

Tema 4.3 .-

Consultas SQL

Acceso a las tablas de la base de datos de ejemplo

- En el servidor de Oracle de *siriusext* existe un esquema denominado **profesor**
- En el esquema **profesor** están creadas y cargadas las tablas de la base de datos de ejemplo a utilizar en las prácticas de *Consultas SQL*
- Los usuarios personales de los alumnos tienen autorizado el acceso de lectura a dichas tablas
 - acceso concedido en las sentencias **grant** del final del guion de creación de los objetos del esquema profesor y que se presentará a continuación
- El tema teórico de álgebra relacional, en su documento 3.2b, presenta el esquema semántico de la base de datos a usar en los primeros compases, tanto en **álgebra relacional** como en **consultas SQL**
 - se trata de una primera versión simplificada, con dos entidades y una interrelación binaria entre ellas, que da lugar a tres relaciones: **s**, **p** y **sp**
 - aunque un poco más adelante se presentará una versión con tres entidades y una interrelación ternaria entre ellas, que da lugar a cuatro relaciones: **s**, **p**, **j** y **spj** y que corresponde a las tablas creadas en el ‘**Guion de Creación de la Base de Datos SPJ en el Esquema Profesor**’



Acceso a las tablas de la base de datos de ejemplo

- El documento ‘*Guion de Creación de la Base de Datos SPJ en el Esquema Profesor*’ describe con todo detalle la estructura y contenido de dicha base de datos
 - Estúdielo detenidamente, hasta su total comprensión, antes de seguir adelante y consulte con el profesor cualquier duda que se le presente.
- En él aparece un nuevo tipo de objeto denominado **vista** (*view*)
 - Una **vista** es una **tabla virtual**, cuyo cuerpo **no existe** y por lo tanto es **calculado** a partir de otras tablas y/o vistas, cada vez que se necesite, según las especificaciones de una **consulta** definida por una sentencia *select*.
 - Dado que las únicas tablas existentes son las de la base de datos ternaria (**s**, **p**, **j**, **spj**) y que sin embargo los primeros pasos en las consultas SQL se van a dar en la base de datos binaria (**s**, **p** y **sp**)
 - se hace necesario crear una **vista sp** en el esquema profesor que obtenga sus datos a partir de una operación de *agregación* de los datos existentes en la tabla **spj**, y cuyo significado exacto ya estudiarán más adelante.
 - De esta forma, siempre podrán consultar de forma transparente, el contenido de las relaciones **s**, **p** y **sp**

Acceso a las tablas de la base de datos de ejemplo

- Pruebe a referirse desde su esquema a dichas tablas con un *describe*
 - Para referirse a un objeto de otro esquema basta con prefijar el nombre del objeto con el nombre del esquema
 - Por ejemplo: *profesor.s*
 - Cuando lo haya conseguido, estudie la siguiente referencia del manual de SQL de Oracle:
 - https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_7001.htm#CJAJCDDF
- en la que se presenta el tipo de objeto *sinónimo*
- Averigüe como utilizarlo para conseguir referirse a las tablas de la susodicha base de datos desde su esquema con los mismos nombres que desde el esquema *profesor*.

Consultas SQL

- Los diferentes SGBD comerciales pueden presentar diferencias entre sí y con respecto al estandar de SQL.
- El formato que se presenta aquí, pretende ser lo más genérico posible.

SELECT

- El esquema general de la sentencia *select* responde, básicamente, al siguiente formato:

SELECT ‘lista de atributos’

FROM ‘relación’

[WHERE ‘condiciones de selección’]

‘lista de atributos’

- Expresa la lista de atributos que se desea obtener.
- Su formato es:

SELECT [ALL | DISTINCT] {atrib₁, atrib₂, ..., atrib_n | *}

- Indica una acción sobre los elementos seleccionados:
 - ALL: permite expresar que **no se desea** omitir del resultado las tuplas repetidas. Es el valor por defecto.
 - DISTINCT: permite expresar que **se desea** omitir del resultado las tuplas repetidas.
- Constituye la lista de selección de los atributos deseados.
 - *: indica que se desean todos los atributos.

‘relación’

- Expresa la lista de tablas (tablas base o vistas) en las que se encuentran los atributos implicados en el ‘SELECT’:
 - los que se desean obtener, y
 - los involucrados en la/las condiciones del WHERE.
- Su formato es:

FROM nombre_tabla₁, {nombre_tabla₂, ..., nombre_tabla_n}

- Una sentencia *select* siempre trabaja sobre **una única relación**.
 - Si se incluye **solo una**, esa será la relación con la que trabaja la sentencia *select*
 - Si se incluye mas de una, la relación será el **producto cartesiano** de las incluídas
- Se puede renombrar una tabla en caso de necesidad:
 - FROM s, p, j
 - Las tablas se referirán como ‘s’, ‘p’ y ‘j’
 - FROM s x, s y
 - Las tablas se referirán como ‘x’ y ‘y’
 - Dichos **alias de tabla** estarán en vigor en el ámbito del bloque *select-from-where* en el que se definen

‘condiciones de selección’

- Expresa el predicado de selección de tuplas de la relación:
 - Expresa una condición de evaluación lógica
 - Se aplica a las tuplas que forman la tabla referida
 - Determina la selección de la tupla como parte de la respuesta.
- Su formato es variopinto
 - Existe toda una gama de diferentes comparadores, entre otros:
 - Relacionales clásicos:
 - $>$, $<$, $>=$, $<=$, $<>$, AND, OR, NOT
 - Específicos:
 - EXISTS, IN, LIKE, IS NULL

Metodología

- Información a obtener:
 - Qué campos son el objeto de nuestra selección.
 - En qué tablas se encuentran dichos campos.
- Condición a cumplir:
 - Identificar los campos implicados.
 - Determinar en qué tablas se encuentran dichos campos.
 - Construir la expresión lógica de selección.
- Tablas involucradas:
 - Se determinan en los dos apartados previos.
 - Necesidad de renombrar alguna de las tablas implicadas.

Operaciones de recuperación.

Obtener los códigos y las situaciones de aquellos suministradores que estén en Las Palmas.

```
SELECT sn, situacion  
FROM s  
WHERE ciudad='Las Palmas';
```

1.1.- Recuperación simple.

Obtener los códigos de las partes suministradas.

```
SELECT pn  
FROM sp;
```

1.2.- Recuperación simple.

Obtener los códigos de las partes suministradas.

```
SELECT DISTINCT pn
```

```
FROM sp;
```

Si la consulta puede producir tuplas repetidas **debe** usarse la cláusula *distinct*.

Obtener los códigos de todas las partes.

```
SELECT pn
```

```
FROM p;
```

Si la consulta no puede producir tuplas repetidas **NO debe** usarse la cláusula *distinct*.

1.3.- Recuperación simple.

Obtener todos los datos de todos los suministradores.

```
SELECT sn, snombre, situacion, ciudad
```

```
FROM s; ..../..
```

```
SELECT *
```

```
FROM s;
```

1.4.- Recuperación cualificada.

Obtener los códigos de los suministradores de Las Palmas con situación mayor que 20.

```
SELECT sn  
FROM s  
WHERE ciudad='Las Palmas'  
AND situacion>20;
```



GRUPO DE ESTRUCTURAS DE DATOS

1.5.- Recuperación con ordenación.

Obtener el código y la situación de los suministradores que viven en Las Palmas o en Telde, en orden descendente de situación, y a igualdad de situación, en orden ascendente del nombre del proveedor.

```
SELECT sn, situacion
```

```
FROM s
```

```
WHERE ciudad='Las Palmas' or ciudad='Telde'
```

```
ORDER BY situacion DESC, snombre;
```

```
SELECT sn, situacion
```

```
FROM s
```

```
WHERE ciudad='Las Palmas' or ciudad='Telde'
```

```
ORDER BY 2 DESC, snombre;
```

- En la cláusula order by:
 - Las columnas de ordenación se pueden citar por nombre o por orden de aparición en la cláusula select.
 - Cada columna puede acompañarse de ASC (valor por defecto) o DESC
 - Cada columna puede acompañarse NULLS FIRST (valor por defecto para orden descendente) o NULLS LAST (valor por defecto para orden ascendente) (estas opciones no están disponibles en MySQL)

Obtener el código y la situación de todos los suministradores, en orden descendente de situación, y a igualdad de situación, en orden ascendente del nombre del proveedor.

```
SELECT sn, situacion
```

```
FROM s
```

```
ORDER BY 2 DESC, snombre;
```

```
SELECT sn, situacion
```

```
FROM s
```

```
ORDER BY 2 DESC NULLS LAST, snombre;
```




GRUPO DE ESTRUCTURAS DE DATOS

Reuniones

1.6.- Recuperación desde más de una tabla.

Obtener, para cada parte suministrada, el código de parte y los nombres de las ciudades donde viven proveedores que venden esa parte.

```
SELECT DISTINCT pn, ciudad
FROM s, sp
WHERE s.sn=sp.sn;
```

- La consulta trabaja con el **producto cartesiano** de *s* y *sp*, dado que en la cláusula **from** aparece más de una tabla.
- Si se añade una condición que iguale los campos comunes de ambas tablas, la consulta trabajará con el **join natural** de dichas relaciones.
 - Con una pequeña diferencia con respecto al álgebra relacional,
 - puesto que no se eliminan las columnas duplicadas
 - sin embargo, esto no es grave puesto que se pueden distinguir prefijándolas con los nombres o alias de sus tablas
 - A dichas condiciones, que reducen un *producto cartesiano* a un *join natural*, se las denomina **condiciones de join**
- Obsérvese que en las *condiciones de join* es muy frecuente que se haga necesario **cualificar** los nombres de las columnas con los nombres (o alias, en su caso) de las tablas a las que pertenecen dichos campos, a fin de poderlos distinguir
 - Las *condiciones de join* no constituyen el único caso en el que se puede hacer necesario usar dicho recurso sintáctico para romper **ambigüedades**; allí dónde aparezca la necesidad de distinguir entre **columnas homónimas**, se solventará mediante la cualificación con el nombre de la tabla

1.7.- Recuperación implicando el JOIN de una tabla consigo misma.

Obtener todos los pares de códigos de proveedores que vivan en la misma ciudad.

```
SELECT x.sn, y.sn  
FROM s x, s y  
WHERE x.ciudad=y.ciudad  
AND x.sn<y.sn;
```

1.8.- Recuperación usando subconsulta anidada.

Obtener los nombres de los suministradores que venden la parte 'P2'.

```
SELECT DISTINCT snombre
FROM s, sp
WHERE s.sn=sp.sn
AND pn='P2';
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE sn = ANY
              (SELECT sn
               FROM sp
               WHERE pn='P2');
```

•En una expresión con subconsulta anidada se presenta la siguiente estructura básica:

select...

from...

where condición conectora (select...
from...
where...);

•La condición conectora (también conocida como conector) comparará **un** valor con **un conjunto** de valores y por tanto se hace necesario el uso de un cuantificador (**ALL** o **ANY**) (*todos* o *alguno*) en conjunción con el operador de comparación (=, <, >, <=, >=, <>)

•Se dice entonces que la expresión SQL tiene 2 bloques select..from...where...,

•uno exterior

•y otro interior, más concretamente, en la cláusula where del bloque exterior

•Es posible analizar la subconsulta independientemente de la consulta exterior y a continuación evaluar el conector para así poder obtener el resultado de la consulta exterior

1.9.- Recuperación usando <ANY.

Obtener los códigos de los suministradores que tengan un valor de situación menor que el máximo valor de situación de la tabla s.

```
SELECT sn  
FROM s  
WHERE situacion < ANY (SELECT situacion  
                        FROM s);
```

1.10.- Recuperación con subconsulta escalar y operador de comparación sin cuantificador.

Obtener los códigos de los suministradores que viven en la misma ciudad que el proveedor cuyo código es 'S1'.

```
SELECT sn
FROM s
WHERE ciudad = (SELECT ciudad
                 FROM s
                 WHERE sn='S1');
```

- Desde el punto de vista semántico está garantizado que la subconsulta **devolverá un único valor**
- Por tanto, se puede **obviar** la necesidad del **cuantificador** en el **conector**
- Este tipo de subconsultas se denominan *subconsultas escalares*

1.11.- Recuperación usando IN.

Obtener los nombres de los suministradores que suministran la parte 'P2'.

```
SELECT DISTINCT snombre  
FROM s, sp  
WHERE s.sn=sp.sn  
AND pn='P2';
```

../..

```
SELECT DISTINCT snombre  
FROM s  
WHERE sn IN(SELECT sn  
FROM sp  
WHERE pn='P2');
```

- Dada su frecuencia de uso, la combinación =**ANY** se puede contraer sintácticamente como **IN**
- Son **totalmente** equivalentes



1.12.- Recuperación con multiples niveles de anidamiento.


Obtener los nombres de los proveedores que venden, al menos, una parte roja.

```
SELECT DISTINCT snombre
FROM s
WHERE sn IN
      (SELECT sn
       FROM sp
       WHERE pn IN
            (SELECT pn
             FROM p
             WHERE color='rojo'));
```


1.13.- Recuperación con subquery con referencia entre bloques: bloques correlacionados.

Obtener los nombres de los suministradores que suministran la parte 'P2'.

```
SELECT DISTINCT snombre  
FROM s  
WHERE 'P2' IN (SELECT pn  
                FROM sp  
                WHERE sn=s.sn);
```



- Existe **correlación** o **referencia entre bloques** cuando una subconsulta hace referencia a una tabla de una consulta superior a ella en su anidamiento.
- En este caso, **NO** es posible analizar la subconsulta **independientemente** de la consulta exterior.
- En su lugar, **para cada fila** procesada en la consulta exterior referenciada:
 - Se evalúa la subconsulta.
 - Se evalúa el conector
 - Se decide sobre la selección de dicha fila de la consulta externa referenciada.
- Las columnas no cualificadas de la subconsulta se resolverán buscándolas primero en las tablas de la subconsulta y a continuación en las tablas de la consulta exterior referenciada.

1.14.- Recuperación con subquery con referencia entre bloques: bloques correlacionados.

Obtener los nombres de los suministradores que suministran la parte 'P2'.

```
SELECT DISTINCT snombre
FROM s, sp
WHERE s.sn=sp.sn
AND pn='P2';
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE sn IN
(SELECT sn
FROM sp
WHERE pn='P2');    ../..
```

```
SELECT DISTINCT snombre
FROM s
WHERE 'P2' IN
(SELECT pn
FROM sp
WHERE sn=s.sn);
```

1.15.- Recuperación con subquery con la misma tabla implicada en ambos bloques.

Obtener los códigos de los proveedores que vendan al menos una parte suministrada por 'S2'.

```
SELECT DISTINCT sn  
FROM sp  
WHERE pn IN  
  
      (SELECT pn  
       FROM sp  
       WHERE sn='S2');
```

1.16.- Recuperación con subquery con referencia entre bloques y la misma tabla implicada en ambos bloques.

Obtener los códigos de las partes vendidas por más de un proveedor.

```
SELECT DISTINCT pn
FROM sp spX
WHERE pn IN
      (SELECT pn
       FROM sp
       WHERE sn <> spX.sn);
```



Grupo de ESTRUCTURAS de DATOS

1.17.- Recuperación usando EXISTS.

Obtener los nombres de los proveedores que vendan la parte 'P2'.

```
SELECT DISTINCT snombre
FROM s
WHERE EXISTS
  (SELECT *
   FROM sp
   WHERE sn=s.sn
   AND pn='P2');
```

- El conector exists se evalúa como verdadero si la subconsulta que le sigue devuelve alguna fila
- En caso contrario se evalúa como falso

1.18.- Recuperación usando EXISTS.

Obtener los nombres de los proveedores que vendan la parte 'P2'.

```
SELECT DISTINCT snombre
FROM s, sp
WHERE s.sn=sp.sn
AND pn='P2';
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE 'P2' IN
    (SELECT pn
     FROM sp
     WHERE sn=s.sn);
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE sn IN
    (SELECT sn
     FROM sp
     WHERE pn='P2');
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE EXISTS
    (SELECT *
     FROM sp
     WHERE sn=s.sn
     AND pn='P2');
```

Diferencias

1.19.- Recuperación usando ALL.

- Obtener los códigos de los suministradores que NO venden la parte 'P2'.
 - En Álgebra Relacional: $s[sn] \text{ minus } (sp \text{ where } pn='P2')[sn]$

```
SELECT sn
FROM s
WHERE sn <> ALL
          (SELECT sn
           FROM sp
           WHERE pn='P2');
```

- En SQL no existe solución monobloque para las diferencias.
- La forma mas simple será expresarla en dos bloques sin correlación,
 - con el minuendo en el bloque exterior,
 - el sustraendo en el interior
 - y un conector basado en la cabecera común de minuendo y sustraendo y que indique que en el resultado han de estar las filas del minuendo que sean diferentes a todas las del sustraendo.



Grupo de ESTRUCTURAS de DATOS

1.20.- Recuperación usando NOT IN.

Obtener los códigos de los suministradores que NO venden la parte 'P2'.

```
SELECT sn
FROM s
WHERE sn <> ALL
      (SELECT sn
       FROM sp
       WHERE pn='P2');
```

../..

```
SELECT sn
FROM s
WHERE sn NOT IN
      (SELECT sn
       FROM sp
       WHERE pn='P2');
```

- Dada su frecuencia de uso, la combinación **<>ALL** se puede contraer sintácticamente como **NOT IN**
- Son **totalmente** equivalentes
- Obsérvese que así como en las **reuniones** se usa el cuantificador **ANY**
 - contrayéndose como **IN** en las **equireuniones**
 - en las **diferencias** se usa el cuantificador **ALL**
 - contrayéndose como **NOT IN**.

1.21.- Recuperación usando NOT IN con correlación.

Obtener los nombres de los suministradores que NO venden la parte 'P2'.

```
SELECT DISTINCT snombre
FROM s
WHERE sn NOT IN
      (SELECT sn
       FROM sp
       WHERE pn='P2');
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE 'P2' NOT IN
      (SELECT pn
       FROM sp
       WHERE sn=s.sn);
```



GRUPO DE ESTRUCTURAS DE DATOS

1.22.- Recuperación usando NOT EXISTS.

Obtener los nombres de los suministradores que NO venden la parte 'P2'.

```
SELECT DISTINCT snombre
FROM s
WHERE sn NOT IN
  (SELECT sn
   FROM sp
   WHERE pn='P2');
```

```
SELECT DISTINCT snombre
FROM s
WHERE 'P2' NOT IN
  (SELECT pn
   FROM sp
   WHERE sn=s.sn);    ../..
```

```
SELECT DISTINCT snombre
FROM s
WHERE NOT EXISTS
  (SELECT *
   FROM sp
   WHERE sn=s.sn
   AND pn='P2');
```



GRUPO DE ESTRUCTURAS DE DATOS

1.23.- Diferencia implícita

🔗 Obtener los códigos de los suministradores que solo venden partes que no sean 'P2'.

(Desambiguando: Obtener los códigos de los suministradores que no venden la parte 'P2', pero venden algo)

```
SELECT DISTINCT sn
FROM sp
WHERE sn NOT IN
  (SELECT sn
   FROM sp
   WHERE pn='P2');
```

```
SELECT DISTINCT sn
FROM sp x
WHERE 'P2' NOT IN
  (SELECT pn
   FROM sp
   WHERE sn=x.sn);    ../..
```

```
SELECT DISTINCT sn
FROM sp x
WHERE NOT EXISTS
  (SELECT *
   FROM sp
   WHERE sn=x.sn
   AND pn='P2');
```



Grupo de ESTRUCTURAS de DATOS

1.24.- Diferencia implícita

➤ Obtener los nombres de los suministradores que solo venden partes que no sean 'P2'.

(Desambiguando: Obtener los nombres de los suministradores que no venden la parte 'P2', pero venden algo)

```
SELECT DISTINCT snombre
FROM s
WHERE sn IN
  (SELECT sn
   FROM sp
   WHERE sn NOT IN
     (SELECT sn
      FROM sp
      WHERE pn='P2'));
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE sn IN
  (SELECT sn
   FROM sp
   WHERE 'P2' NOT IN
     (SELECT pn
      FROM sp
      WHERE sn=s.sn));
```

../..

```
SELECT DISTINCT snombre
FROM s
WHERE sn IN
  (SELECT sn
   FROM sp
   WHERE NOT EXISTS
     (SELECT *
      FROM sp
      WHERE sn=s.sn
      AND pn='P2'));
```



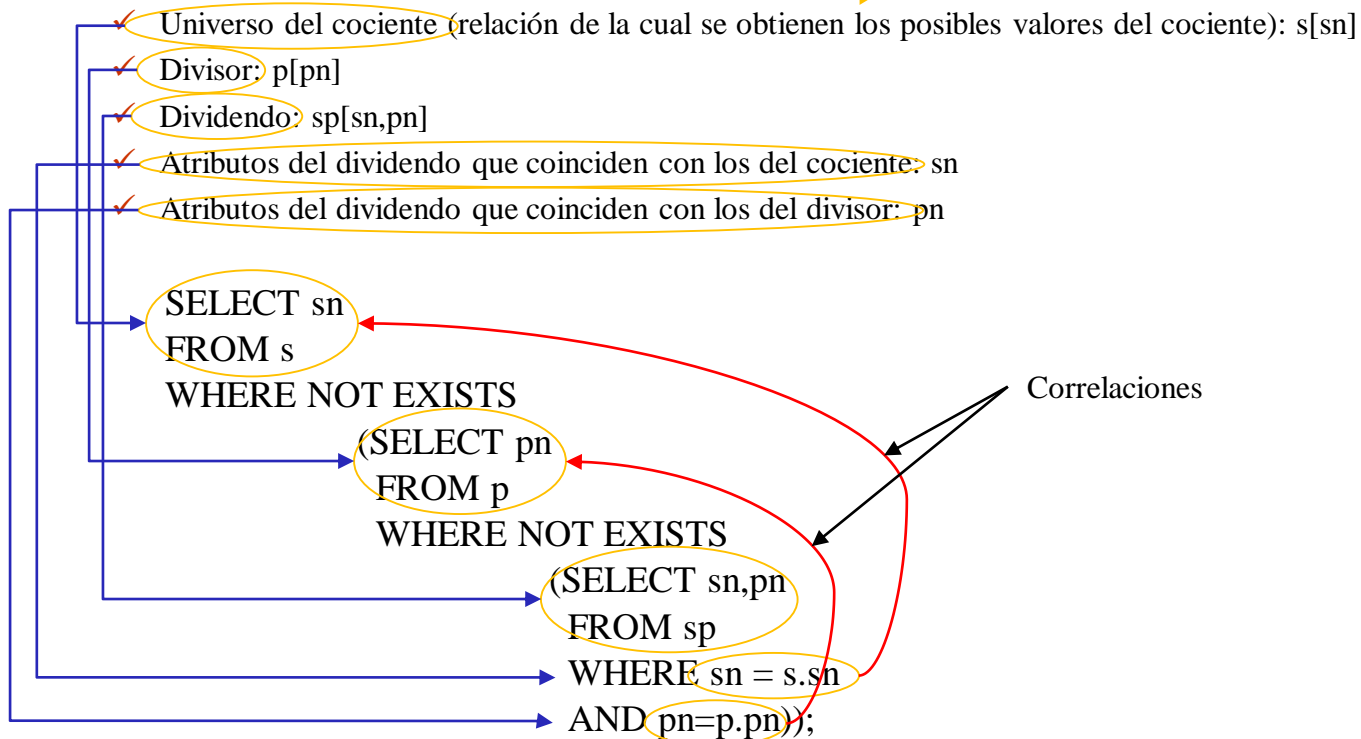
GRUPO DE ESTRUCTURAS DE DATOS

Divisiones

1.25.- Recuperación usando NOT EXISTS.

- Obtener los códigos de los suministradores que suministran todas las partes.

- En Álgebra Relacional: $sp[sn,pn] \div p[pn]$ ➡ Análisis de la expresión algebraica:



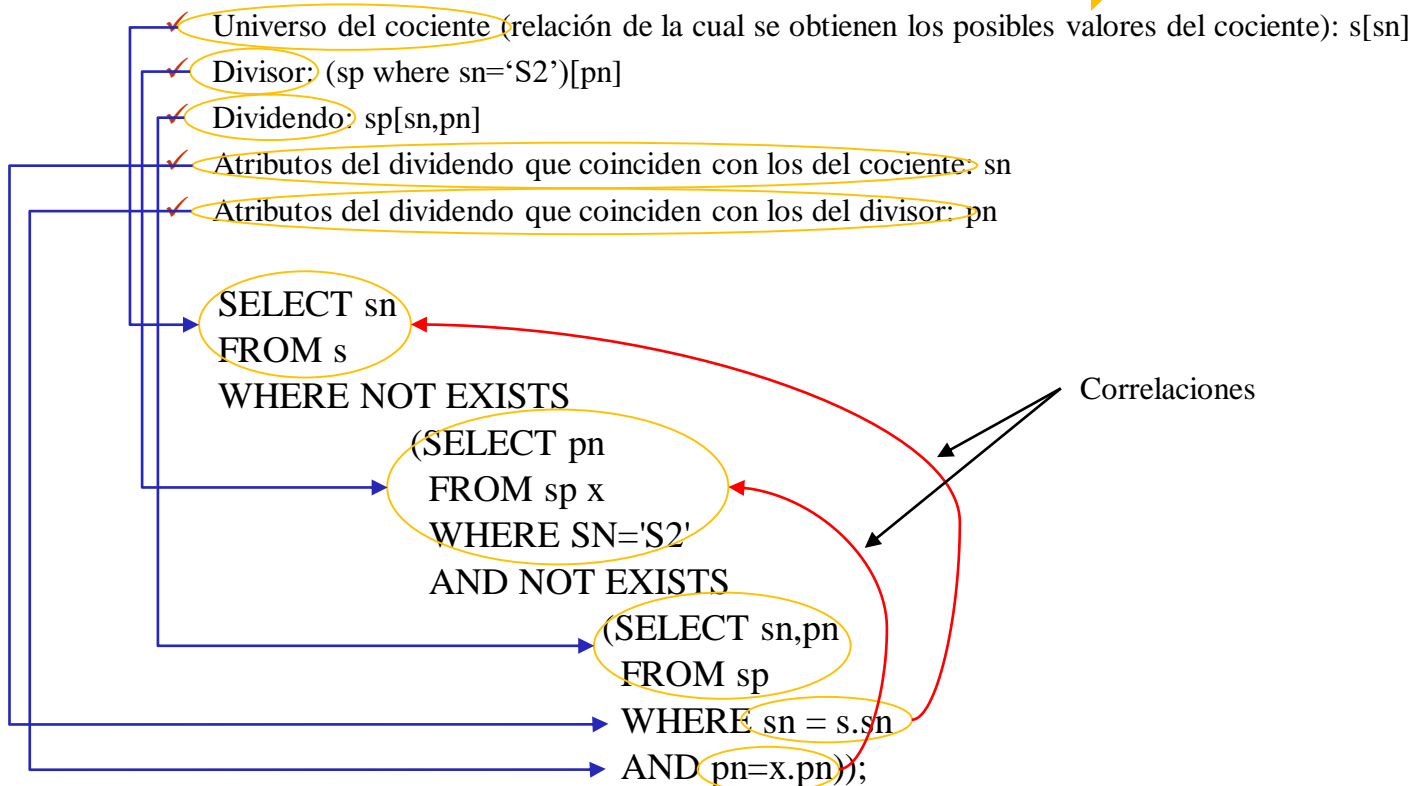


GRUPO DE ESTRUCTURAS DE DATOS

1.26.- Recuperación usando NOT EXISTS.

➤ Obtener los códigos de los suministradores que venden, al menos, todas las partes suministradas por el proveedor 'S2'.

- En Álgebra Relacional: $sp[sn,pn] \text{ div } (sp \text{ where } sn='S2')[pn]$ ➡ Análisis de la expresión algebraica:





Grupo de ESTRUCTURAS de DATOS

1.27.- Recuperación usando NOT EXISTS.

Obtener los nombres de los suministradores que suministran todas las partes.

```
SELECT DISTINCT snombre
FROM s
WHERE NOT EXISTS
    (SELECT *
    FROM p
    WHERE NOT EXISTS
        (SELECT *
        FROM sp
        WHERE sn = s.sn
        AND pn=p.pn));
```



Operadores de conjuntos



1.28.- Recuperacion usando UNION.

Obtener los códigos de las partes que o bien pesan mas de 18 kilos o bien son suministradas por 'S2' (o ambas cosas a la vez).

```
SELECT pn
FROM p
WHERE peso > 18
UNION
SELECT pn
FROM sp
WHERE sn='S2';
```

- El operador UNION elimina automáticamente los posibles duplicados del resultado, procedentes de:
 - duplicados internos a cada bloque y/o
 - repeticiones del mismo valor en dos bloques distintos.
- También existen los operadores UNION ALL, INTERSECT y MINUS.
 - En MySQL no existe INTERSECT, pero no es un operador básico, como ya se estudió en Álgebra Relacional
- El operador UNION ALL no elimina automáticamente ninguno de los posibles duplicados del resultado
- Los operadores INTERSECT y MINUS eliminan automáticamente cualquier posible duplicado del resultado.

Valores calculados



GRUPO DE ESTRUCTURAS DE DATOS

1.29.- Recuperacion de valores calculados y renombrar columnas o expresiones de la cláusula select.

- Obtener los códigos de parte y el peso en gramos de cada una de las partes.
 - Los pesos de la tabla p están en kilogramos.

```
SELECT pn, peso*1000
```

```
FROM p;                ../..
```

```
SELECT pn, peso*1000 Peso_en_gramos
```

```
FROM p;                ../..
```

```
SELECT pn, peso*1000 AS Peso_en_gramos
```

```
FROM p;                ../..
```

```
SELECT pn, peso*1000 "Peso en gramos"
```

```
FROM p;                ../..
```

```
SELECT pn "Código de producto", peso*1000 "Peso en gramos"
```

```
FROM p;
```

- Como puede observarse en las ejecuciones:
 - Oracle ignora las especificaciones de mayúsculas y minúsculas en los alias de columnas a no ser que se usen comillas dobles,
 - mientras que MySQL las observa, se usen comillas dobles o no.



GRUPO DE ESTRUCTURAS DE DATOS

Nulos

1.30.- Recuperación implicando NULL.

Obtener los códigos de los suministradores con situación nula.

```
SELECT sn  
FROM s  
WHERE situacion IS NULL;
```

Obtener los códigos de los suministradores con situación no nula.

<pre>SELECT sn FROM s WHERE situacion IS NOT NULL; ..</pre>	<pre>SELECT sn FROM s WHERE situacion = situacion;</pre>
---	--



Condiciones de rango



1.31.- Recuperación usando BETWEEN.

Obtener las ventas cuya cantidad esté entre 200 y 800, ambos inclusive.

```
SELECT *  
FROM sp  
WHERE cantidad BETWEEN 200 AND 800;
```

Búsqueda de patrones en ristras de caracteres

1.32.- Recuperación usando LIKE.

- Su sintaxis es: *ristra* **LIKE** *patrón*, donde *ristra* es una constante o una columna y *patrón* es una ristra que puede contener:
 - caracteres, representandose a sí mismos y
 - los caracteres especiales:
 - '%' que representa a cualquier ristra, con 0 o mas caracteres
 - '_' que representa a exactamente 1 carácter
- Obtener los datos de los proveedores cuya ciudad contenga los caracteres 'al' seguidos y en cualquier posición.

```
SELECT *  
FROM s  
WHERE ciudad LIKE '%al%';
```

- Obtener los datos de los proveedores cuya ciudad contenga los caracteres 'al' en la segunda y tercera posición.

```
SELECT *  
FROM s  
WHERE ciudad LIKE '_al%';
```

Expresiones alternativas para los derivados del Producto Cartesiano



Productos Cartesianos y Reuniones internas

--Productos Cartesianos

```
select * from s,p;  
select * from s cross join p;
```

--Reuniones internas

--Join-Theta

```
select * from s,p where s.ciudad<>p.ciudad;  
select * from s join p on s.ciudad<>p.ciudad;
```

--Equijoin

```
select * from s,p where s.situacion=p.peso;  
select * from s join p on s.situacion=p.peso;  
select * from s,p where s.ciudad=p.ciudad;  
select * from s join p on s.ciudad=p.ciudad;
```

--Join Natural

```
select * from s join p using(ciudad);  
select * from s natural join p;
```

Reuniones externas

--Join-Theta izquierdo

```
select * from s,p where s.ciudad<>p.ciudad(+);
select * from s left join p on s.ciudad<>p.ciudad;
```

--Equijoin izquierdo

```
select * from s,p where s.situacion=p.peso(+);
select * from s left join p on s.situacion=p.peso;
select * from s,p where s.ciudad=p.ciudad(+);
select * from s left join p on s.ciudad=p.ciudad;
```

--Join Natural izquierdo

```
select * from s left join p using(ciudad);
select * from s natural left join p;
```

--Join-Theta derecho

```
select * from s,p where s.ciudad(+)<>p.ciudad;
select * from s right join p on s.ciudad<>p.ciudad;
```

--Equijoin derecho

```
select * from s,p where s.situacion(+)=p.peso;
select * from s right join p on s.situacion=p.peso;
select * from s,p where s.ciudad(+)=p.ciudad;
select * from s right join p on s.ciudad=p.ciudad;
```

--Join Natural derecho

```
select * from s right join p using(ciudad);
select * from s natural right join p;
```

--Join-Theta completo

```
select * from s,p where s.ciudad<>p.ciudad(+)
union
select * from s,p where s.ciudad(+)<>p.ciudad;
select * from s full join p on s.ciudad<>p.ciudad;
```

--Equijoin completo

```
select * from s,p where s.situacion=p.peso(+)
union
select * from s,p where s.situacion(+)=p.peso;
select * from s full join p on s.situacion=p.peso;
select * from s,p where s.ciudad=p.ciudad(+)
union
select * from s,p where s.ciudad(+)=p.ciudad;
select * from s full join p on s.ciudad=p.ciudad;
```

--Join Natural completo

```
select * from s full join p using(ciudad);
select * from s natural full join p;
```

Agregación



GRUPO DE ESTRUCTURAS DE DATOS

Agregación

- Hasta aquí, si se exceptúa el uso de *distinct*, siempre se ha cumplido la siguiente regla:
 - El conjunto respuesta de un bloque *select* tiene tantas filas como las de la relación de la cláusula *from* que cumplen la restricción de la cláusula *where*.
- La **agregación** rompe con esa regla:
 - Forma grupos de filas, tantos como le indique la cláusula *group by* que se explicará posteriormente.
 - Si no existe cláusula *group by*, formará un único grupo englobando a todas las filas.
 - El conjunto respuesta del bloque *select* con **agregación** tendrá, como máximo, tantas filas como grupos se hayan formado.
 - Se pueden restringir las filas del conjunto respuesta de un bloque *select* con agregación mediante la cláusula *having*, como se verá posteriormente.
- ¿Cuándo hay agregación en un bloque *select*?:
 - Cuando en la cláusula *select* se quieren obtener valores agregados,
 - es decir, se hace uso de las **funciones de agregación** disponibles en SQL
 - para **agrupar** o **resumir** de alguna manera valores de la **misma columna**
 - procedentes de **diferentes filas** que **comparten** un mismo **grupo**.
 - se haga o no uso de la cláusula *group by*
 - dado que, como ya se ha mencionado anteriormente, si no aparece dicha cláusula, por defecto se forma un único grupo con todas las filas.
 - Cada valor obtenido en la cláusula *select* debe ser un **representante válido de la agrupación** que se ha formado en el bloque:
 - es decir, debe ser un **valor único para todas las filas del grupo**.

2.1.- Función COUNT en la cláusula select.

Obtener el número total de suministradores:

```
SELECT COUNT(*)  
FROM s;          ../..
```

- Devuelve 6, porque tiene 6 proveedores en el único grupo formado.

```
SELECT COUNT(sn)  
FROM s;          ../..
```

- Devuelve 6, porque tiene 6 proveedores en el único grupo formado, todos ellos con sn no nulo.

```
SELECT COUNT(situacion)  
FROM s;
```

- Devuelve 5, porque tiene 6 proveedores en el único grupo formado, 5 de ellos con situación no nula.

2.2.- Función COUNT en la cláusula select.

Obtener el número total de suministradores que suministren realmente alguna parte.

```
SELECT COUNT(DISTINCT sn)  
FROM sp;
```

2.3.- Función COUNT en la cláusula select con un predicado.

Obtener el número de ventas de la parte 'P2'.

```
SELECT COUNT(*)  
FROM sp  
WHERE pn='P2';
```

2.4.- Función SUM en la cláusula select con un predicado.

Obtener la cantidad total suministrada de la parte ‘P2’.

```
SELECT SUM(cantidad)
FROM sp
WHERE pn='P2';
```

2.5.- Función MAX en un subquery.

Obtener los códigos de los suministradores con situación menor que el máximo valor de situación de la tabla s.

```
SELECT sn  
FROM s  
WHERE situacion <  
      (SELECT MAX(situacion)  
      FROM s);
```

2.6.- Uso de GROUP BY.

Obtener, para cada parte suministrada, el código de parte y el total de la cantidad suministrada.

```
SELECT pn, SUM(cantidad) as "Suma"  
FROM sp  
GROUP BY pn;
```

- Las filas de la tabla *sp* se dividen en **grupos diferentes** para **cada valor de *pn***.
- Así pues, el bloque agregado tendrá tantas filas en el conjunto respuesta como valores diferentes de *pn* existan en *sp*:
 - 6 filas, una para cada grupo, en las que aparecen:
 - *pn*, que es un representante válido porque dentro de cada grupo el valor es el mismo y
 - *sum(cantidad)*, que es un representante válido porque dentro de cada grupo su valor es único.

2.7.- Uso de GROUP BY con HAVING.

Obtener los códigos de las partes vendidas por más de un proveedor.

- Es el mismo enunciado que el ejemplo 1.16

```
SELECT DISTINCT pn
FROM sp spX
WHERE pn IN
      (SELECT pn
       FROM sp
       WHERE sn <> spX.sn);      ../..
```

```
SELECT pn
FROM sp
GROUP BY pn
HAVING COUNT(*) > 1;
```

- La cláusula *having* **restringe las filas** del conjunto respuesta del **bloque agregado**.
- Al igual que en la cláusula *select* de un bloque agregado, las condiciones han de establecerse sobre **representantes válidos de los grupos** del bloque.
- La cláusula *having* es al *group by* lo mismo que la cláusula *where* es al *from*
- Nótese que esta segunda expresión no funcionaría con la interrelación ternaria *spj*.

2.8.- Ejemplo resumen.

Obtener el código de parte y la cantidad máxima suministrada de esa parte para todas las partes tales que la cantidad suministrada es mayor que 300 (excluyendo del total las ventas cuya cantidad sea menor o igual a 200), ordenando el resultado en orden descendente del código de parte dentro del orden ascendente de la cantidad maxima

```
SELECT pn, MAX(cantidad)
FROM sp
WHERE cantidad > 200
GROUP BY pn
HAVING SUM(cantidad) > 300
ORDER BY 2, pn DESC;
```