




ANÁLISIS Y DISEÑO LÓGICO DE SISTEMAS

Octubre 2023

Luis E.Canales C.
lcanales@utalca.cl

Resumen

- Revisión
 - **Queries:** Select, From, Where
 - **Joins** 
- Otros Join
 - Outer Joins
 - Self Joins
- Implementación de Join

JOINS

- Las claves foráneas describen una relación entre tablas
- Join realizan combinaciones de datos
 - A veces los datos se combinan utilizando una clave foránea, pero no siempre.

Los Joins se parecen a los ciclos anidados

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S Join Registros AS R
ON S.UserID = R.UserID
```

```
For each row1 in Sueldos:
  For each row2 in Registros:
    If row1.userID == row2.userID
      output(row1.nombre, row2.car)
```

Sueldos

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

Registros

UserID	Car
123	Charger
567	Civic
567	Pinto
007	Aston Martin

S.Nombre	R.Car
Juan	Charger
Magda	Civic
Magda	Pinto

Inner Join Sintaxis

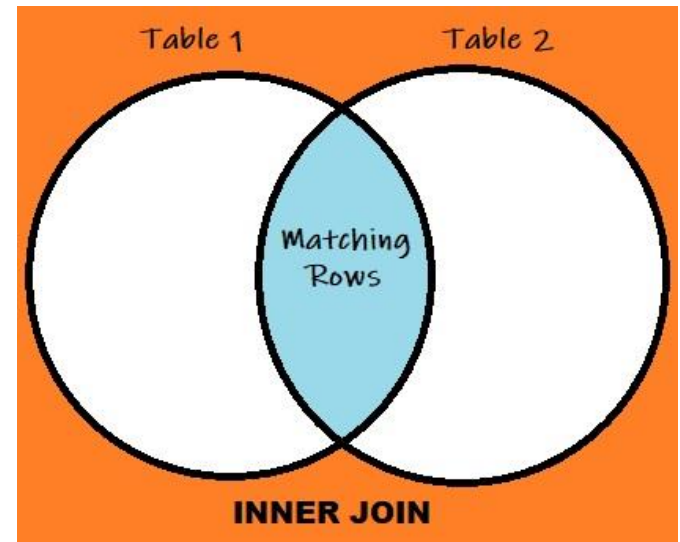
Explícito

```
SELECT S.Nombre, R.Car
  FROM Sueldos AS S
    INNER JOIN Registros AS R
  ON S.UserID = R.UserID
```


Implícito

```
SELECT S.Nombre, R.Car
  FROM Sueldos AS S, Registros AS R
 WHERE S.UserID = R.UserID
```

```
For each row1 in Sueldos:
  For each row2 in Registros:
    If row1.userID == row2.userID
      output(row1.nombre, row2.car)
```



Resumen

- Revisión
 - **Queries:** Select, From, Where
 - Joins
- Otros Join
 - Outer Joins 
 - Self Joins
- Implementación de Join

OUTER JOINS

- Ahora quiero incluir los nombres de todos, incluso si no tienen un auto.

UserID	Nombre	Trabajo	Salario	UserID	Car
123	Juan	Contador	500000	123	Charger
345	Aline	Contador	600000	567	Civic
567	Magda	Profesora	900000	567	Pinto
789	Diana	Profesora	1000000	007	Aston Martin

Nombre	Car
Juan	Charger
Magda	Civic
Magda	Pinto

VS

Nombre	Car
Juan	Charger
Aline	??
Magda	Civic
Magda	Pinto
Diana	??

OUTER JOINS

- Ahora quiero incluir los nombres de todos, incluso si no tienen un auto.

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

UserID	Car
123	Charger
567	Civic
567	Pinto
007	Aston Martin

```
SELECT S.Nombre, R.Car
  FROM Sueldos AS S, Registros AS R
 WHERE S.UserID = R.UserID
```

```
For each row1 in Sueldos:
  For each row2 in Registros:
    If row1.userID == row2.userID
      output(row1.name, row2.car)
```

Nuestro INNER JOIN asume implícitamente que ambos userIDs no son nulos

De forma análoga, nuestro SQL asumía que había una fila en ambas tablas

OUTER JOINS

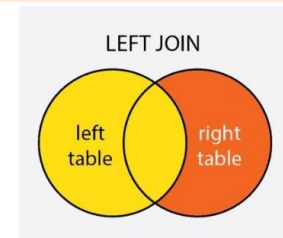
- Ahora quiero incluir los nombres de todos, incluso si no tienen un auto.

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

UserID	Car
123	Charger
567	Civic
567	Pinto
007	Aston Martin

```
For each row1 in Sueldos:
    Boolean found= false;
    For each row2 in Registros:
        if row1.userID == row2.userID
            output(row1.nombre, row2.car)
            found= true;
    if !found
        output (row1.nombre, NULL);
```

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S
LEFT OUTER JOIN Registros AS R
ON S.UserID = R.UserID
```



OUTER JOINS

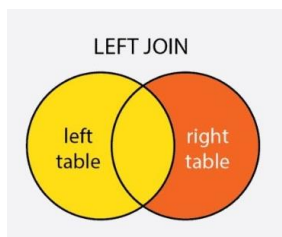
- Ahora quiero incluir los nombres de todos, incluso si no tienen un auto.

Nombre	Car
Juan	Charger
Aline	??
Magda	Civic
Magda	Pinto
Diana	??

NULL es el valor marcador de posición de SQL.

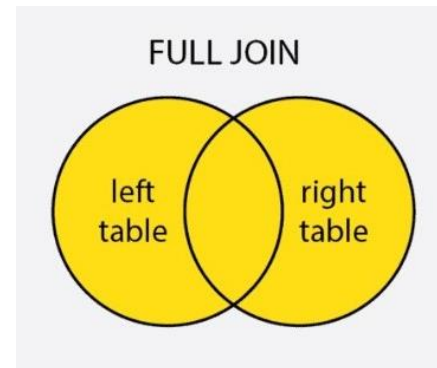
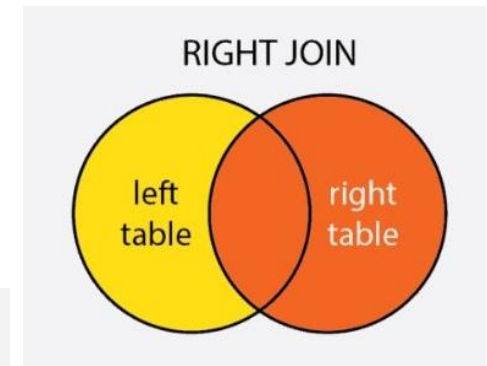
Dependiendo de sus datos, usted puede interpretarlo como desconocido, no aplicable, etc.

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S
LEFT OUTER JOIN Registros AS R
ON S.UserID = R.UserID
```




Diferentes OUTER JOINS

- LEFT OUTER JOIN
 - Se conservan todas las filas de la tabla de la izquierda
- RIGHT OUTER JOIN
 - Se conservan todas las filas de la tabla derecha
- FULL OUTER JOIN
 - Se conservan todas las filas



Resumen

- Revisión
 - **Queries:** Select, From, Where
 - Joins
- Otros Join
 - Outer Joins
 - Self Joins 
- Implementación de Join

Porqué Self Joins?

- Supongamos que queremos encontrar a todos los que conducen un Civic y un Ferrari. ¿Cómo será esta consulta.

Sueldos

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

Registros

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registros AS R
WHERE S.UserID = R.UserID AND
        R.Car = 'Civic' AND
        R.Car = 'Ferrari'
```

Porqué Self Joins?

- Supongamos que queremos encontrar a todos los que conducen un Civic y un Ferrari. ¿Cómo será esta consulta.

Sueldos

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

Registros

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

- No, devuelve el conjunto vacío
 - Recuerde la semántica **for each** (es decir, por fila) de SQL.



```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registro AS R
WHERE S.UserID = R.UserID AND
      R.Car = 'Civic' AND
      R.Car = 'Ferrari'
```

Porqué Self Joins?

- Supongamos que queremos encontrar a todos los que conducen un Civic y un Ferrari. ¿Cómo será esta consulta.

Sueldos

UserID	Nombre	Trabajo	Salario	
123	Juan	Contador	500000	Charger
345	Aline	Contador	600000	NULL
567	Magda	Profesora	900000	Civic
567	Magda	Profesora	900000	Ferrari
789	Diana	Profesora	1000000	NULL

```
For each s in Sueldos:
  For each r in Registros:
    If s.userID == r.userID and
       r.car == 'Civic' and
       r.car == 'Ferrari'
      output(r.nombre, p.car)
```

Self Joins = todos los pares (..o todos los triples, o ...)

UserID	Car
123	Charger
567	Civic
567	Ferrari

UserID	Car
123	Charger
567	Civic
567	Ferrari

```
SELECT *  
  FROM Registros AS R1,  
       Registros AS R2;
```

R1.UserID	R1.Car	R1.UserID	R1.Car
123	Charger	123	Charger
123	Charger	567	Civic
123	Charger	567	Ferrari
567	Civic	123	Charger
567	Civic	567	Civic
567	Civic	567	Ferrari
567	Ferrari	123	Charger
567	Ferrari	567	Civic
567	Ferrari	567	Ferrari

Ejemplo: todos los pares

UserID	Car
123	Charger
567	Civic
567	Ferrari

UserID	Car
123	Charger
567	Civic
567	Ferrari

Todos los pares de autos propiedad de una persona

```
SELECT R1.Car, R2.Car
FROM Registros AS R1,
     Registros AS R2
WHERE R1.UserID = R2.UserID;
```

R1.UserID	R1.Car	R2.UserID	R2.Car
123	Charger	123	Charger
123	Charger	567	Civic
123	Charger	567	Ferrari
567	Civic	123	Charger
567	Civic	567	Civic
567	Civic	567	Ferrari
567	Ferrari	123	Charger
567	Ferrari	567	Civic
567	Ferrari	567	Ferrari

Ejemplo: todos los pares

UserID	Car
123	Charger
567	Civic
567	Ferrari

UserID	Car
123	Charger
567	Civic
567	Ferrari

Todos los pares de autos propiedad de una persona (excluye duplicados)

```
SELECT R1.Car, R2.Car
FROM Registros AS R1,
     Registros AS R2
WHERE R1.UserID = R2.UserID AND
      R1.Car != R2.Car;
```

R1.UserID	R1.Car	R2.UserID	R2.Car
123	Charger	123	Charger
123	Charger	567	Civic
123	Charger	567	Ferrari
567	Civic	123	Charger
567	Civic	567	Civic
567	Civic	567	Ferrari
567	Ferrari	123	Charger
567	Ferrari	567	Civic
567	Ferrari	567	Ferrari

JOIN y SELF - JOIN

- Una consulta SQL puede tener varias relaciones en su cláusula FROM:

```
...  
FROM Registros AS R,  
      Sueldos AS S  
...
```

- Cuando una relación aparece más de una vez, la llamamos a esa consulta un **self join**

```
...  
FROM Registros AS R1,  
      Registros AS R2,  
      Sueldos   AS S  
...
```

JOIN y SELF - JOIN

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

- a todos los que conducen un Civic y un Ferrari.

Todos los pares de autos propiedad de una sola persona (excluye los duplicados)

```
SELECT R1.Car, R2.Car
FROM Registros AS R1,
     Registros AS R2
WHERE R1.UserID = R2.UserID AND
     R1.Car != R2.Car;
```

Todas las personas propietarios de un Civic

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S,
     Registros AS R
WHERE S.UserID= R.UserID AND
     R.Car= 'Civic';
```

JOIN y SELF - JOIN

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

P.nombre	R1.UserID	R1.Car	R2.UserID	R2.Car
Magda	567	Civic	567	Ferrari
Magda	567	Ferrari	567	Civic

```
SELECT P.Nombre
FROM Registros AS R1,
     Registros AS R2,
     Sueldos AS S
WHERE R1.UserID = S.UserID AND
      R2.UserID = S.UserID AND
      R1.Car != R2.Car;
```

JOIN y SELF - JOIN

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000


UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

UserID	Car
123	Charger
567	Civic
567	Ferrari
007	Aston Martin

P.nombre	R1.UserID	R1.Car	R2.UserID	R2.Car
Magda	567	Civic	567	Ferrari

```
SELECT S.Nombre
FROM   Registros AS R1,
       Registros AS R2,
       Sueldfos  AS S
WHERE  R1.UserID = S.UserID AND
       R2.UserID = S.UserID AND
       R1.Car = 'Civic' AND
       R2.Car = 'Ferrari';
```

Resumen

- Revisión
 - **Queries:** Select, From, Where
 - Joins
- Otros Join
 - Outer Joins
 - Self Joins
- Implementación de Join 

Discusión

- Supongamos **k** relaciones, cada una de ellas de tamaño **n**.
- ¿Cuál es la complejidad de la semántica de los bucles anidados?

```
...  
FROM Rel1 AS R1,  
      Rel2 AS R2,  
      ...  
      Relk AS Rk  
...
```


Discusión

- Supongamos **k** relaciones, cada una de ellas de tamaño **n**.
- ¿Cuál es la complejidad de la semántica de los bucles anidados?

```
...  
FROM Rel1 AS R1,  
      Rel2 AS R2,  
      ...  
      Relk AS Rk  
...
```



$O(n^k)$

- Las bases de datos **NO** calculan las consultas utilizando la semántica de bucle

Posible optimización

- El orden de los bucles **for each** no importa.

```
For each p in Sueldos:  
  For each r in Registros:  
    ...
```

```
For each r in Registros:  
  For each p in Sueldos:  
    ...
```

- Podemos comprobar los predicados antes de tiempo:
 - Aplicar el filtro **r.Car = 'Ferrari'** antes de recorrer el bucle Sueldos
 - ... ¡o antes de hacer un segundo bucle sobre Registros!
- La base de datos podría recuperar filas con **r.Car = 'Ferrari'** sin iterar sobre todas las filas
 - Conocido como un **índice** (index), cubrirá en varias semanas.

Independencia de los datos físicos

- SQL es un lenguaje declarativo
 - Lo que queremos, no cómo conseguirlo
- El optimizador de consultas es responsable de elegir el mejor plan de ejecución para la consulta
- Esto es la **independencia física de los datos**.

Outer JOIN: Reflexiones finales

- **Outer joins** tienen menos potencial de optimización:
 - utilizar con moderación
- Caso típico de **LEFT OUTER JOIN**: una relación de 1 a muchos
- relaciones:
 - Una empresa fabrica 0 o más productos
 - Un estudiante realiza 0 o más cursos
 - Un cliente hace 0 o más pedidos
- No conozco ninguna buena razón para utilizar un RIGHT OUTER JOIN o un FULL OUTER JOIN.
 - Algunos SADB ni siquiera los soporta.

Un poco más de SQL

UserID	Nombre	Trabajo	Salario
123	Juan	Contador	500000
345	Aline	Contador	600000
567	Magda	Profesora	900000
789	Diana	Profesora	1000000

- **ORDER BY**

- Ordena las tuplas resultantes por los atributos especificados (Ascendente predeterminado)

```
SELECT S.Nombre, S.UserI  
FROM Sueldos AS S  
WHERE S.Trabajo = 'Contador'  
ORDER BY S.Salario, S.Nombre ASC|DESC;
```

Nombre	UserID
Juan	123
Aline	345

- **DISTINCT**

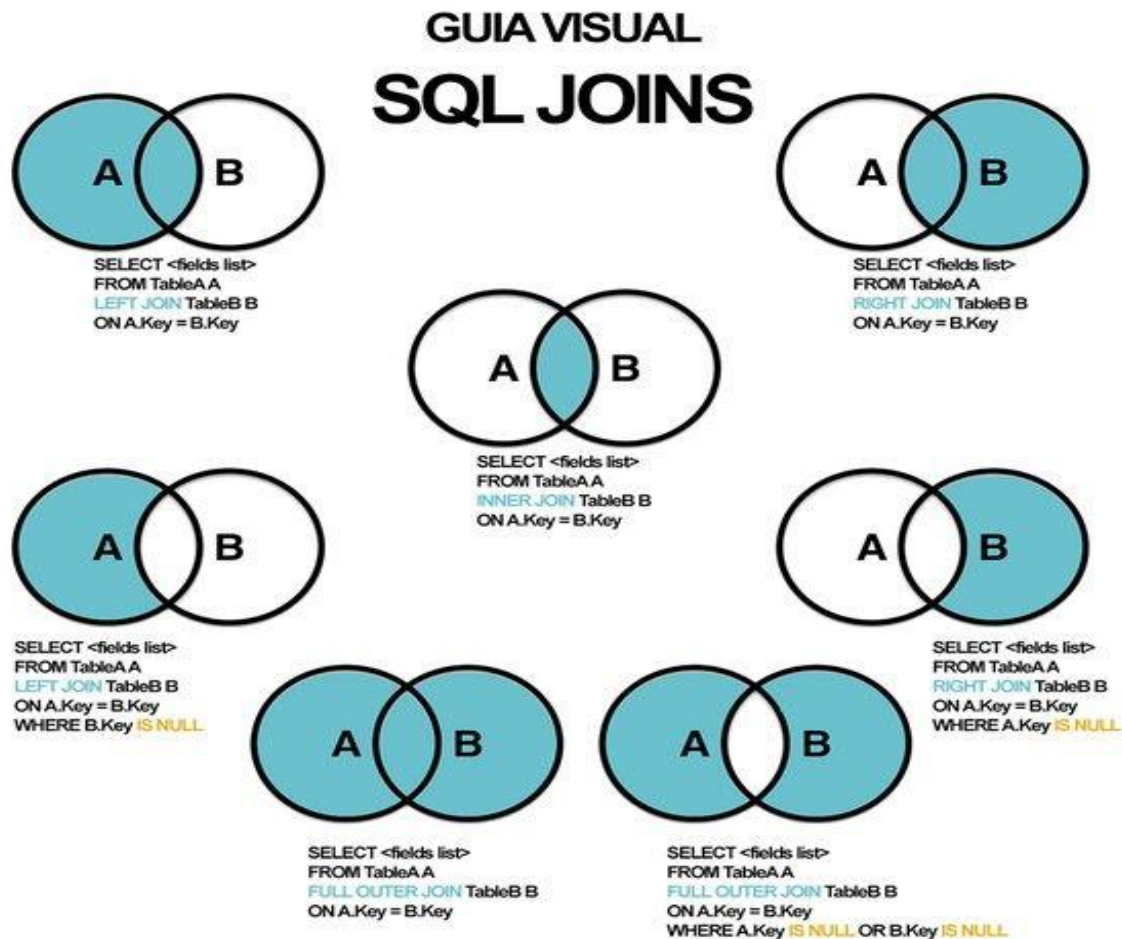
- Deduplica los resultados de las tuplas

```
SELECT DISTINCT S.Trabajo  
FROM Sueldos AS S  
WHERE S.Salario > 450000;
```

Trabajo
Contador
Profesora

Resumen - Join

- Use join para combinar datos de diferentes tablas.



Resumen - Join

- Use join para combinar datos de diferentes tablas.
 - Semántica de Loops anidados
 - Semántica de producto cruzado filtrado
 - Inner join (el más común)
 - Outer Join pueden conservar la información

