# TEMA 9 PARTE I.
# LENGUAJE DE CONSULTA DE DATOS (DQL)

## 9.1. LA INSTRUCCIÓN SELECT

Como su nombre lo indica, esta instrucción se emplea para obtener datos almacenados por el RDBMS (objetos). Algunos autores clasifican a esta instrucción como una categoría más del lenguaje SQL: Data Query Language (DQL).

En realidad, la sintaxis de la instrucción es sencilla, sin embargo, rara vez se emplea su forma simple. Generalmente una consulta requiere hacer uso de condiciones, joins, agrupación de datos, etc., todas estas operaciones empleadas para filtrar y personalizar la lista de resultados a obtener.

### *9.1.1. Sintaxis SQL estándar:*

```
select [distinct]
{
  [<qualifier>.]<column-name>| * |<expression>
  | <pseudocolumn> as <column-alias>
},...
from
{
 <table-or-view-name> | <inline-view> | [[as] <table-alias>]
}
[where <predicate>]
[group by [<qualifier>.]<column-name>,...
 [having <predicate>]
]
[order by {<column-name>|<column-number>}
 [asc|desc]
];
```

### *9.1.2. Selección de una sola columna.*

*Ejemplo:*

Seleccionar las claves de los puestos existentes.

```
select clave_puesto
from puesto;

CLAVE_PUESTO
============
DG
JD
```

### *9.1.3. Selección de múltiples columnas:*

*Ejemplo:*

Seleccionar el nombre completo de todos los empleados que nacieron a partir del año 1983

```
select nombre, apellido_paterno,apellido_materno
from empleado
where fecha_nacimiento >= to_date('01/01/1983','DD/MM/YYYY');
```

```
NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO
====== ================ ================
ANGEL  JUAREZ           AGUIRRE
```

### *9.1.4. Selección de todas las columnas de una tabla.*

*Ejemplo:*

Seleccionar todos los datos de las quincenas del año 2010

```
select *
from quincena
where fecha_inicio >=to_date('01/01/2010','dd/mm/yyyy')
and fecha_fin <= to_date('31/12/2010','DD/MM/YYYY');
```

```
QUINCENA_ID NUMERO_QUINCENA FECHA_INICIO        FECHA_FIN
=========== =============== ================== ==================
1           1               2010-01-01 00:00:00 2010-01-15 00:00:00
2           2               2010-01-16 00:00:00 2010-01-31 00:00:00
3           3               2010-02-01 00:00:00 2010-02-15 00:00:00
4           4               2010-02-16 00:00:00 2010-02-28 00:00:00
```

### *9.1.5. Distinct, unique, all*

La cláusula `distinct` se emplea para eliminar información duplicada del conjunto de datos obtenido al ejecutar la sentencia `select`. La sentencia `distinct` descarta ***registros*** duplicados. Es decir, si en la matriz de datos se detectan 2 registros con valores de sus columnas idénticos, solo se muestra uno de ellos, descartando a todos los demás.

*Ejemplo:*

Considere la siguiente lista de empleados:

```
select empleado_id,nombre,apellido_paterno,
   apellido_materno, fecha_nacimiento
from empleado;
```

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO FECHA_NACIMIENTO
=========== ====== ================ ================ ==================
2           ANGELA RAMIREZ          LUNA             1980-01-11 00:00:00
1           JUAN   MARTINEZ         LOPEZ            1980-01-10 10:40:00
5           ANGEL  JUAREZ           AGUIRRE          1983-06-10 10:40:00
6           MARIO  JUAREZ           LOPEZ            1986-05-09 16:38:00
```

Mostrar un listado de todos los apellidos paternos de los empleados:

```
select apellido_paterno
from empleado;
```

```
APELLIDO_PATERNO
================
RAMIREZ
MARTINEZ
JUAREZ
JUAREZ
```

Eliminando Duplicados:

```
select distinct apellido_paterno
from empleado;

APELLIDO_PATERNO
===============
JUAREZ
RAMIREZ
MARTINEZ
```

#### *9.1.5.1. Unique*

La cláusula `unique` es un sinónimo de `distinct`. Se pueden emplear de forma indistinta, aunque `distinct` es mucho más popular.

#### *9.1.5.2. All*

La cláusula `all` es la opción por `default`, es decir, se obtienen todos los registros. En la práctica no se usa `all` ya que es la opción por default, por lo que no se requiere especificarla en la cláusula `select`.

### ***9.1.6. Order by***

Se emplea para realizar el ordenamiento de los registros obtenidos de una consulta.

*Sintaxis:*

```
select <column-list>
from <table-name>
[where <condition-list>]
[order by <column-list>[asc| desc]]
```

- Si no se especifica `asc` (ascendente) o `desc` (descendente), el default es `asc`
- Order by es la última cláusula que puede aparecer en una sentencia `select`.
- Si se especifica más de una columna, el ordenamiento se realiza considerando la primer columna, después la segunda y así sucesivamente.

*Ejemplo:*

```
select *
from estudiante
order by apellido_paterno,apellido_materno,nombre;
```

- Las columnas que se emplean en la cláusua order by no necesariamente tienen que aparecer en la lista de columnas de la cláusula `select`.

*Elemplo:*

```
select  estudiante_id
from estudiante
order by apellido_paterno,apellido_materno,nombre;
```

- En el valor de `<column_list>` se pueden emplear números en lugar del nombre. Cada número corresponde con el número de columna que se especificó en la cláusula `select`.

*Elemplo:*

```
select  nombre,apellido_paterno,apellido_materno, estudiante_id
from estudiante
order by 2,3,1;
```

En este caso los registros se ordenan por apellido paterno, después por apellido materno, y finalmente por nombre.

### *9.1.7. Literales, funciones y columnas calculadas.*

La instrucción `select` no se emplea únicamente para seleccionar columnas de una tabla, por lo que dicha instrucción no siempre vendrá acompañada de una tabla. La siguiente sección ilustra este escenario.

#### *9.1.7.1. Tablas dummy*

Se emplean principalmente cuando la sentencia `select` obtiene algún resultado que no existe almacenado, si no que este se obtiene al vuelo, tal vez proporcionado por una función o por algún cálculo en donde no requiere la asociación con alguna tabla.

Sin embargo, a pesar de que el resultado no involucre el uso de una tabla, es necesario indicar la sentencia `from` para completar la instrucción `select`. Es en este caso donde se emplean las llamadas "tablas dummy".

*Ejemplo:*
Para calcular la fecha actual del sistema no se requiere acceder a los datos de a alguna tabla. Cada manejador implementa este concepto de formas distintas:

Oracle: Se emplea la tabla dummy `dual`.

```
select sysdate from dual;

SYSDATE
===================
2010-11-15 18:47:50

select (5+5) from dual;

(5+5)
=====
10

select cos(3.1416/2) from dual;

COS(3.1416/2)
========================================
-.00000367320510337250859767736713696354
```

DB2: Se emplea la tabla dummy `system.sysdummy1`

```
select (5+5) from sysibm.sysdummy1;
```

### *9.1.8. Alias en columnas.*

Un alias permite proporcionar una forma distinta de nombrar a una columna únicamente para ser presentada como parte de la respuesta de la instrucción SELECT.

*Ejemplo:*
Retomando la sentencia que muestra la suma de 2 números, observar que el nombre de la columna resultante es "(5+5)"

```
(5+5)
=====
10
```

En lugar de esto, sería más adecuado renombrar el nombre de la columna resultante por un valor más descriptivo:

```
select (5+5) as suma from dual;
SUMA
====
10
```

En oracle, la palabra "AS" es opcional, es decir, la siguiente sentencia es equivalente:

```
select (5+5) suma from dual;
```

### *9.1.9. Alias en tablas*

De manera similar, los alias a nivel de tabla se emplean para nombrar o señalar a una tabla empleando algún otro nombre.
Los alias de tablas se emplean principalmente para las sentencias `select` que requieren hacer uso de joins con otras tablas (se revisa más adelante este tema). En este caso normalmente las tablas se representan con unos cuantos caracteres para evitar escribir el nombre completo de la tabla.

*Ejemplo:*
Mostrar los datos del empleado `ANGELA RAMIREZ LUNA` y los nombres de sus hijos.

```
select  e.nombre, e.apellido_paterno, e.apellido_materno, h.nombre,
    h.apellido_paterno, h.apellido_materno
from  empleado e,  hijo_empleado h
where  e.empleado_id = h.empleado_id
and e.nombre='ANGELA'
and e.apellido_paterno='RAMIREZ'
and e.apellido_materno='LUNA';
```

Observar que se emplean los alias "e" y "h" para hacer referencia a la tabla `empleado` e `hijo_empleado`. De otra forma, en todos los lugares donde aparece E y H se tendría que sustituir el valor completo del nombre de la tabla.

Cuando en una sentencia `select` intervienen más de una tabla, y si ambas tablas contienen columnas con el mismo nombre, (como en el ejemplo anterior), es necesario indicar en la sentencia `select` el nombre de la tabla del campo que queremos seleccionar empleando la sintaxis

`<nombre-tabla-o-alias>.<nombre-campo>`

Si no se especifica lo anterior, el manejador generará un error de campos ambiguos, ya que este no sabe a cuál de las 2 tablas referirse.

Ejemplo: `empleado.nombre,hijo_empleado.nombre`

Lo anterior es mejor si usamos un alias: `e. nombre, h.nombre`

## 9.2. ÁLGEBRA RELACIONAL.

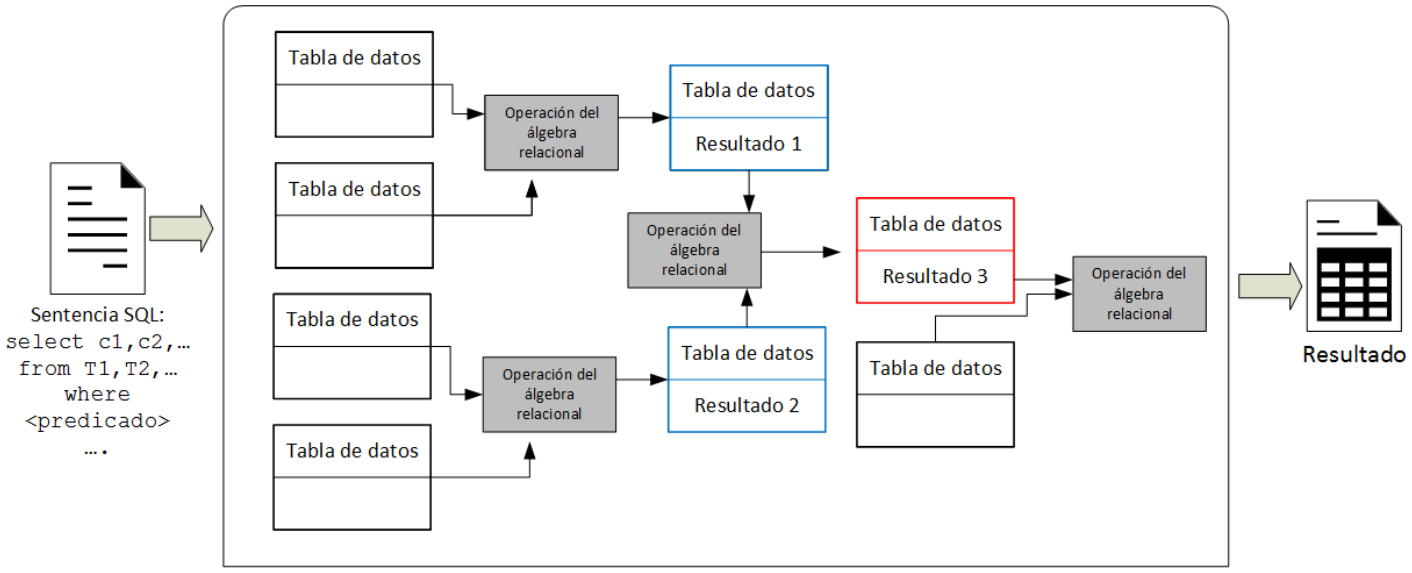
El álgebra relacional es un lenguaje formal integrado principalmente por una serie de ***operadores*** unarios y binarios que se aplican sobre una o varias relaciones.

Recordando de temas anteriores, una relación en términos de base de datos corresponde a una tabla, o a nivel más general, a un conjunto dentro del contexto de la ***teoría de conjuntos***.

El resultado de aplicar estos operadores a dichas relaciones, obtiene como resultado una nueva ***relación*** sin que se alteren las relaciones originales. Este principio es exactamente el mismo concepto empleado en teoría de conjuntos, es decir, dados 2 conjuntos A y B, cualquier operación que se aplique entre ellos, generará un nuevo conjunto C (***propiedad de cerradura***).

Los RDBMS trabajan de manera similar, la instrucción `select` trabaja con conjuntos de datos. Típicamente cada conjunto de datos se representa a través de una relación en términos del modelo relacional.

El estudio del álgebra relacional es fundamental en el estudio de la instrucción `select`. En la práctica, todas las instrucciones `select` que un usuario envía a un RDBMS para ser ejecutadas se procesan a través de la descomposición de la consulta a un conjunto de operaciones del álgebra relacional como se ilustra en la siguiente figura.



El resultado de aplicar un operador del álgebra relacional a uno o más conjuntos de datos produce un nuevo conjunto de datos que puede ser la entrada para otro operador***: encadenamiento de operaciones***.

Antes de iniciar con la revisión de estos operadores, revisaremos brevemente los principales conceptos de la teoría de conjuntos.

### *9.2.1. Teoría de conjuntos*

Conjunto: Colección de objetos. Cada uno de estos objetos es llamado "elemento del conjunto" y pueden ser de diferentes tipos.

*Ejemplos:*

$$A = \{1,2,3\}, B = \{x \mid x \in N, 1 \leq x \leq 10\}$$

- Típicamente el nombre del conjunto se expresa en mayúsculas y sus elementos en minúsculas.
- El orden de los elementos dentro del conjunto es irrelevante (similar al orden en el que aparecen los registros de una tabla que no usan `order by`). $A = \{1,2,3\}, B = 3,2,1\} \rightarrow A = B$
- Los conjuntos pueden estar formados por otros conjuntos. $A = \{1,2,3,\{5,6,7,8,9\}\}$
- Un conjunto puede ser vacío (sin elementos). $A = \{\emptyset\}, B = \{\emptyset\}$
- En teoría de conjuntos, los elementos duplicados son ignorados. $A = \{1,1,1,2,3,3,3\} = \{1,2,3\}$
- Los conjuntos pueden ser infinitos. $Z = \{\ldots -3, -2, -1, 0, 1, 2, 3 \ldots\}$

### *9.2.2. Operadores relacionales.*

El álgebra relacional define desde un punto de vista teórico, la forma en la que se realiza la manipulación de los datos de una tabla empleando los siguientes operadores conocidos también como *set operators*.

Considerar 2 relaciones R y S, los operadores del álgebra relacional se muestran a continuación:

- Operadores relacionales básicos

| Nombre | Representación | Tipo |
|---|---|---|
| Select | $\sigma_{<predicado>}(R)$ | Unario |
| Project | $\pi_{c1,c2,..}(R)$ | Unario |
| Union | $R \cup S$ | Binario |
| Difference | $R - S$ | Binario |
| Product | $R \times S$ | Binario |

- Operadores compuestos: Se pueden obtener a partir de la combinación de los operadores básicos.

| Nombre | Representación | Tipo |
|---|---|---|
| Join | $R \bowtie_{predicado} S$ | Binario |
| Intersect | $R \cap S$ | Binario |



A ⋈ a.p-i > 900,000 AND sa.clave = REGISTRADO SA

Para ilustrar estos conceptos, suponer que las relaciones R y S contienen los siguientes datos:

R

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 3 | 40 | 200 | Y | True |
| 4 | 60 | 300 | Z | False |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

S

| A | F | G | H | I |
|---|---|---|---|---|
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 3 | 55 | 9 | T | False |
| 4 | 67 | 11 | U | True |
| 5 | 70 | 400 | W | False |

### *9.2.2.1. Operador Select: $\sigma_p(R)$*

Obtiene los valores de todos los registros de la tabla o relación $R$ que satisfacen una condición o predicado $P$. Se representa por $\sigma_p(R)$

*Ejemplo:*

Obtener $\sigma_{a\geq 4}(R)$

| A | B | C | D | E |
|---|---|---|---|---|
| 4 | 60 | 300 | Z | False |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

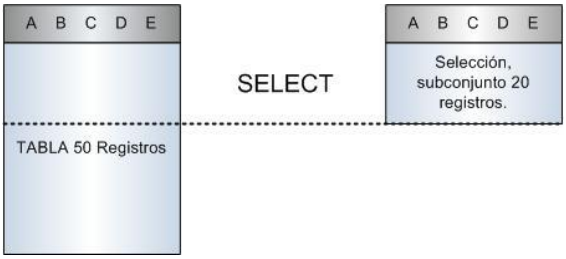
*Ejemplo:*

Seleccionar todas las tuplas que contengan `'Gómez'` como apellido paterno en la relación `estudiante e`.

$$\sigma_{apellido\_paterno='Gomez'}(E)$$

Una condición puede ser una combinación booleana, donde se pueden usar operadores como: $\wedge, \vee,$ combinándolos con operadores $<, >, \geq, \leq, =, != =$

El operador `select` considera la obtención horizontal de un subconjunto de los datos de una tabla.




*Sintaxis SQL:*

```
select {* | <column_name1>,<column_name2>,...,<column_namen>}
from <table_name>
where <condition>;
```

*Ejemplo:*

```
select *
from estudiante
where apellido_paterno='gomez';
```

### 9.2.2.2. Operador Project $\pi_A(R)$

columnas

La operación de proyección obtiene un subconjunto vertical de una tabla o relación (R). Es decir, obtiene todas Las tuplas de R para un subconjunto de sus atributos o columnas especificadas $A_1, A_2, A_3, \ldots, A_n$
Se representa por: $\pi_A(R)$

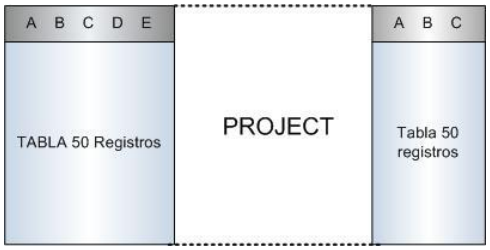
*Ejemplo:*
Obtener $\pi_{D,E}(R)$

| D | E |
|---|---|
| X | True |
| Y | True |
| Z | False |
| W | False |
| M | False |

*Ejemplo:*
Seleccionar el nombre y apellidos de todos los estudiantes.

$$\pi_{nombre,apellido\_paterno,apellido\_materno}(E)$$

La operación de proyección representa un corte vertical de una tabla:




*Sintaxis SQL:*

```
select {<column_name1>,<column_name2>,...}
from <table_name>
```

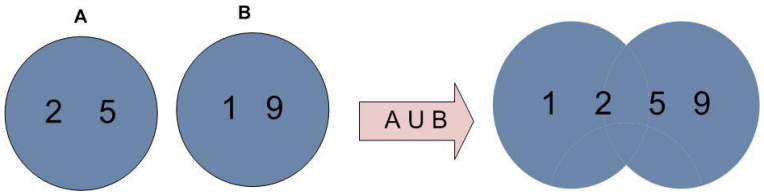
*Ejemplo:*

```
select nombre,apellido_paterno,apellido_materno
from estudiante;
```

### 9.2.2.3. Operador Union $R \cup S$

La unión de 2 relaciones o conjuntos A y B es un nuevo conjunto que contiene todos los elementos de A y de B.

$A = \{2,5\}$
$B = \{1,9\}$
$A \cup B = \{1,2,5,9\}$




- Duplicados se excluyen.

*Ejemplo:*

Obtener $R \cup S$

R

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 3 | 40 | 200 | Y | True |
| 4 | 60 | 300 | Z | False |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

S

| A | F | G | H | I |
|---|---|---|---|---|
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 3 | 55 | 9 | T | False |
| 4 | 67 | 11 | U | True |
| 5 | 70 | 400 | W | False |

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 3 | 40 | 200 | Y | True |
| 4 | 60 | 300 | Z | False |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 3 | 55 | 9 | T | False |
| 4 | 67 | 11 | U | True |

- En el ejemplo, se elimina uno de los 2 registros duplicados:

| 5 | 70 | 400 | W | False |
|---|---|---|---|---|

*Sintaxis SQL:*

- Para realizar la unión, las columnas de ambas tablas deben ser compatibles en cuanto a número de columnas, y deben ser del mismo tipo de dato. Típicamente son conjuntos de la misma tabla, pero con condiciones diferentes.
- Se emplea la palabra `union`.

```
select <column_name1>,<column_name2>,..., from <table_name_1> ...
 union
select <column_name1>,<column_name2>,..., from <table_name_2> ...
```

Si se requiere incluir registros duplicados, se emplea `union all`.

*Ejemplo:*

Considerar el contenido de la tabla `empleado`:

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO FECHA_NACIMIENTO    CONYUGE_EMPLEADO_ID
=========== ====== ================ ================ =================== ===================
2           ANGELA RAMIREZ          LUNA             1980-01-11 00:00:00 3
7           MARIA  AGUILAR          GUZMAN           1980-06-10 10:40:00 {null}
1           JUAN   MARTINEZ         LOPEZ            1980-01-10 10:40:00 {null}
6           MARIO  JUAREZ           LOPEZ            1986-05-09 16:38:00 7
5           ANGEL  JUAREZ           AGUIRRE          1983-06-10 10:40:00 2
```

Suponga que se les desea dar un aumento de sueldo a todos los empleados que hayan nacido durante el mes de enero de 1980 del 3 %. Adicionalmente, a los empleados que sean casados, se les dará un

aumento del 2.4%. Genere una sentencia SQL que muestre un reporte de todos los empleados que tuvieron aumento.

```
select empleado_id,nombre,apellido_paterno,
   apellido_materno,fecha_nacimiento,conyuge_empleado_id
from empleado
where fecha_nacimiento >=to_date('01/01/1980','dd/mm/yyyy')
and fecha_nacimiento <=to_date('31/01/1980','dd/mm/yyyy')
union
select empleado_id,nombre,apellido_paterno,
   apellido_materno,fecha_nacimiento,conyuge_empleado_id
from empleado
where conyuge_empleado_id is not null;
```

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO FECHA_NACIMIENTO    CONYUGE_EMPLEADO_ID
=========== ====== ================ ================ =================== ===================
1           JUAN   MARTINEZ         LOPEZ            1980-01-10 10:40:00 {null}
2           ANGELA RAMIREZ          LUNA             1980-01-11 00:00:00 {null}
5           ANGEL  JUAREZ           AGUIRRE          1983-06-10 10:40:00 2
6           MARIO  JUAREZ           LOPEZ            1986-05-09 16:38:00 7
```

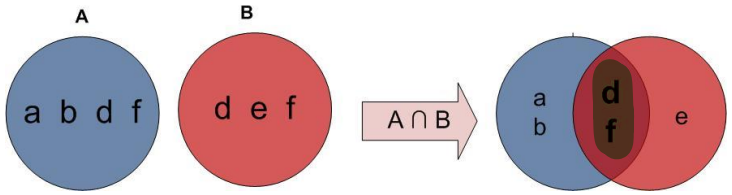
Del resultado anterior se puede observar lo siguiente:

- De la primera sentencia SQL se obtienen los registros con `empleado_id` 2,1
- De la segunda sentencia SQL se obtienen los registros con `empleado_id` 2,6,5
- Al aplicar la operación unión se obtienen los registros con `empleado_id` 1,2,5 y 6. Observar que se elimina uno de los 2 registros con `empleado_id` = 2.
- Observar que los registros aparecen ordenados empleando las columnas de izquierda a derecha. En este caso empleando el atributo `empleado_id`. Este ordenamiento se realiza para poder realizar la detección de registros duplicados.

#### *9.2.2.4. Operador intersect $R \cap S$*

La intersección de 2 conjuntos A y B es el resultado de obtener todos los elementos comunes a A y B.

$A = \{a, b, d, f\}$
$B = \{d, e, f\}$
$A \cap B = \{d, f\}$




- El resultado puede ser obtenido en términos de operadores básicos: $R \cap S = R - (R - S)$

*Ejemplo:*

Obtener $R \cap S$

R S

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 3 | 40 | 200 | Y | True |
| 4 | 60 | 300 | Z | False |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

| A | F | G | H | I |
|---|---|---|---|---|
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 3 | 55 | 9 | T | False |
| 4 | 67 | 11 | U | True |
| 5 | 70 | 400 | W | False |

| A | B | C | D | E |
|---|---|---|---|---|
| 5 | 70 | 400 | W | False |

*Sintaxis SQL:*

- A nivel de base de datos se obtienen los registros que aparecen en ambas tablas con valores idénticos.
- Para realizar la intersección, las columnas de ambas tablas deben ser compatibles en cuanto a número de columnas, y deben ser del mismo tipo de dato. Típicamente son conjuntos de la misma tabla, pero con condiciones diferentes.
- Se emplea la palabra `intersect`

*Sintaxis SQL*:

```
select <column_name1>,<column_name2>,..., from <table_name_1> ...
 intersect
select <column_name1>,<column_name2>,..., from <table_name_2> ...
```

*Ejemplo:*

Considere nuevamente los datos de la tabla `EMPLEADO`:

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO JEFE_INMEDIATO PUESTO_ID
=========== ====== ================ ================ ============== =========
2           ANGELA RAMIREZ          LUNA             1              1
7           MARIA  AGUILAR          GUZMAN           {null}         2
1           JUAN   MARTINEZ         LOPEZ            {null}         1
6           MARIO  JUAREZ           LOPEZ            {null}         2
5           ANGEL  JUAREZ           AGUIRRE          {null}         2
```

Suponga que se requiere obtener un reporte de todos los empleados que no cuentan con jefe inmediato, así como los empleados cuyo puesto sea Jefe de Departamento (JD)

```
select empleado_id,nombre,apellido_paterno,apellido_materno,
    jefe_inmediato,puesto_id
from empleado
where jefe_inmediato is null
intersect
select empleado_id,nombre,apellido_paterno,apellido_materno,
    jefe_inmediato,puesto_id
from empleado
where puesto_id = (select puesto_id from puesto where clave_puesto='jd');
```

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO JEFE_INMEDIATO PUESTO_ID
=========== ====== ================ ================ ============== =========
5           ANGEL  JUAREZ           AGUIRRE          {null}         2
6           MARIO  JUAREZ           LOPEZ            {null}         2
7           MARIA  AGUILAR          GUZMAN           {null}         2
```

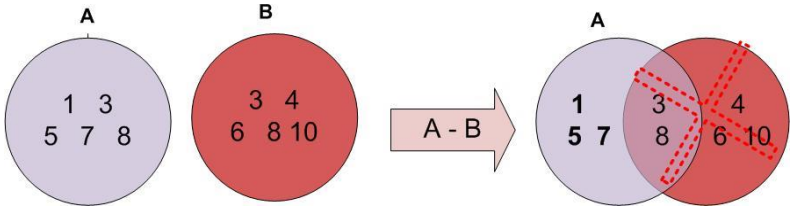
### *9.2.2.5. Operador difference $R - S$*

La diferencia de 2 conjuntos A y B se define como el conjunto A-B que consiste de todos los elementos de A que no existen en B.

$A = \{1,3,5,7,8\}$
$B = \{3,4,6,8,10\}$

$A - B = \{1,5,7\}$
$B - A = \{4,6,10\}$

- Observar que A – B es diferente a B – A




*Ejemplo:*
Obtener $R - S$

R

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 3 | 40 | 200 | Y | True |
| 4 | 60 | 300 | Z | False |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

S

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 3 | 55 | 9 | T | False |
| 4 | 67 | 11 | U | True |
| 5 | 70 | 400 | W | False |

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 3 | 40 | 200 | Y | True |
| 4 | 60 | 300 | Z | False |
| 6 | 80 | 500 | M | False |

*Sintaxis SQL:*

- En base de datos la diferencia produce todos los registros que no son encontrados en la otra tabla.
- Para realizar la diferencia, las columnas de ambas tablas deben ser compatibles en cuanto a número de columnas, y deben ser del mismo tipo de dato. Típicamente son conjuntos de la misma tabla, pero con condiciones diferentes.
- Se emplea la cláusula `minus`

*Sintaxis SQL*:

```
select <column_name1>,<column_name2>,..., from <table_name_1> ...
 minus
select <column_name1>,<column_name2>,..., from <table_name_2> ...
```

*Ejemplo:*

Considere nuevamente la lista de empleados de una empresa.

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO FECHA_NACIMIENTO    PUESTO_ID
=========== ====== ================ ================ =================== =========
1           JUAN   MARTINEZ         LOPEZ            1980-01-10 10:40:00 1
2           ANGELA RAMIREZ          LUNA             1980-01-11 00:00:00 1
7           MARIA  AGUILAR          GUZMAN           1980-06-10 10:40:00 2
5           ANGEL  JUAREZ           AGUIRRE          1983-06-10 10:40:00 2
6           MARIO  JUAREZ           LOPEZ            1986-05-09 16:38:00 2
```

Se desea otorgar un incremento de sueldo del 0.56% a todos los jefes de departamento. Sin embargo, el mes anterior, la empresa otorgó el 0.5% de incremento de sueldo a todos los empleados que nacieron entre los años 1980 y 1985. Genere una sentencia SQL que obtenga a todos los empleados que recibirán aumento, considerando que, si el empleado ya recibió aumento el mes pasado, ya no se le aplicará en esta ocasión.

*Conjunto A*: Lista de empleados que son Jefes de Departamento recibirán aumento. Dentro de este conjunto, pudieran existir empleados que tuvieron aumento el mes pasado, por lo que hay que excluirlos del conjunto A. ¿Quiénes son estos empleados?

*Conjunto B:* Lista de empleados que nacieron entre 1980 y 1985.

Si realizamos A-B, se excluirán todos los empleados del conjunto A que son jefes de departamento, pero que nacieron entre 1980 y 1985, que es justamente el resultado deseado:

```
select empleado_id,nombre,apellido_paterno,apellido_materno,
    fecha_nacimiento, puesto_id
from empleado
where puesto_id = (select puesto_id from puesto where clave_puesto='jd')
minus
select empleado_id,nombre,apellido_paterno,apellido_materno,
    fecha_nacimiento, puesto_id
from empleado
where fecha_nacimiento >=to_date('01/01/1980','dd/mm/yyyy')
and fecha_nacimiento <=to_date('31/12/1985','dd/mm/yyyy');
```

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO FECHA_NACIMIENTO    PUESTO_ID
=========== ====== ================ ================ =================== =========
6           MARIO  JUAREZ           LOPEZ            1986-05-09 16:38:00 2
```

Después de revisar los operadores `union`, `intersect` y `minus` se puede concluir lo siguiente:

- Las columnas deben ser compatibles en cuanto a número y a tipo de dato, o al menos tipos de datos compatibles.
- Una vez que se obtienen los registros, estos se ordenan con respecto a las columnas de izquierda a derecha y se eliminan registros duplicados.
- La excepción al punto anterior es con la sentencia `union all`. En este caso no se realiza ordenamiento ya que los duplicados no se eliminan.

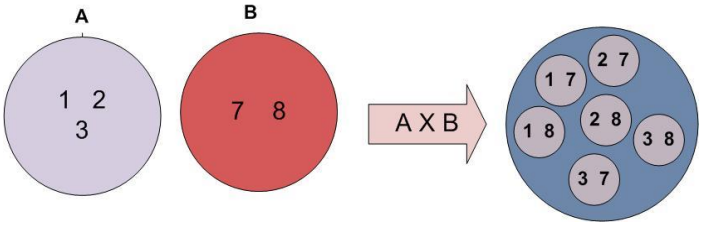
- Si se desea obtener los resultados en un orden diferente, se puede especificar la cláusula `order by` y debe aparecer hasta la última sentencia.
- Finalmente, una sentencia SQL puede contener una combinación de estos 3 operadores. En este caso, las sentencias se evalúan en el orden en el que se especifican (no hay precedencia, ojo, solo para estos 3 operadores). Si se desea forzar un orden de ejecución, se emplean paréntesis.

### *9.2.2.6. Operador Product $R \times S$*

El producto, o llamado también producto cartesiano formado por A X B de 2 conjuntos A y B, es un nuevo conjunto integrado por parejas de elementos {`x,y`}, donde `x` es un elemento de A, y `y` es un elemento de B.

$A = \{1,2,3\}$
$B = \{7,8\}$

$A\ X\ B = \{(1,7), (1,8), (2,7), (2,8), (3,7), (3,8)\}$




*Ejemplo:*

Obtener $R \times S$

R

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

S

| A | F | G | H | I |
|---|---|---|---|---|
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 5 | 70 | 400 | W | False |

| A | B | C | D | E | A | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 30 | 100 | X | True | 2 | 90 | 3 | M | False |
| 5 | 70 | 400 | W | False | 2 | 90 | 3 | M | False |
| 6 | 80 | 500 | M | False | 2 | 90 | 3 | M | False |
| 2 | 30 | 100 | X | True | 2 | 78 | 5 | S | False |
| 5 | 70 | 400 | W | False | 2 | 78 | 5 | S | False |
| 6 | 80 | 500 | M | False | 2 | 78 | 5 | S | False |
| 2 | 30 | 100 | X | True | 5 | 70 | 400 | W | False |
| 5 | 70 | 400 | W | False | 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False | 5 | 70 | 400 | W | False |

*Sintaxis SQL*

- En términos de SQL, al producto cartesiano se le conoce como `cross join`.

- En bases de datos el producto cartesiano de 2 relaciones R y S es una nueva relación formada por una combinación de todas las tuplas de R con cada una de las tuplas de S y sus atributos corresponden a los de R seguidos por los de S.

Sintaxis anterior

```
select {<column_name1>,<column_name2>,..| *}
from <table_name1>,<table_name2>,...
```

Sintaxis estándar

```
select {<column_name1>,<column_name2>,..| *}
from <table_name1> cross join <table_name2>...
```

- Observar que el producto cartesiano se puede aplicar entre cualquier pareja de tablas.

*Ejemplo:*
Obtener el producto cartesiano para las siguientes relaciones: BECA X LIBRO.

Contenido de BECA:

```
BECA_ID TIPO         DURACION MONTO
======= ============ ======== =====
500     ESCOLAR      5        6000
501     ALIMENTICIA  2        3000
502     VITALICIA    1        1000
503     TEMPORAL     1        1000
```

Contenido de LIBRO:

```
NOMBRE               TIPO_LIBRO PRECIO
==================== ========== ======
FABULAS              I          120.13
MATEMATICAS A        {null}     500.13
HISTORIA UNIVERSAL   C          500.14
```

Obteniendo el producto cartesiano:

Sintaxis anterior

```
select * from beca,libro;
```

Sintaxis estándar

```
select * from beca cross join libro;
```

| BECA_ID | TIPO | DURACION | MONTO | NOMBRE | TIPO_LIBRO | PRECIO |
|---|---|---|---|---|---|---|
| 500 | ESCOLAR | 5 | 6000 | FABULAS | I | 120.13 |
| 501 | ALIMENTICIA | 2 | 3000 | FABULAS | I | 120.13 |
| 502 | VITALICIA | 1 | 1000 | FABULAS | I | 120.13 |
| 503 | TEMPORAL | 1 | 1000 | FABULAS | I | 120.13 |
| 500 | ESCOLAR | 5 | 6000 | MATEMATICAS A | {null} | 500.13 |
| 501 | ALIMENTICIA | 2 | 3000 | MATEMATICAS A | {null} | 500.13 |
| 502 | VITALICIA | 1 | 1000 | MATEMATICAS A | {null} | 500.13 |
| 503 | TEMPORAL | 1 | 1000 | MATEMATICAS A | {null} | 500.13 |
| 500 | ESCOLAR | 5 | 6000 | HISTORIA UNIVERSAL | C | 500.14 |
| 501 | ALIMENTICIA | 2 | 3000 | HISTORIA UNIVERSAL | C | 500.14 |
| 502 | VITALICIA | 1 | 1000 | HISTORIA UNIVERSAL | C | 500.14 |
| 503 | TEMPORAL | 1 | 1000 | HISTORIA UNIVERSAL | C | 500.14 |

- Observar que el número de registros obtenidos es el resultado de multiplicar el número de registros de la tabla BECA (4) con el número de registros de la tabla LIBRO (3): 3 X 4 = 12.
- Observar en el resultado, para todos los registros de la tabla BECA, se combinan con cada valor de la tabla LIBRO.

### *9.2.2.7. Operador Join $R \bowtie_p S$*

El operador Join realiza una unión de 2 relaciones aplicando un predicado $p$ para asociar las tuplas de $R$ con las tuplas de $S$.

Este operador es fundamental en bases de datos para realizar consultas que actúan sobre un conjunto de tablas relacionadas entre sí. Existen varios tipos de Joins, mismos que se revisarán en la siguiente sección. A nivel general y en términos de álgebra relacional se define de la siguiente manera:

$$R \bowtie_p S = \sigma_p(R \times S)$$

En una operación Join típicamente el predicado verifica la igualdad entre los valores de una columna de R y otra de S. En la práctica, se emplea la PK de una tabla y la FK de la otra, es decir, se obtienen todos los registros de S que se relacionan con cada registro de R.

*Ejemplo:*

Obtener $R \bowtie_{R.A=S.A} S$

- Paso 1: Aplicando el producto cartesiano entre R y S:

R

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

S

| A | F | G | H | I |
|---|---|---|---|---|
| 2 | 90 | 3 | M | False |
| 2 | 78 | 5 | S | False |
| 5 | 70 | 400 | W | False |

R X S

| A | B | C | D | E | A | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| **2** | 30 | 100 | X | True | **2** | 90 | 3 | M | False |
| 5 | 70 | 400 | W | False | 2 | 90 | 3 | M | False |
| 6 | 80 | 500 | M | False | 2 | 90 | 3 | M | False |
| **2** | 30 | 100 | X | True | **2** | 78 | 5 | S | False |
| 5 | 70 | 400 | W | False | 2 | 78 | 5 | S | False |
| 6 | 80 | 500 | M | False | 2 | 78 | 5 | S | False |
| 2 | 30 | 100 | X | True | 5 | 70 | 400 | W | False |
| 5 | 70 | 400 | W | False | 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False | 5 | 70 | 400 | W | False |

- Paso 2: El predicado $R.A = S.A$ indica que el resultado del Join estará formado por todas aquellas tuplas donde los valores de las columnas A en R y A en S sean iguales. De la tabla anterior, se observa que 2 tuplas cumplen con el predicado del Join.

El resultado será:

| A | B | C | D | E | A | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 30 | 100 | X | True | 2 | 90 | 3 | M | False |
| 2 | 30 | 100 | X | True | 2 | 78 | 5 | S | False |
| 5 | 70 | 400 | W | False | 5 | 70 | 400 | W | False |

- Del resultado se comprueba que un registro de R se asocia con 2 registros de R.

En la práctica los manejadores no emplean la definición formal para realizar una operación Join debido a que el producto cartesiano es una operación costosa por la cantidad de registros que se genera. Se emplean técnicas más eficientes como son: Nesded Loop, Hash Joins, y Sort Merge Join principalmente.

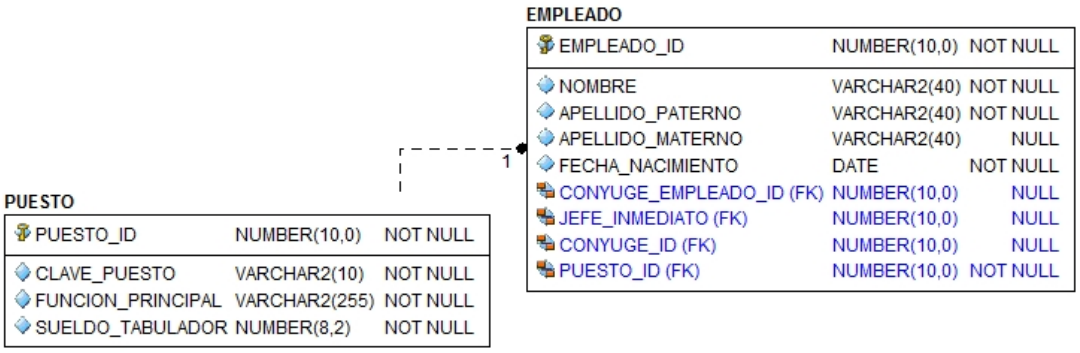
### *9.2.2.8. Transformación de consultas en términos de álgebra relacional.*

- Una consulta puede ser diseñada en términos de operadores del álgebra relacional.
- Una consulta SQL puede ser descompuesta en un conjunto de operaciones del álgebra relacional.

En términos generales la descomposición consiste en aplicar varias operaciones de manera sucesiva. El orden de aplicación de operadores puede variar. Esto implica que una sentencia puede temer múltiples expresiones algebraicas equivalentes.

*Ejemplos:*

Considerar las siguientes 2 relaciones Empleado E, y Puesto P del modelo relacional de la nómina:




A. Obtener una expresión en términos de álgebra relacional que obtenga el nombre y apellidos de todos los empleados donde el identificador de su puesto sea el 500:

$$R = \pi_{nombre,apellido-paterno,apellido-materno}(\sigma_{puesto-id=500}(E))$$

Otra solución es:

$$R = \sigma_{puesto-id=500}(\pi_{nombre,apellido-paterno,apellido-materno}(E))$$

B. Obtener el sueldo y el nombre de todos los empleados que tienen como función principal: 'manager'.

En este caso se requiere asociar a ambas tablas ya que se requiere obtener el sueldo de los empleados. Es decir, se requiere aplicar una operación Join:

$$R = \pi_{p.sueldo-tabulador,e.nombre}\left(\sigma_{funcion-principal='manager'}(P) \bowtie_{puesto-id} E\right)$$

Observar el predicado de la operación Join: puesto_id. Generalmente solo se expresa el nombre del campo que va a ser empleado en ambas tablas para verificar la correspondencia. En realidad, el predicado es: $e.puesto\_id = e.puesto\_id$

C. A los empleados que no tienen jefe así como a los empleados que tienen `puesto_id = 5` se les dará un incremento de sueldo. Mostrar el nombre y apellidos de los empleados que recibirán aumento.

$$R = \pi_{empleado-id,nombre,ap-pat,ap-mat}\left(\sigma_{jefe\ is\ null}(E)\right) \cup \pi_{empleado-id,nombre,ap-pat,ap-mat}\left(\sigma_{puesto-id}(E)\right)$$

## 9.3. Tipos de Joins

Como se mencionó anteriormente, para realizar la consulta de datos que se encuentran almacenados en varias tablas, es necesario relacionar, ligar (join) o asociar de alguna manera a las tablas participantes para poder extraer la información de manera adecuada.

Para realizar la liga o asociación de tablas, siempre se realiza *igualando los valores de los campos que tengan en común*. Típicamente la manera de ligar o asociar tablas es mediante el uso de las PKs y las FKs de las tablas involucradas. Por ejemplo, en el caso de la tabla `empleado`, `hijo_empleado`, el atributo en común es `empleado_id`. Es decir, las tablas están asociadas de manera directa a través de la FK en la tabla `hijo_empleado`.
Aunque en la práctica es poco común, es posible asociar tablas independientes (sin relación de PKs y FKs), por ejemplo, asociar el campo `nombre` de la tabla `empleado` con el campo `nombre` de la tabla `pensionada`.

### *9.3.1. Clasificación de las operaciones Join.*

- Inner joins
  - Equi join
  - Non equi join
  - Natural join
- Outer joins
  - Left outer join
  - Right outer join
- Cross join
- Self join

### *9.3.2. Inner Joins:*

La principal diferencia de los Inner joins con los outer joins es que los inner joins únicamente incluyen en la salida de la consulta los registros que tengan correspondencia con la tabla asociada. Por ejemplo, considere las siguientes tablas de datos `cliente` Y `tarjeta_credito`. Esta última contiene los datos de la tarjeta de crédito de un cliente.

CLIENTE

| CLIENTE_ID (pk) | NOMBRE | AP_PATERNO | AP_MATERNO |
|---|---|---|---|
| 1001 | JIMENA | MORALES | ROSALES |
| **1002** | HUGO | ALCAZAR | BENITEZ |
| **1003** | FERNANDO | ZARATE | PEREZ |
| **1004** | JORGE | MUNGUIA | SOLANO |
| 1005 | GERARDO | LUNA | MEJIA |

TARJETA_CREDITO

| CLIENTE_ID (fk) | TARJETA_ID | NUMERO | TIPO |
|---|---|---|---|
| **1002** | 3500 | 9876543234567890 | VISA |
| **1003** | 3501 | 93847284923849328 | MASTER |
| **1004** | 3502 | 54837849834893849 | AMERICAN |

Al realizar un inner join con respecto al campo `cliente_id`, por ejemplo, para mostrar un reporte que muestre los datos de los clientes y los datos de su tarjeta, un inner join daría como resultado únicamente los registros que tienen correspondencia en el campo `cliente_id` con el campo `cliente_id` de la tabla `tarjeta_credito`:

El resultado de realizar un inner join en las tablas anteriores, es el siguiente:

| CLIENTE_ID | NOMBRE | AP_PATERNO | AP_MATERNO | TARJETA_ID | NUMERO | TIPO |
|---|---|---|---|---|---|---|
| **1002** | HUGO | ALCAZAR | BENITEZ | 3500 | 9876543234567890 | VISA |
| **1003** | FERNANDO | ZARATE | PEREZ | 3501 | 93847284923849328 | MASTER |
| **1004** | JORGE | MUNGUIA | SOLANO | 3502 | 54837849834893849 | AMERICAN |

*9.3.2.1. Sintaxis SQL estándar para un inner Join*

```
select ...
from <table-1>
[inner |natural|cross] join <table-2>
[on <condition> | using <column-name>,...],...
```

join por defecto

- Se emplea la palabra `inner` para indicar que se trata de este tipo de join. Sin embargo, la palabra es opcional. Se usa para distinguir con mayor claridad a un inner join de un outer join. En la práctica la palabra `inner` es poco empleada.

*Ejemplo:*

El Join anterior entre las tablas `cliente` y `tarjeta` se expresa en SQL de la siguiente manera:

```
select  c.cliente_id, c.nombre,c.ap_paterno,c.ap_materno,
    t.tarjeta id,t.numero,t.tipo
from cliente c
join tarjeta t
on c.cliente_id=t.cliente_id;
```

- Para el caso de los campos `nombre`, `ap_paterno` y `ap_materno` el alias de la tabla es opcional, aunque se considera como buena práctica especificarlo.
- El único caso donde el alias es requerido, es para los casos en donde se tengan atributos con el mismo nombre en ambas tablas. El alias permite eliminar la ambigüedad ya que el manejador no sabría distinguir a cuál de las 2 tablas referirse. En este caso, la columna `cliente_id` requiere el uso de alias.
- Es posible emplear expresiones como `c.*, t.* y *.` En este último caso, la consulta mostraría todos los campos de ambas tablas, incluyendo el campo `cliente_id` de ambas tablas.

### *9.3.2.2. Sintaxis anterior para inner join*

La sintaxis vista anteriormente para expresar un inner join empleando `join - on` es la sintaxis actual y estándar reconocida por la especificación SQL a partir de 1992. Sin embargo, existe una forma alternativa para expresar un join llamada "sintaxis antigua o anterior". A pesar de no ser el estándar, esta sintaxis sigue vigente y muy utilizada en la práctica. A diferencia de la sintaxis estándar, la condición del join se especifica en la clausula `where`.

*Sintaxis SQL:*

```
select <attributes>
from table1,table2,... tableN
...where
[<qualifier>.]<column_name>
   <join-condition>
  [<qualifier>.]<column_name>
[
and
[<qualifier>.]<column_name>
   <join-condition>
  [<qualifier>.]<column_name>
]...
```

- Observar que, a diferencia de la sintaxis estándar, en la cláusula `select` se especifica la lista de tablas involucradas en la consulta. No existen las cláusulas `join`, `on`.
- `<join-condition>` contiene el predicado del join, la misma condición empleada en la cláusula `on: alias1.campo =alias2.campo`
- Todos los predicados se especifican en la cláusula `where`, incluido el predicado del Join.

*Ejemplo:*

El Join anterior entre las tablas `cliente` y `tarjeta` se expresa en SQL de la siguiente manera empleando sintaxis anterior

```
select  c.cliente_id, c.nombre,c.ap_paterno,c.ap_materno,
    t.tarjeta_id,t.numero,t.tipo
from cliente c,tarjeta t
where c.cliente_id=t.cliente_id;
```

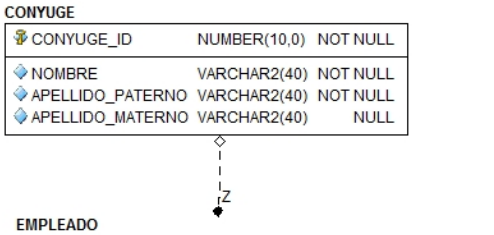
*Ejemplo:*

Mostrar los datos del empleado con id = 2, así como los datos de su cónyuge, considere que el cónyuge del empleado no es un empleado (es externo a la empresa, es decir, emplear el campo `conyuge_id` para aplicar la operación Join). Expresar la consulta en sintaxis estándar y sintaxis anterior.

*Sintaxis estándar:*

```
select e.nombre,e.apellido_paterno,
```

```
   e.apellido_materno,c.nombre,
   c.apellido_paterno,c.apellido_materno
from empleado e
join conyuge c
on e.conyuge_id=c.conyuge_id
where e.empleado_id=2
```




*Sintaxis anterior:*

```
select e.nombre,e.apellido_paterno,
   e.apellido_materno,c.nombre,
   c.apellido_paterno,c.apellido_materno
from empleado e, conyuge c
where e.conyuge_id=c.conyuge_id
and e.empleado_id=2
```

#### *9.3.2.3. Equi Join, Non Equi Join.*

- Estos 2 conceptos se emplean únicamente para distinguir el tipo de operador que se emplea en la condición de un inner join.
  - Equi Join emplea el operador de igualdad '='. Por ejemplo, `e.conyuge_id=c.conyuge_id`. De lo anterior, los ejemplos vistos hasta el momento son Inner Joins tipo Equi Join ya que hacen uso del operador de igualdad.
  - Non equi join emplea un operador diferente al de igualdad: $>, <, \leq, \geq$. Por ejemplo, `e.conyuge_id>=c.conyuge_id` . A este tipo de joins se les conoce también ***como theta*** $(\theta)$ ***Joins***. Son utilizados con muy poca frecuencia en la práctica.

#### *9.3.2.4. Natural Join.*

Natural Join es un Inner Join con una variante.

- En un Inner Join ***siempre*** se debe especificar el predicado o condición del join.
- En Natural Join, el predicado se construye automáticamente por lo que puede ser omitido. La regla para construir el predicado de forma automática o 'natural' es considerando el *nombre* de las columnas. Todas las columnas que tengan el mismo nombre serán empleadas para construir el predicado del Join.

Sintaxis estándar para Natural Join:

```
select ...
from <table-1>
[inner |natural|cross] join  <table-2>
[on <condition> | using <column-name>,...],...
```

En general se emplean las instrucciones `natural` y `using`.

*Ejemplo:*

Suponer que se tienen las relaciones R y S con los siguientes atributos:

R

| A | B | C | D | E |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 5 | 70 | 400 | W | False |
| 6 | 80 | 500 | M | False |

S

| A | B | F | G | H |
|---|---|---|---|---|
| 2 | 30 | 100 | X | True |
| 5 | 80 | 400 | W | False |
| 6 | 89 | 500 | M | False |

La expresión SQL que implementa un natural Join será:

```
select *
from r natural join s;
```

Y el resultado será:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 2 | 30 | 100 | X | True | 100 | X | True |

- Observar que la sentencia SQL no contiene el predicado del Join. En su lugar se emplea la palabra `natural`.
- En este ejemplo, existen 2 columnas con el mismo nombre: A y B. Por lo tanto, el predicado que se aplica de forma automática es: `R.A = S.A and R.B = S.B`
- El predicado solo se cumple para el primer registro, por lo que el resultado del natural Join solo contiene un registro.
- Observar que, en el resultado, solo aparecen las columnas A y B una sola vez. Empleando Natural Join, las columnas con el mismo nombre se consideran duplicadas, y por lo tanto se muestran una sola vez.

¿Qué sucede si existen columnas con el mismo nombre, pero no se desea que sean tomadas en cuenta como parte del predicado del Join?

En este caso se puede emplear una variante de Natural join a través del uso de la cláusula `using`.

*Ejemplo:*

Para la consulta anterior, solo se desea aplicar un Join empleando la columna A como predicado. La sentencia SQL que obtiene el resultado empleando natural join será:

```
select *
from r join s using(A);
```

El resultado será:

| A | B | C | D | E | B | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 2 | 30 | 100 | X | True | 30 | 100 | X | True |
| 5 | 70 | 400 | W | False | 80 | 400 | W | False |
| 6 | 80 | 500 | M | False | 89 | 500 | M | False |

- Dentro de la cláusula `using` se especifica la lista de atributos separados por coma, que serán considerados para construir el predicado del Join. En este caso, todas las columnas con nombre 'A'

se consideran como una sola y con ella se construirá el predicado `R.A = S.A` Notar que ahora 3 registros cumplen con el predicado.

- Observar que, en el resultado, a pesar de existir una columna 'B' que es común a ambas tablas esta aparece en el resultado 2 veces y se tratan como columnas diferentes ya que no fue especificada en la cláusula `using`.
- Otro punto importante es con respecto a los alias. Todas las columnas que se consideran como 'iguales', NO deben ser acompañadas de un alias de tabla o del nombre de la tabla. La razón es que por definición de natural Join, las columnas son equivalentes y por lo tanto no deben distinguirse con un alias.
- Finalmente, si una consulta emplea `using` para construir el predicado del join, NO debe incluir la palabra `natural`.

*Ejemplo:*

- ¡La siguiente consulta es incorrecta!

```
select r.a, s.a
from r join s using(A);
```

- En este caso, la columna A se está asociando a 2 tablas diferentes, pero en la cláusula `using` se está indicando que la columna A es equivalente en ambas tablas. Esto representa una inconsistencia en la consulta, por lo que se debe corregir de la siguiente forma:

```
select a
from r join s using(A);
```

*Ejemplo:*

- ¡La siguiente consulta es incorrecta!

```
select a
from r join s using(r.A);
```

Nuevamente, no se deben emplear alias en las columnas que se consideran similares o equivalentes entre las tablas que participan en un Join, en este caso dentro de la cláusula `using` nunca se debe especificar alias.

*Ejemplo:*

- ¡La siguiente consulta es incorrecta!

```
select a
from r natural join s using(a);
```

- En este caso se está empleando la palabra `natural` y `using` para ligar a las tablas r y s lo cual es incorrecto. No pueden emplearse ambas para un mismo Join.

Recomendaciones:

- Usar `natural` y/o `using` solo en casos necesarios para evitar confusión con la cláusula `on` empleada en un inner join. No hacer mezclas de estas 3 cláusulas.

*Ejemplo:*

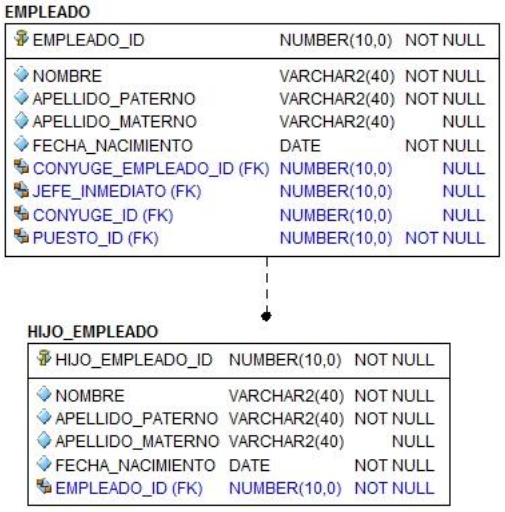
Mostrar el nombre del cliente, su identificador y el número de tarjeta de todos los clientes tipo VISA. Emplear Natural Join.

```
select cliente_id, c.nombre, t.tipo
from  cliente c
natural join tarjeta_credito t
where t.tipo = 'VISA';
```

- En este ejemplo no se requiere `using` ya que la única columna en común es `cliente_id` y es la que se emplea para construir el predicado del join.
- Observar que todas las columnas pueden hacer uso del alias de su tabla excepto la columna `cliente_id`.

*Ejemplo:*

Mostrar el identificador del empleado, el nombre y apellidos del empleado, así como el nombre y los apellidos de sus hijos. Emplear Natural Join.




```
Select
empleado_id,e.nombre,e.apellido_paterno,
  e.apellido_paterno, h.nombre,
  h.apellido_paterno, h.apellido_materno
from  empleado e  join  hijo_empleado h
using(empleado_id) where empleado_id =1
```

- Observar que en este ejemplo se requiere hacer uso de `using` ya que existen 3 columnas con el mismo nombre, pero se desea emplear como predicado únicamente a la columna `empleado_id`.
- Por lo anterior, se requiere incorporar `using` indicando que solo la columna `cliente_id` se empleará para formar el predicado.
- Observar que el resto de las columnas hacen uso de alias para distinguir entre los valores de empleado e `hijo_empleado`. La columna `cliente_id` No debe llevar alias.

*Ejemplo:*

Suponer que se desea aplicar una variante a la consulta anterior: Obtener los datos de los empleados donde el nombre y el apellido del papá y del hijo coincidan. Empleando Natural Join, la consulta será:

```
select nombre,apellido_paterno,e.apellido_materno as empleado_ap_mat,
  h.apellido_materno as hijo_ap_mat
from empleado e join hijo_empleado h
using(empleado_id,nombre, apellido_paterno)
where empleado_id  =1;
```

- Observar que, en este caso, se agregan los campos nombre y apellido_paterno a la cláusula using, y por lo tanto, no deben acompañarse de alias.

### *9.3.3. Outer Joins.*

En el caso de un outer join es posible mostrar los registros faltantes, aunque no tengan una correspondencia con la tabla asociada. Por ejemplo, suponga que se desea mostrar el reporte de todos los clientes, aunque estos no cuenten con una tarjeta de crédito.

Si la consulta se hiciera con inner join, se obtendría lo siguiente:

| CLIENTE_ID | NOMBRE | AP_PATERNO | AP_MATERNO | TARJETA_ID | NUMERO | TIPO |
|---|---|---|---|---|---|---|
| 1002 | HUGO | ALCAZAR | BENITEZ | 3500 | 9876543234567890 | VISA |
| 1003 | FERNANDO | ZARATE | PEREZ | 3501 | 93847284923849328 | MASTER |
| 1004 | JORGE | MUNGUIA | SOLANO | 3502 | 54837849834893849 | AMERICAN |

Un outer join mostraría lo siguiente:

| CLIENTE_ID | NOMBRE | AP_PATERNO | AP_MATERNO | TARJETA_ID | NUMERO | TIPO |
|---|---|---|---|---|---|---|
| 1001 | JIMENA | MORALES | ROSALES | {NULL} | {NULL} | {NULL} |
| 1002 | HUGO | ALCAZAR | BENITEZ | 3500 | 9876543234567890 | VISA |
| 1003 | FERNANDO | ZARATE | PEREZ | 3501 | 93847284923849328 | MASTER |
| 1004 | JORGE | MUNGUIA | SOLANO | 3502 | 54837849834893849 | AMERICAN |
| 1005 | GERARDO | LUNA | MEJIA | {NULL} | {NULL} | {NULL} |

En este caso, el primer resultado es incorrecto ya que la consulta dice que se deben mostrar a TODOS los clientes sin importar si cuentan o no con tarjeta. Para estas situaciones, se debe emplear outer join.

En un outer Join se obtienen los registros que tienen correspondencia en ambas tablas similar a un inner join, y adicionalmente se obtienen los registros que no tienen una correspondencia en la otra tabla. Los campos que no contengan dicha correspondencia se presentan como nulos, tal cual como se muestra en la tabla anterior.

Existen 2 casos de outer joins: `left outer join` y `right outer join`.

***Sintaxis estándar***:

```
... from <table-name1> {left|right|full [outer]} join
    <table-name2> [on <condition>] | [using <column-name>,...],...
```

*Ejemplo:*

Retomando el ejemplo del cliente y sus tarjetas de crédito:

```
select c.nombre,c.ap_paterno, c.ap_materno,t.tarjeta_id,t.numero,t.tipo
from cliente c
left join tarjeta_credito t
on c.cliente_id=t.cliente_id;
```

- La palabra `outer` es opcional, en la práctica es común omitirla, pero se puede dejar para hacer énfasis.

### *9.3.3.1. Diferencias entre Left Outer Join y Right Outer Join.*

*Escenario a considerar:*
Un estudiante puede o no tener una beca, y una beca puede o no estar asociada a un estudiante, es decir, puede haber becas vacantes. Observar los datos de estas 2 tablas.

ESTUDIANTE

| ESTUDIANTE_ID | NOMBRE | AP_PATERNO | AP_MATERNO | BECA_ID |
|---|---|---|---|---|
| 2001 | MARIA | JIMENEZ | SALAZAR | 500 |
| 2002 | HUGO | RODRIGUEZ | BENITEZ | {NULL} |
| 2003 | ARMANDO | ZARATE | MARTINEZ | 501 |
| 2004 | RODRIGO | MUNGUIA | SOLANO | {NULL} |
| 2005 | JULIO | LUNA | MEJIA | 502 |

BECA

| BECA_ID | TIPO | DURACION | MONTO |
|---|---|---|---|
| 500 | ESCOLAR | 5 | 6000 |
| 501 | ALIMENTICIA | 2 | 3000 |
| 502 | VITALICIA | 1 | 1000 |
| 503 | TEMPORAL | 1 | 1500 |

*Left outer join:*

Mostrar todos los estudiantes y si tienen beca, mostrar los datos de la beca.

```
select *
from estudiante e
left outer join  beca b
on e.beca_id=b.beca_id;
```

```
ESTUDIANTE_ID NOMBRE   AP_PATERNO AP_MATERNO BECA_ID BECA_ID TIPO         DURACION MONTO
============= ======= ========== ========== ======= ======= =========== ======== ======
2001          MARIA    JIMENEZ    SALAZAR    500     500     ESCOLAR     5        6000
2002          HUGO     RODRIGUEZ  BENITEZ    {null}  {null}  {null}      {null}   {null}
2003          ARMANDO  ZARATE     MARTINEZ   501     501     ALIMENTICIA 2        3000
2004          RODRIGO  MUNGUIA    SOLANO     {null}  {null}  {null}      {null}   {null}
2005          JULIO    LUNA       MEJIA      502     502     VITALICIA   1        1000
```

*Right outer join:*

Mostrar todas las becas disponibles, y en caso de que esté asociada a un estudiante, mostrar sus datos:

```
select *
from estudiante e
right outer join beca b
on e.beca_id=b.beca_id;
```

```
ESTUDIANTE_ID NOMBRE   AP_PATERNO AP_MATERNO BECA_ID BECA_ID TIPO         DURACION MONTO
============= ======= ========== ========== ======= ======= =========== ======== =====
2001          MARIA    JIMENEZ    SALAZAR    500     500     ESCOLAR     5        6000
2003          ARMANDO  ZARATE     MARTINEZ   501     501     ALIMENTICIA 2        3000
2005          JULIO    LUNA       MEJIA      502     502     VITALICIA   1        1000
{null}        {null}   {null}     {null}     {null}  503     TEMPORAL    1        1000
```

Las sentencias `left` y `right` reflejan el orden en el cual las tablas son procesadas por el manejador. La primera tabla es la que se especifica en la cláusula `from` corresponde con el “lado izquierdo”, y la segunda tabla corresponde al “lado derecho”.

*Importante:*

- En un left outer join, se obtienen los registros que hacen match en ambas tablas con respecto al campo especificado en la instrucción ON (similar a un inner join), **y adicionalmente los registros en la tabla del lado izquierdo que no tienen correspondencia con el lado derecho** (en el ejemplo anterior, se adicionan los registros con `estudiante_id` 2002 y 2004, ya que ambos no tienen correspondencia con la tabla `beca`).

- En un right outer join se obtienen los registros que hacen match en ambas tablas con respecto al campo especificado en la instrucción `on` (similar a un inner join), **y adicionalmente los registros en la tabla del lado derecho que no contienen correspondencia con el lado izquierdo** (en el ejemplo anterior, se anexa el registro de la tabla BECA con `beca_id =503`, ya que este no tiene correspondencia con la tabla `estudiante`).

Los registros resaltados en los 2 resultados anteriores corresponden a registros adicionales agregados por las características de un left y right outer join respectivamente. En ambos casos, si se hiciera un inner join, ninguno de estos registros aparecería en la lista de resultados.

### *9.3.3.2. Outer joins con sintaxis anterior.*

Es posible usar la sintaxis anterior en outer joins de los 2 tipos. El único inconveniente en este caso es que, la sintaxis es específica a cada manejador. Por ejemplo, para Oracle, se agrega la cadena "(+)".

Este operador se escribe después del nombre de la tabla que no necesariamente tiene correspondencia con la otra.

*Sintaxis*:

***Left outer join***

```
...where
     [<qualifier>.]<column_name>=
            [<qualifier>.]<column_name>(+)
   [
    and
     [<qualifier>.]<column_name>=
            [<qualifier>.]<column_name>(+)
   ]...
```

Sin correspondencia

***Right outer join***

```
...where
     [<qualifier>.]<column_name>(+)=
            [<qualifier>.]<column_name>
   [
    and
     [<qualifier>.]<column_name>(+)=
            [<qualifier>.]<column_name>
   ]...
```

*Ejemplos:*

Mostrar todos los estudiantes y si tienen beca, mostrar los datos de la beca.

```
select *
from estudiante e, beca b
where e.beca_id=b.beca_id(+);
```

Mostrar todas las becas disponibles, y en caso de que, asociada a un estudiante, mostrar sus datos:

```
select *
from estudiante e, beca b
where e.beca_id(+)=b.beca_id;
```

mandar correo campos

### 9.3.4. Cross Join

Corresponde a realizar una operación de producto cartesiano, obtiene todas las parejas posibles de registros de ambas tablas.

*Ejemplo:*

Sintaxis estándar:

```
select *
from estudiante e
cross join beca;
```

*Ejemplo:*
Sintaxis anterior:

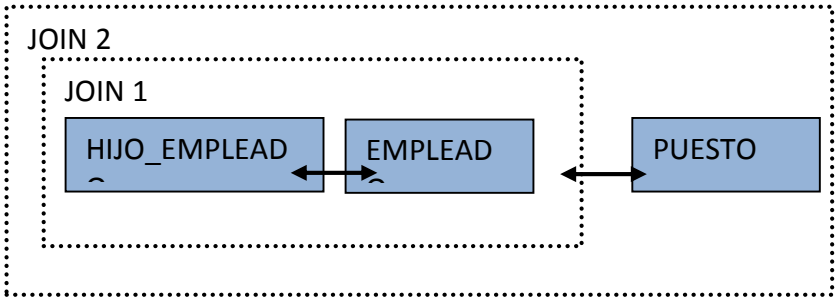
```
select *
from estudiante e, beca b;
```

### 9.3.5. Joins en consultas con múltiples tablas.

El resultado de hacer un join entre 2 tablas se puede visualizar como una ***nueva tabla virtual***, a la cual se le puede aplicar otro join con la siguiente tabla indicada en la sentencia haciendo una especie de encadenamiento con la instrucción `join`.

*Ejemplo:*
Seleccionar el nombre, apellidos y clave del puesto del empleado cuyo hijo tiene por nombre JUAN MARTINEZ SALAZAR.

```
select e.empleado_id, e.nombre, e.apellido_paterno,
   e.apellido_materno, p.clave_puesto
from hijo_empleado h
join empleado e on h.empleado_id=e.empleado_id
join puesto p on e.empleado_id = p.puesto_id
where h.nombre ='juan'
and h.apellido_paterno='martinez'
and h.apellido_materno='salazar';
```



*Ejemplo:*

Sintaxis anterior:

```
select e.empleado_id, e.nombre, e.apellido_paterno,
   e.apellido_materno, p.clave_puesto
from  hijo_empleado h, empleado e, puesto p
where h.empleado_id = e.empleado_id
and e.puesto_id= p.puesto_id
and h.nombre ='juan'
and h.apellido_paterno='martinez'
and h.apellido_materno='salazar';
```

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO CLAVE_PUESTO
=========== ====== ================ ================ ============
1           JUAN   MARTINEZ         LOPEZ            DG
```

#### *9.3.5.1. Regla N-1 Joins para sintáxis anterior*

Deben existir al menos ***n-1*** joins en una sentencia SQL expresada en sintaxis anterior en donde aparecen ***n*** tablas. De no cumplir esta condición, se generará un producto cartesiano, y se obtendrán resultados incorrectos.

*Ejemplo:*
Si en el ejemplo anterior, se omite por accidente el join `e.puesto_id= p.puesto_id`, se obtiene:

```
EMPLEADO_ID NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO CLAVE_PUESTO
=========== ====== ================ ================ ============
1           JUAN   MARTINEZ         LOPEZ            DG
1           JUAN   MARTINEZ         LOPEZ            AC
1           JUAN   MARTINEZ         LOPEZ            TC
1           JUAN   MARTINEZ         LOPEZ            SJ
1           JUAN   MARTINEZ         LOPEZ            JD
```

Observar que se obtienen 5 registros, todos ellos con el mismo empleado. Esto se debe a que en ausencia del join entre las tablas `empleado` y `puesto` a través del campo `puesto_id`, se produce un producto cartesiano, entre el resultado obtenido y la tabla `puesto`. Observar que el campo `empleado_id=1` se combina con cada uno de los registros existentes en la tabla PUESTO (5 registros). Esto se puede observar en el campo `clave_puesto`, aparecen todos los valores posibles del campo.

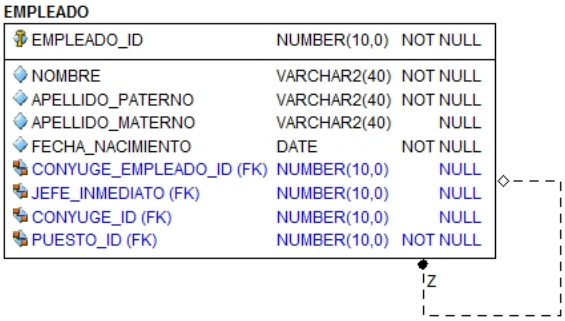
### *9.3.6. Self Joins*

En un self join una tabla es ligada consigo misma en lugar de ser ligada con otra. Internamente se trabaja con 2 conjuntos de datos, ambos asociados a la misma tabla.
No existe como tal una cláusula para este tipo de Join, es decir, no existe la palabra 'self'. La distinción de este tipo de Join es únicamente para hacer énfasis de que se trata de un Join aplicado a 2 conjuntos de datos que pertenecen a la misma relación.

*Ejemplo:*

Mostrar el nombre y apellidos de los empleados, así como el nombre y apellidos de su cónyuge, el cual también es un empleado.

```
select e.nombre,e.apellido_paterno,
e.apellido_materno,c.nombre,
c.apellido_paterno, c.apellido_materno
from empleado e
join empleado c
on e.conyuge_empleado_id=c.empleado_id;
```




- En este tipo de Joins se debe tener especial cuidado en el uso de alias y en la condición del join.
- Observar que se emplea la misma tabla 2 veces, pero con alias diferentes. El alias 'e' identifica a los empleados, y el alias 'c' a sus cónyuges.
- Observar la condición del Join, se debe igualar el identificador del cónyuge de un empleado con su identificador del empleado. De hacer otra combinación se obtendrán resultados incorrectos.
- Otra forma de construir el predicado es pensar en ambos conjuntos como tablas separadas. `conyuge_empleado_id` es la FK que proviene de una tabla padre con llave primaria `empleado_id`. En este caso la PK de la tabla padre representa a los cónyuges, y es `empleado_id`. Por lo tanto, el predicado se forma igualando la PK de la tabla padre (`c.empleado_id`) con la FK de la tabla hija (`e.conyuge_empleado_id`). Es decir: `e.conyuge_empleado_id=c.empleado_id`

*Ejemplo:*

Sintaxis anterior:

```
select e.nombre,e.apellido_paterno,e.apellido_materno,
   c.nombre,c.apellido_paterno, c.apellido_materno
from empleado e,empleado c
where e.conyuge_empleado_id=c.empleado_id;
```

Se obtendrán los siguientes datos:

```
NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO
====== ================ ================ ====== ================ ================
ANGEL  JUAREZ           AGUIRRE          ANGELA RAMIREZ          LUNA
MARIO  JUAREZ           LOPEZ            MARIA  AGUILAR         GUZMAN
```

*Ejemplo:*

*9.3.6.1. self join con outer join:*

Mostrar un reporte de todos los empleados, y si están casados con otro empleado(a), mostrar sus datos también.

```
select e.nombre,e.apellido_paterno,e.apellido_materno,
   c.nombre,c.apellido_paterno, c.apellido_materno
from empleado e
left join empleado c
on e.conyuge_empleado_id=c.empleado_id;
```

- En este caso se obtienen todos los empleados sin importar si hay correspondencia con `empleado_conyuge_id`.

```
NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO NOMBRE APELLIDO_PATERNO APELLIDO_MATERNO
====== ================ ================ ====== ================ ================
ANGEL  JUAREZ           AGUIRRE          ANGELA RAMIREZ          LUNA
MARIO  JUAREZ           LOPEZ            MARIA  AGUILAR          GUZMAN
JUAN   MARTINEZ         LOPEZ            {null} {null}           {null}
MARIA  AGUILAR          GUZMAN           {null} {null}           {null}
ANGELA RAMIREZ          LUNA             {null} {null}           {null}
```

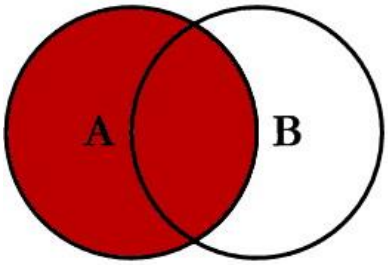
### *9.3.7. Resumen Joins:*

La siguiente tabla muestra un resumen de los estilos y sintaxis empleada en el uso de Joins.

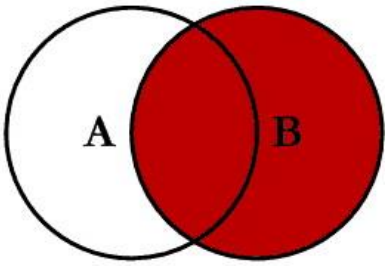
| Clasificación | Tipo | Ejemplo |
|---|---|---|
| CROSS | Cross Join | `select * from t1,t2` |
| | | `select * from t1 cross join t2` |
| INNER | Sintaxis anterior | `select * from t1,t2 where t1.c1=t2.c1` |
| | Natural Join | `select * from t1 natural join t2` |
| | Natural con Using | `select * from t1 join t2 using (c1)` |
| | Condicional | `select * from t1 join t2 on t1.c1= t2.c1` |
| OUTER | Right | `select * from t1 right outer join t2 on t1.c1=t2.c1` |
| | Left | `select * from t1 left outer join t2 on t1.c1=t2.c1` |
| | Full | `select * from t1 full outer join t2 on t1.c1=t2.c1` |
| | Right, sintaxis anterior Oracle | `select * from t1,t2 where t1.c1(+)=t2.c1` |
| | Left sintaxis anterior Oracle | `select * from t1,t2 where t1.c1=t2.c1(+)` |

En forma gráfica:

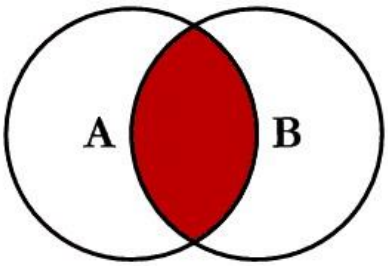
# SQL JOINS



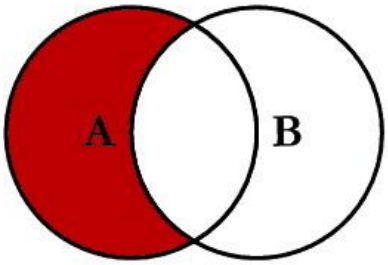

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



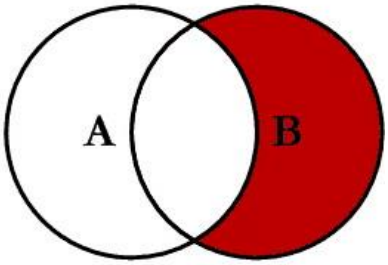

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



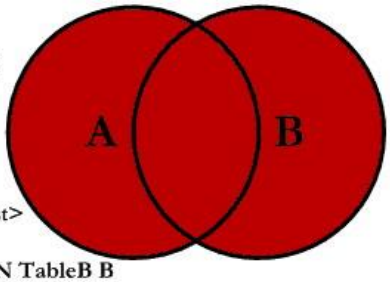

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



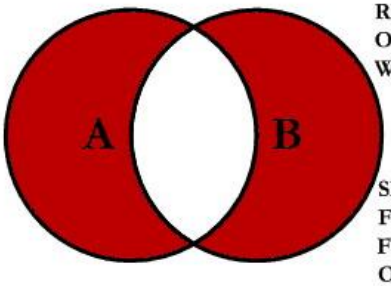

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```