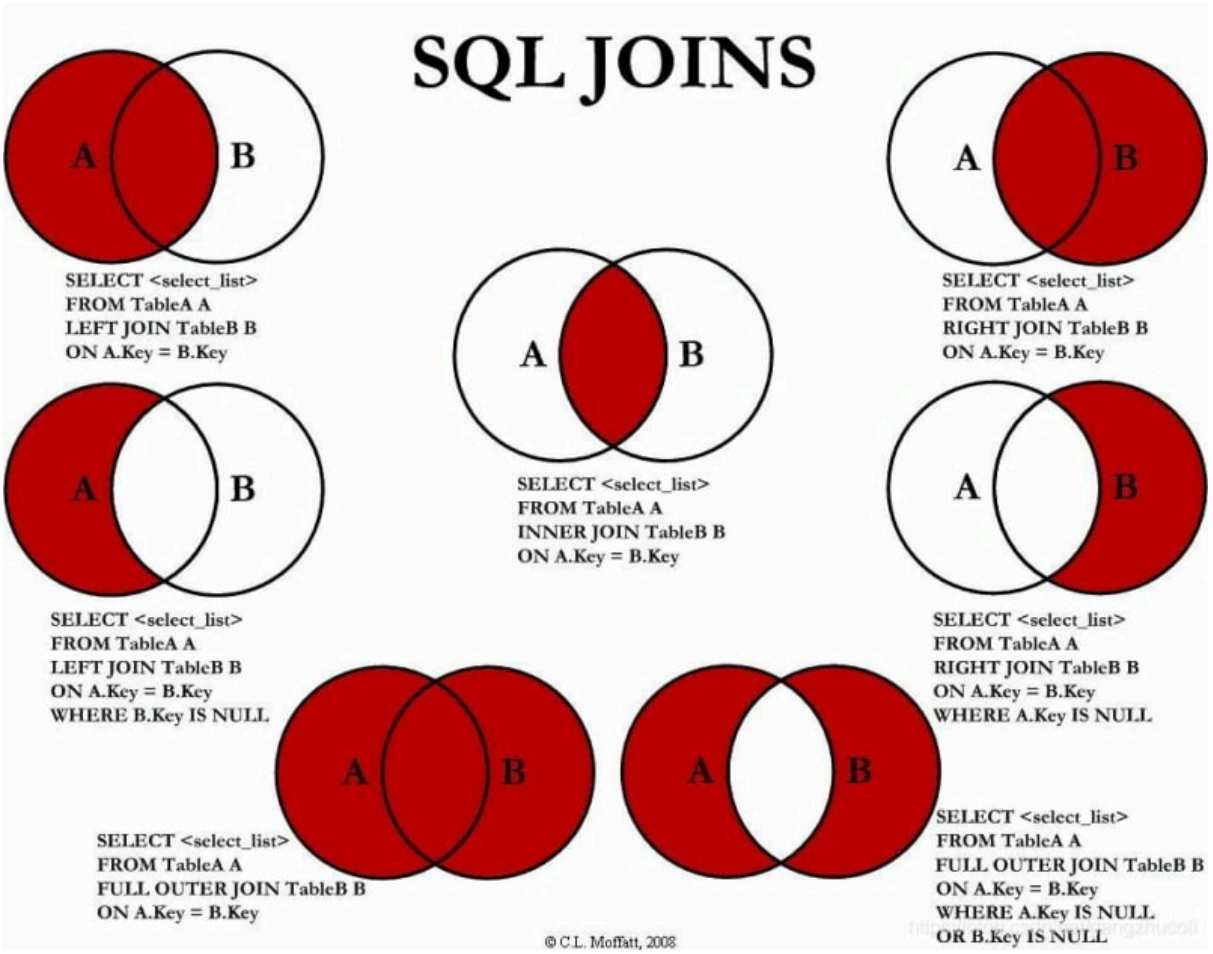
- **Notación**: `Relación1 ⟖ Relación2`
- **Ejemplo**: `Empleado ⟖ Departamento` devuelve todos los departamentos, y si un departamento no tiene empleados asignados, las columnas correspondientes a `Empleado` se rellenarán con `NULL`.

Este tipo de join es útil cuando se quiere garantizar que todas las tuplas de una relación aparezcan en el resultado, independientemente de la existencia de tuplas coincidentes en la otra relación.

6. **Full Outer Join (⟗)**

El **full outer join** combina las propiedades del left outer join y el right outer join. Devuelve todas las tuplas de ambas relaciones, llenando con `NULL` las columnas faltantes en cualquiera de las dos relaciones cuando no hay coincidencias.

- Notación: `Relación1 ⟗ Relación2`
- Ejemplo: Combina todas las tuplas de `Empleado` y `Departamento`, incluyendo aquellas tuplas sin correspondencia en cualquiera de las dos relaciones, rellenando con `NULL` cuando sea necesario.



3. División (÷)

La operación de **división** es utilizada para consultas que involucran el concepto de "para todos". Es útil cuando se quiere obtener tuplas que estén relacionadas con **todas** las tuplas de otra relación. La división devuelve las tuplas de la relación dividida que tienen correspondencias con todas las tuplas de la relación divisor.

- Notación: `Relación1 ÷ Relación2`
- Ejemplo: Si `Empleado` tiene los empleados y sus proyectos, y `Proyecto` tiene todos los proyectos disponibles, la división `Empleado ÷ Proyecto` devolverá los empleados que trabajan en **todos** los proyectos de la relación `Proyecto`.

4. Renombramiento (ρ)

Aunque no es una operación que manipule los datos, el **renombramiento** permite cambiar los nombres de las relaciones o de los atributos de una relación para evitar ambigüedades, especialmente en operaciones como el producto cartesiano o los joins.

- Notación: `ρ(nombre_nuevo, Relación)`
- Ejemplo: `ρ(E, Empleado)` renombra la relación `Empleado` como `E`.

Resumen

Las operaciones derivadas, como la **intersección**, los diferentes tipos de **join** y la **división**, ofrecen herramientas más avanzadas para consultar y combinar datos en bases de datos relacionales. Estas operaciones permiten realizar consultas complejas y flexibles, facilitando el trabajo con datos distribuidos entre varias relaciones. Cada tipo de join está diseñado para resolver situaciones específicas en la combinación de tablas, desde coincidencias exactas hasta la inclusión de tuplas no coincidentes. La división es especialmente útil para responder consultas de tipo "para todos", completando el conjunto de herramientas disponibles en el álgebra relacional.

1.2.2. Cálculo relacional:

El **cálculo relacional** es otro de los enfoques utilizados en el modelo relacional para expresar consultas sobre bases de datos. A diferencia del **álgebra relacional**, que es procedimental (especifica cómo obtener los resultados), el cálculo relacional es un lenguaje **no procedimental** o **declarativo**. Esto significa que en lugar de especificar los pasos exactos para obtener los resultados, se describe **qué** información se desea obtener, basándose en las condiciones que deben cumplir los datos.

El cálculo relacional se basa en la lógica de predicados y permite formular consultas en términos de **tuplas** o **dominios**. Estas consultas especifican qué condiciones deben cumplir los datos, y el sistema de bases de datos se encarga de determinar cómo obtener las tuplas que satisfacen esas condiciones.

1.2.2.1. Cálculo relacional orientado a dominios.

El **cálculo relacional orientado a dominios** (Domain Relational Calculus, DRC) se centra en las variables que representan los **valores de los dominios de los atributos** (columnas) de las relaciones. En este enfoque, las consultas especifican qué valores deben cumplir los

atributos de las tuplas que se desean obtener. Cada variable representa un valor individual de un dominio, y las condiciones se formulan en términos de esos valores.

Características del Cálculo Relacional Orientado a Dominios:

- Las variables de la consulta representan valores individuales de atributos (dominios), no tuplas completas.
- Las consultas consisten en una fórmula lógica que describe las condiciones que deben cumplir los valores de los atributos.
- Se utiliza la lógica de predicados para especificar las condiciones, lo que hace que el cálculo orientado a dominios sea similar a un lenguaje declarativo.

Sintaxis General del Cálculo Relacional Orientado a Dominios:

Una consulta en el cálculo relacional orientado a dominios se puede expresar de la siguiente forma:

```
{<d1, d2, ..., dn> | Fórmula(d1, d2, ..., dn)}
```

Donde:

- `<d1, d2, ..., dn>` son las variables que representan los valores de los atributos que se desean recuperar.
- `Fórmula(d1, d2, ..., dn)` es una expresión lógica que describe las condiciones que deben cumplir los valores de los dominios.

Ejemplo de Cálculo Relacional Orientado a Dominios:

Supongamos que tenemos una relación `Empleado(Nombre, Departamento, Salario)` y queremos obtener los nombres de los empleados que trabajan en el departamento "Ventas" y ganan más de 3000. La consulta sería:

```
{<n> | ∃ d ∃ s (Empleado(n, d, s) ∧ d = 'Ventas' ∧ s > 3000)}
```

En este caso:

- `n` es una variable que representa el nombre del empleado.
- `d` representa el departamento.
- `s` representa el salario.
- La fórmula lógica `Empleado(n, d, s) ∧ d = 'Ventas' ∧ s > 3000` define las condiciones que deben cumplir los empleados: deben trabajar en "Ventas" y ganar más de 3000.

1.2.2.2. Cálculo relacional orientado a tuplas.

El **cálculo relacional orientado a tuplas** (Tuple Relational Calculus, TRC) se centra en las **tuplas completas** en lugar de los valores individuales de los atributos. En este enfoque, las consultas describen qué condiciones deben cumplir las tuplas que se desean recuperar. Cada variable representa una tupla completa de una relación, y las condiciones se formulan en función de las propiedades de esas tuplas.

Características del Cálculo Relacional Orientado a Tuplas:

- Las variables representan tuplas enteras de la relación.
- Las consultas describen las condiciones que deben cumplir las tuplas, en lugar de los valores individuales de los atributos.
- Se utiliza lógica de predicados para expresar las condiciones de las tuplas.

Sintaxis General del Cálculo Relacional Orientado a Tuplas:

Una consulta en el cálculo relacional orientado a tuplas tiene la siguiente forma:

```
{t | Fórmula(t)}
```

Donde:

- `t` es una variable que representa una tupla completa de la relación.
- `Fórmula(t)` es una expresión lógica que describe las condiciones que deben cumplir las tuplas.

Ejemplo de Cálculo Relacional Orientado a Tuplas:

Supongamos nuevamente que tenemos la relación `Empleado(Nombre, Departamento, Salario)` y queremos obtener las tuplas de empleados que trabajan en el departamento "Ventas". La consulta sería:

```
{t | t ∈ Empleado ∧ t.Departamento = 'Ventas'}
```

En este caso:

- `t` es una variable que representa una tupla de la relación `Empleado`.

- La fórmula `t ∈ Empleado ∧ t.Departamento = 'Ventas'` especifica que queremos todas las tuplas donde el departamento sea "Ventas".

1.2.2.3. Transformación de consultas entre álgebra y cálculo relacional.

Una característica importante del álgebra relacional y el cálculo relacional es que son **equivalentes en poder expresivo**. Esto significa que cualquier consulta que se puede expresar en álgebra relacional también se puede expresar en cálculo relacional, y viceversa. La **transformación** de consultas entre álgebra relacional y cálculo relacional es un proceso clave en la teoría de bases de datos, ya que permite la optimización de consultas y la traducción de lenguajes de alto nivel (como SQL) a operaciones más eficientes.

Proceso de Transformación:

1. **De Álgebra a Cálculo Relacional**: Las consultas en álgebra relacional, que son procedimentales, pueden transformarse en consultas declarativas en cálculo relacional simplemente especificando qué condiciones deben cumplir las tuplas o valores. Por ejemplo, una operación de **selección** en álgebra relacional puede transformarse en una fórmula lógica en cálculo relacional.
    - **Ejemplo de Selección**: Si tenemos una consulta en álgebra relacional que selecciona empleados del departamento "Ventas" (`σ_Departamento='Ventas'(Empleado)`), en cálculo relacional orientado a tuplas sería: `{t | t ∈ Empleado ∧ t.Departamento = 'Ventas'}`.
2. **De Cálculo Relacional a Álgebra**: Las consultas declarativas en cálculo relacional pueden transformarse en consultas procedimentales en álgebra relacional especificando las operaciones que deben realizarse para obtener el resultado. Por ejemplo, una fórmula lógica en cálculo relacional puede convertirse en una secuencia de operaciones de selección, proyección, etc.
    - **Ejemplo de Proyección**: Si tenemos una consulta en cálculo relacional orientado a dominios que obtiene los nombres de empleados en el departamento "Ventas" (`{<n> | ∃ d ∃ s (Empleado(n, d, s) ∧ d = 'Ventas')}`), en álgebra relacional se transformaría en: `π_Nombre(σ_Departamento='Ventas'(Empleado))`.

Optimización de Consultas

Una de las ventajas de transformar consultas entre álgebra relacional y cálculo relacional es la capacidad de optimizar las consultas. Por ejemplo, en álgebra relacional es más fácil reorganizar y combinar operaciones (como mover una selección antes de un join para

reducir el número de tuplas procesadas), lo que puede mejorar significativamente el rendimiento de la consulta.

Resumen

El **cálculo relacional** ofrece un enfoque declarativo para formular consultas sobre bases de datos relacionales, ya sea centrándose en los valores de los atributos (**cálculo orientado a dominios**) o en las tuplas completas (**cálculo orientado a tuplas**). Además, la equivalencia entre álgebra relacional y cálculo relacional permite transformar las consultas entre ambos lenguajes, lo que facilita la optimización y la expresión flexible de consultas sobre bases de datos.

1.3. Lenguajes comerciales: SQL (Structured Query Language), QBE (Query By Example):

Los **lenguajes comerciales** se utilizan en sistemas de bases de datos relacionales para interactuar con los datos. Los dos lenguajes más conocidos y utilizados son **SQL (Structured Query Language)** y **QBE (Query By Example)**. Estos lenguajes facilitan la creación, manipulación, y consulta de bases de datos relacionales de manera eficiente. En esta sección, nos enfocaremos principalmente en **SQL**, dado que es el lenguaje más difundido en la industria, aunque también mencionaremos **QBE** como una alternativa visual para consultas.

1.3.1. Orígenes y evolución del SQL.

SQL (Structured Query Language) es un lenguaje estándar para interactuar con bases de datos relacionales. Fue desarrollado en la década de 1970 por **IBM** como parte del proyecto **System R**, que tenía como objetivo demostrar la viabilidad de las bases de datos relacionales basadas en el modelo relacional propuesto por el matemático Edgar F. Codd en 1970. SQL fue inicialmente llamado **SEQUEL** (Structured English Query Language), pero luego fue renombrado como SQL debido a problemas de marca registrada.

Etapas en la Evolución del SQL:

1. **Inicios en System R (1970s)**: IBM desarrolló SQL como parte de su proyecto de investigación System R, que implementaba un sistema de bases de datos relacionales basado en el modelo relacional de Codd.
2. **Adopción por Oracle (1979)**: Oracle fue la primera compañía en comercializar una base de datos que utilizaba SQL, lo que ayudó a SQL a ganar popularidad.
3. **Estándar ANSI (1986)**: En 1986, SQL fue adoptado como estándar por el **American National Standards Institute (ANSI)** y más tarde por la **International Organization for Standardization (ISO)** en 1987, lo que estableció su uso generalizado en la industria.
4. **Evolución Continua**: A lo largo de los años, SQL ha evolucionado con varias revisiones y expansiones, como:
    - **SQL-92**: Introdujo nuevas características como subconsultas y un mayor soporte de integridad referencial.
    - **SQL:1999**: Añadió características de programación procedural, como cursores y triggers.
    - **SQL:2003**: Introdujo soporte para XML y tipos de datos de gran tamaño.
    - **SQL:2016**: Añadió soporte para JSON y mejoras en la analítica.

1.3.2. Características del SQL.

SQL es un lenguaje que combina un enfoque declarativo y procedimental para interactuar con bases de datos relacionales. Entre sus principales características destacan:

1. **Lenguaje Declarativo**: SQL permite describir **qué** resultados se quieren obtener sin especificar **cómo** se deben obtener. Por ejemplo, una consulta SQL define qué datos deben extraerse de una base de datos, pero no el método exacto que el sistema debe seguir para obtener esos datos.
2. **Versatilidad de Consultas**: SQL soporta diversos tipos de operaciones sobre bases de datos, que se agrupan en varias categorías:
    - **Consultas de selección de datos (SELECT)**: Permiten extraer información específica de la base de datos.
    - **Manipulación de datos (INSERT, UPDATE, DELETE)**: Permiten insertar, modificar o eliminar datos de la base de datos.
    - **Definición de datos (CREATE, ALTER, DROP)**: Permiten crear y modificar las estructuras de las tablas, índices y otros objetos de la base de datos.
    - **Control de acceso (GRANT, REVOKE)**: Gestiona los permisos de los usuarios para realizar operaciones en la base de datos.
3. **Integridad de Datos**: SQL proporciona mecanismos para mantener la integridad de los datos mediante la implementación de **restricciones** como claves primarias, claves foráneas, restricciones de unicidad y comprobaciones de datos.
4. **Portabilidad y Estándar**: SQL es compatible con la mayoría de los **Sistemas de Gestión de Bases de Datos Relacionales (RDBMS)**, lo que lo convierte en un estándar altamente **portátil**. Aunque diferentes sistemas tienen extensiones

específicas, las operaciones básicas de SQL son compatibles en cualquier sistema que implemente el estándar.

5. **Soporte para Procedimientos Almacenados**: En versiones más avanzadas, SQL soporta la programación procedimental mediante **procedimientos almacenados**, **triggers** y **funciones**, lo que permite a los desarrolladores encapsular lógica de negocio directamente en la base de datos.
6. **Operaciones Transaccionales**: SQL tiene un robusto soporte para transacciones, lo que permite a los usuarios realizar un conjunto de operaciones como una unidad indivisible, garantizando la integridad y consistencia de los datos en caso de fallos. Las transacciones soportan las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento y Durabilidad).

1.3.3. Sistemas de Gestión de bases de datos con soporte SQL.

La mayoría de los **Sistemas de Gestión de Bases de Datos Relacionales (RDBMS)** modernos implementan SQL como su lenguaje principal de interacción. Entre los sistemas más conocidos que soportan SQL se encuentran:

1. **Oracle Database**: Uno de los sistemas de bases de datos más robustos y utilizados en grandes empresas. Oracle ha implementado muchas extensiones propietarias a SQL, lo que lo hace muy versátil en entornos empresariales críticos.
2. **MySQL**: Un sistema de bases de datos relacional muy popular por ser **open-source**. Es utilizado ampliamente en aplicaciones web, y es conocido por su simplicidad y eficiencia.
3. **PostgreSQL**: Un sistema de bases de datos **open-source** que se destaca por su robustez, conformidad con estándares y extensibilidad. PostgreSQL sigue el estándar SQL de manera rigurosa, aunque también ofrece características avanzadas como soporte para JSON y manejo de datos no estructurados.
4. **Microsoft SQL Server**: Un sistema de bases de datos desarrollado por Microsoft, ampliamente utilizado en el entorno empresarial. Tiene una fuerte integración con los productos de Microsoft y proporciona un entorno robusto y seguro para aplicaciones críticas.
5. **SQLite**: Una base de datos SQL ligera y embebida que no requiere un servidor independiente. Es utilizada en aplicaciones móviles y de escritorio debido a su simplicidad y eficiencia.
6. **MariaDB**: Una bifurcación de MySQL creada por los desarrolladores originales de MySQL, que sigue siendo **open-source** y proporciona mejoras de rendimiento y características adicionales.

QBE (Query By Example)

Query By Example (QBE) es un lenguaje visual desarrollado por IBM en los años 70 que permite a los usuarios realizar consultas a una base de datos mediante una interfaz gráfica. En lugar de escribir una consulta en SQL, el usuario rellena un formulario o tabla con ejemplos de los datos que está buscando. El sistema luego genera la consulta SQL correspondiente en el backend.

Características del QBE:

- **Facilidad de Uso**: QBE es ideal para usuarios que no están familiarizados con SQL, ya que no requiere conocimientos de sintaxis de SQL. Las consultas se crean mediante una interfaz gráfica, lo que lo hace más accesible para usuarios no técnicos.
- **Generación Automática de Consultas**: A partir de la entrada proporcionada por el usuario en la interfaz gráfica, el sistema genera automáticamente una consulta SQL que realiza la acción requerida.

QBE es utilizado en algunas herramientas de bases de datos modernas como **Microsoft Access**, proporcionando una alternativa visual y más intuitiva para consultas simples.

Resumen

Los **lenguajes comerciales** como SQL y QBE proporcionan formas robustas y accesibles para interactuar con bases de datos relacionales. SQL, siendo el lenguaje estándar para la mayoría de los RDBMS, ha evolucionado para incluir características avanzadas y mejorar la portabilidad, integridad de datos, y versatilidad de las consultas. QBE, por otro lado, ofrece una opción visual que simplifica la creación de consultas para usuarios no técnicos, proporcionando una experiencia más intuitiva sin requerir la escritura de código.