edix
Digital Workers

UF4-Programación Consultas Avanzadas

Profesor RAÚL SALGADO VILAS

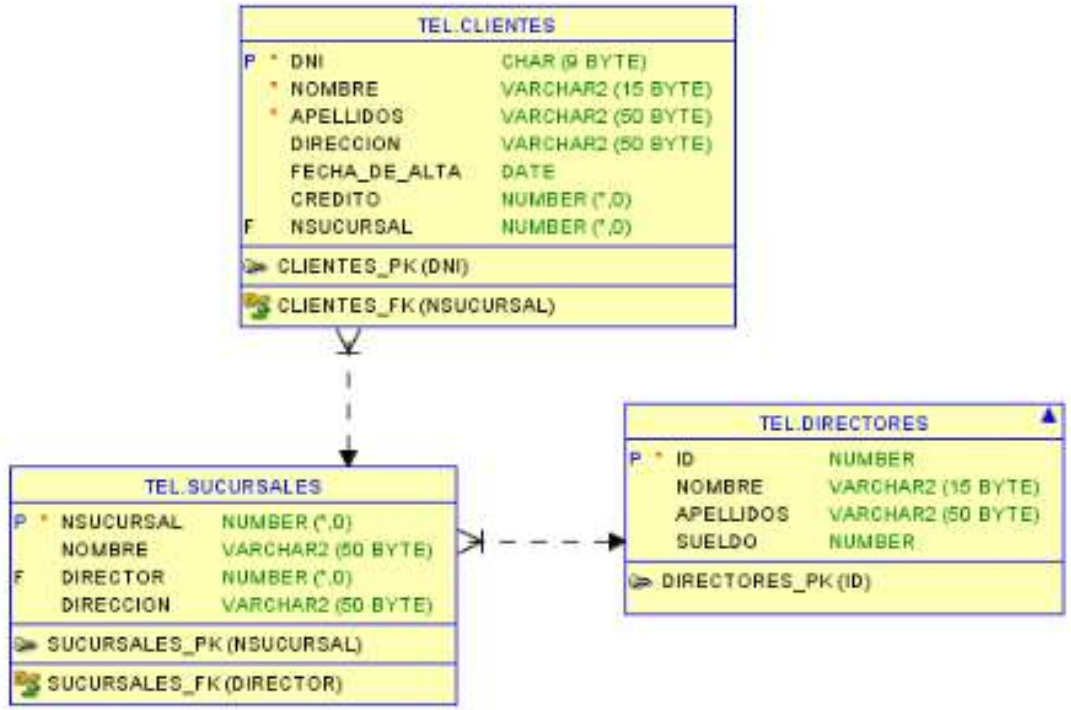
UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS

- Las consultas avanzadas nos **permiten profundizar en la búsqueda de información**, pudiendo usar en una **sola sentencia las condiciones y resultados** intermedios para poder**llegar a la información requerida**.

- Objetivos
  - Conocer cómo se pueden obtener datos de diferentes tablas y combinar el resultado
  - Unir diferentes consultas en una sola
  - Realizar consultas agrupadas

Bases de datos de ejemplo: Entidades bancarias y clientes

- A lo largo de la unidad se va a trabajar con numerosos ejemplos que en algunos casos va a hacer referencia a un modelo de datos ya definido.



Bases de datos de ejemplo: Entidades bancarias y clientes

SUCURSALES

NSUCURSAL	NOMBRE	DIRECTOR	DIRECCION
1001	Sucursal Centro	12	Avd. del Estilo, 45
1002	Sucursal Oeste	15	Avd. Mediterráneo, 14
1003	Sucursal Este	20	Calle Martínez, 45
1004	Sucursal Norte	(null)	Calle Especias, 23

CLIENTES

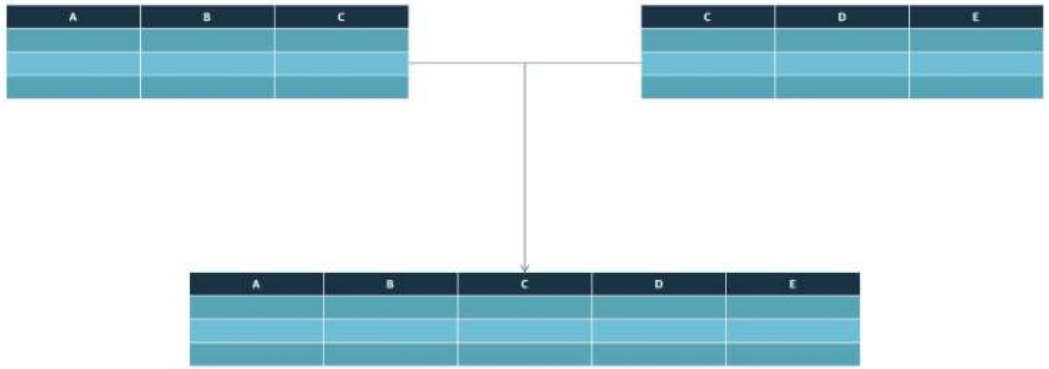
DNI	NOMBRE	APELLIDOS	DIRECCION	FECHA_DE_ALTA	CREDITO	NSUCURSAL
30515454K	Ana	Martín Martín	Calle del Socorro, 1	08/01/18	1500	1001
33358796A	Marta	López Ruiz	Calle Martinez, 76	14/09/17	600	1001
78458784B	Antonio	Castillo Mentas	Calle Soles, 14	(null)	500	1002

DIRECTORES

ID	NOMBRE	APELLIDOS	SUELDO
12	Alberto	Pérez Martín	1800
15	Antonio	López López	1500
20	Silvia	Martín Martín	1300
13	Pedro	García Martín	1900

UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS

- Introducción:
- En las consultas que hemos estudiado hasta este momento hemos trabajado con los resultados de una sola tabla, es decir los datos que se entregaban sólo pertenecían a una tabla.
- SQL nos ofrece la posibilidad de juntar tablas en el resultado final de una consulta de forma que la información asociada a través de claves foráneas pueda mostrarse en un sólo registro.
- La unión de relaciones que se almacenan en lugares distintos forma parte de las operaciones del álgebra relacional.
- La instrucción que se utiliza en SQL es **JOIN** y sus **variantes** nos permiten trabajar con varias tablas. **JOIN** en realidad es el **operador cartesiano de Algebra relacional** y **combina** los registros de dos tablas tal que su resultado es la **combinación de registros de la primera tabla, con los de la segunda**.



- Consultas multitabla: En las primeras consultas con las que hemos trabajado sólo hemos utilizado una tabla, pero hay veces que una consulta necesita columnas de varias tablas. En este caso, las tablas se expresarán a la derecha de la palabra reservada FROM separadas por comas y se relacionarán campos de ambas tablas. Normalmente la relación será entre la clave primaria de la tabla1 y la clave foránea de la tabla2.

- La sintaxis general sería:

```
SELECT columna1,columna2, .... de las tablas indicadas en el FROM
FROM tabla1, tabla2...
WHERE tabla1.columna = tabla2.columna .... ;
```

- Ejemplo: Listar los nombres de sucursales junto con los nombres y apellidos de sus directores:

```
SELECT sucursales.nombre, directores.nombre,
directores.apellidos FROM sucursales, directores
WHERE sucursales.director=directores.id;
```

- Como se puede observar se están solicitando campos de las dos tablas. En este caso el nombre de la tabla sucursales y el nombre y apellidos de la tabla directores. También comparo el identificador de director en ambas tablas para que sólo muestre los registros relacionados.

- En las condiciones del WHERE además de los campos a relacionar se pueden añadir más condiciones y de esta manera filtrar más el resultado. Por ejemplo si quisiéramos excluir al director que se llame 'Antonio' la sentencia sería:

```
SELECT sucursales.nombre, directores.nombre,
directores.apellidos FROM sucursales,directores
where sucursales.director=directores.id and directores.nombre<>'Antonio';
```

- Cuando combinamos varias tablas, hemos de tener en cuenta una serie de reglas:
  - Es posible unir tantas tablas como deseemos.
  - En la cláusula SELECT se pueden citar columnas de todas las tablas.
  - Si hay columnas con el mismo nombre en las distintas tablas de la cláusula FROM, se deben identificar, especificando NombreTabla.NombreColumna.
  - Si el nombre de una columna existe sólo en una tabla, no será necesario especificarla como NombreTabla.NombreColumna. Sin embargo, hacerlo mejoraría la legibilidad de la sentencia SELECT.
  - El criterio que se siga para combinar las tablas ha de especificarse en la cláusula WHERE. Si se omite esta cláusula, que especifica la condición de combinación, el resultado será un PRODUCTO CARTESIANO, que emparejará todas las filas de una tabla con cada fila de la otra.

UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS

- Tipos de JOIN: Según la comparación entre los campos de las tablas relacionadas existen dos tipos principales de JOIN
  - NON-EQUI JOIN: Por desigualdad, cuando no existe la correspondencia directa entre campos de tablas. La relación se establecerá usando criterios de rango (=<, =>, <, >, BETWEEN, ...)
  - EQUI JOIN: JOIN sobre dos o más tablas, por igualdad de campos.
- Dentro de los EQUI JOIN hay que distinguir los siguientes tipos:
  - Combinación interna: INNER JOIN, NATURAL JOIN.
  - Combinación externas: LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN.

- Tanto la palabra INNER como la palabra OUTER son opcionales, por lo que es válido utilizar esta nomenclatura:
  - JOIN
  - LEFT JOIN
  - RIGHT JOIN
  - FULL JOIN

- **[INNER] JOIN:** Es el tipo más común de unión. INNER JOIN devuelve todas las filas de varias tablas donde se cumple la condición de unión. La sintaxis para INNER JOIN es:

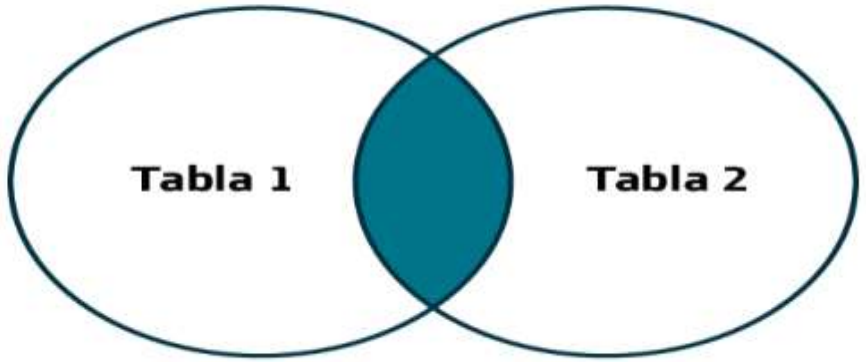
```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

```
use banco;
SELECT *
FROM directores INNER JOIN sucursales
ON directores.id = sucursales.director;
```

```
SELECT *
FROM sucursales, directores
WHERE sucursales.director=directores.id ;
```

Diagrama visual

En este diagrama visual, INNER JOIN devuelve el área sombreada:



Se devolvería los registros donde se unen tabla1 y tabla2 .

- Se devolvería los registros donde se unen tabla1 y tabla2 . Ejemplo: Se necesitan saber los nombres de los directores y a que sucursal están asignados.

```
SELECT directores.nombre, directores.apellidos, sucursales.nombre as NOMBRE_SUCURSAL
FROM directores
INNER JOIN sucursales
ON directores.id=sucursales.director;
```

- **Resultado:** En la siguiente imagen se muestra el resultado obtenido. Como se observa no aparece "Pedro García Martín" porque no está en ambas tablas.

NOMBRE	APELLIDOS	NOMBRE_SUCURSAL
Alberto	Pérez Martín	Sucursal Centro
Antonio	López López	Sucursal Oeste
Silvia	Martín Martín	Sucursal Este

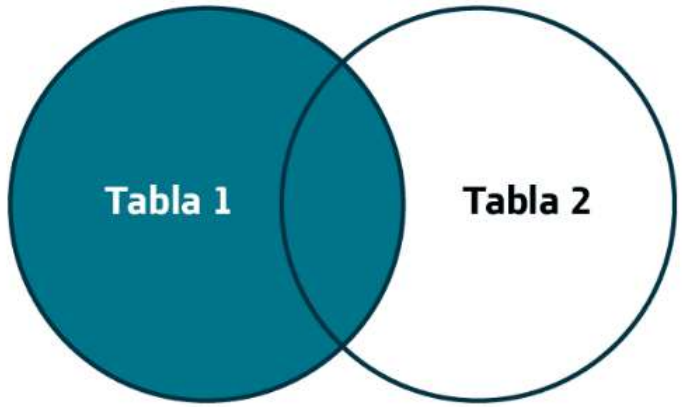
- **LEFT [OUTER] JOIN:** Otro tipo de unión se llama LEFT OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla IZQUIERDA especificada en la condición ON y solo aquellas filas de la otra tabla donde los campos combinados son iguales (se cumple la condición de unión). La sintaxis para LEFT OUTER JOIN es:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

- En este diagrama visual, LEFT OUTER JOIN devuelve el área sombreada:

Diagrama visual

En este diagrama visual, LEFT OUTER JOIN devuelve el área sombreada:



LEFT OUTER JOIN devolverá todos los registros de la tabla 1 y solo aquellos registros de la tabla 2 que se cruzan con la tabla1 .

FORMA SIMULAR EN MYSQL SUCURSALES – NOMBRE:
SELECT DISTINCT sucursales.nombre, directores.nombre, directores.apellidos FROM sucursales LEFT JOIN directores ON sucursales.director=directores.id WHERE directores.id IS NULL;

- Ejemplo: Se necesitan saber los directores y sus sucursales. Mostrar todos los directores incluso si no tienen asignados sucursales.

```
SELECT directores.nombre, directores.apellidos, sucursales.nombre as NOMBRE_SUCURSAL
FROM directores
LEFT JOIN sucursales ON directores.id=sucursales.director;
```

- Resultado: Como se puede observar se listan todos los directores, a pesar de que el director "Pedro García Martín" aún no tiene asignada Sucursal.

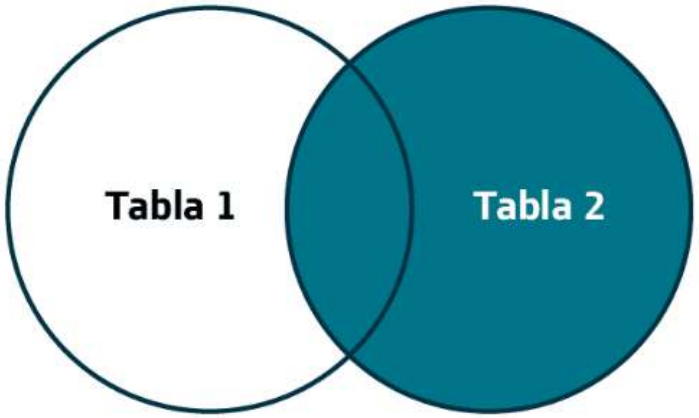
NOMBRE	APELLIDOS	NOMBRE_SUCURSAL
Alberto	Pérez Martín	Sucursal Centro
Antonio	López López	Sucursal Oeste
Silvia	Martín Martín	Sucursal Este
Pedro	Garcia Martín	

- **RIGHT [OUTER] JOIN:** Otro tipo de unión es RIGHT OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla DERECHA especificada en la condición ON y solo aquellas filas de la otra tabla donde los campos combinados son iguales (se cumple la condición de unión). La sintaxis para RIGHT OUTER JOIN es:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2 ON table1.column = table2.column;
```

Diagrama visual

En este diagrama visual, RIGHT OUTER JOIN devuelve el área sombreada:



RIGHT OUTER JOIN devolverá todos los registros de la tabla 2 y sólo aquellos registros de la tabla 1 que se cruzan con la tabla 2.

Forma de simular SUCURSALES –DIRECTORES:
SELECT DISTINCT sucursales.nombre, directores.nombre, directores.apellidos FROM sucursales right JOIN directores ON sucursales.director=directores.id WHERE sucursales.director IS NULL;

- Ejemplo: Se necesitan saber los directores y sus sucursales asociadas.

```
SELECT directores.nombre, directores.apellidos, sucursales.nombre as NOMBRE_SUCURSAL
FROM directores
RIGHT JOIN sucursales ON directores.id=sucursales.director;
```

- Resultado: Como se puede observar se listan todas las sucursales, a pesar que una de ellas no tiene asociado director. Esto es porque el right join lo hemos hecho sobre la tabla sucursales y se incluyen todos sus registros.

NOMBRE	APELLIDOS	NOMBRE_SUCURSAL
Alberto	Pérez Martín	Sucursal Centro
Antonio	López López	Sucursal Oeste
Silvia	Martín Martín	Sucursal Este
		Sucursal Norte

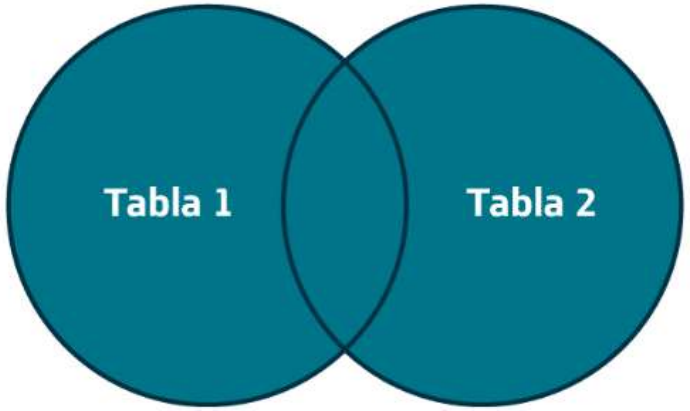
- **FULL [OUTER] JOIN:** Otro tipo de unión es FULL OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla IZQUIERDA y la tabla DERECHA con nulos en el lugar donde no se cumple la condición de unión. La sintaxis para FULL OUTER JOIN es:

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

```
SELECT * FROM t1
LEFT JOIN t2 ON t1.id = t2.id
UNION
SELECT * FROM t1
RIGHT JOIN t2 ON t1.id = t2.id
```

Diagrama visual

En este diagrama visual, FULL OUTER JOIN devuelve el área sombreada:



FULL OUTER JOIN devolverá todos los registros de tabla 1 y tabla 2.

```
-- No funciona EL full outer join y de simularlo
SELECT sucursales.nombre, directores.nombre,
directores.apellidosFROM sucursalesFULL JOIN directoresON
sucursales.director=directores.id ;
```

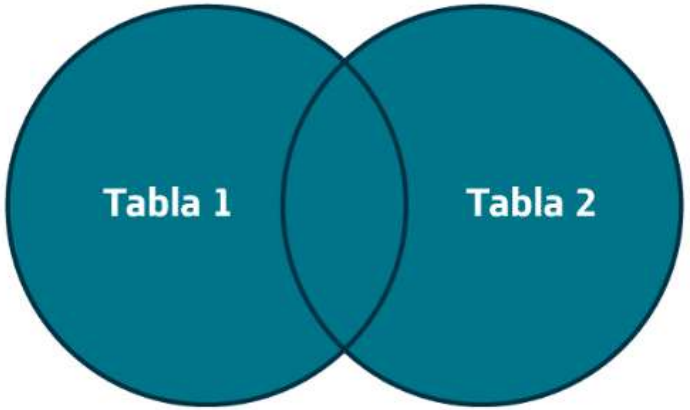
```
-- EL full outer join simulado
SELECT *
FROM sucursales LEFT JOIN directores
ON sucursales.director=directores.id
UNION
SELECT *
FROM directores RIGHT JOIN sucursales
ON sucursales.director=directores.id;
```

- **FULL [OUTER] JOIN:** Otro tipo de unión es FULL OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla IZQUIERDA y la tabla DERECHA con nulos en el lugar donde no se cumple la condición de unión. La sintaxis para FULL OUTER JOIN es:

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

Diagrama visual

En este diagrama visual, FULL OUTER JOIN devuelve el área sombreada:



FULL OUTER JOIN devolverá todos los registros de tabla 1 y tabla 2.

- Ejemplo: Se necesitan saber los directores y sus sucursales asociadas.

```
SELECT sucursales.nombre as NOMBRE_SUCURSAL, directores.nombre, directores.apellidos
FROM sucursales FULL OUTER JOIN directores
ON sucursales.director=directores.id;
```

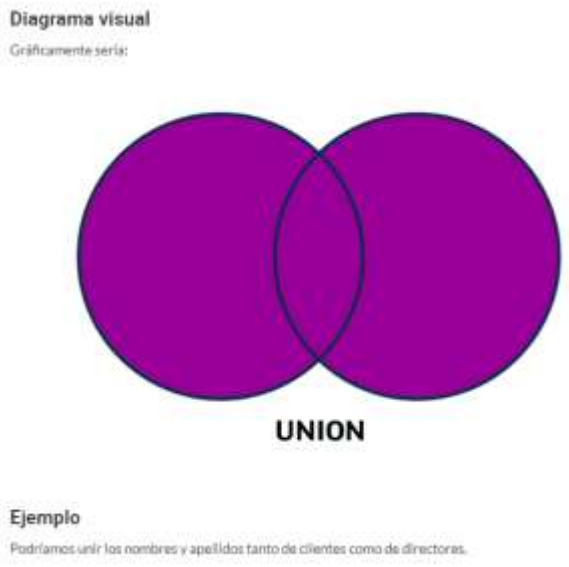
- Resultado: En este caso se muestran todos los directores y sucursales, aunque haya directores sin sucursales y sucursales sin directores.

NOMBRE_SUCURSAL	NOMBRE	APELLIDOS
Sucursal Centro	Alberto	Pérez Martín
Sucursal Oeste	Antonio	López López
Sucursal Este	Silvia	Martín Martín
	Pedro	García Martín
Sucursal Norte		

- **Unión:** Ocasionalmente, es conveniente combinar los resultados de dos o más consultas en una única tabla de resultados totales. SQL permite esta capacidad gracias a la característica UNION de la sentencia SELECT. Se utiliza la palabra reservada UNION, y la sintaxis es:

```
SELECT columna1, columna2 FROM tabla1
UNION
SELECT columna1, columna2 FROM tabla2
```

- El resultado es una tabla que contendrá la columna1 y columna2 de ambas tablas. Es decir, UNION crea una sola tabla con registros que estén presentes en cualquiera de las consultas. Si están repetidas sólo aparecen una vez. Para mostrar los duplicados se utiliza UNION ALL en lugar de la palabra UNION



/ilas

```
SELECT nombre, apellidos FROM directores
UNION
SELECT nombre, apellidos FROM clientes;
```

- **Restricciones:** Hay varias restricciones sobre las tablas que pueden combinarse con una operación UNION:
  - Ambas tablas deben contener el mismo número de columnas.
  - El tipo de datos de cada columna en la primera tabla debe ser el mismo que el tipo de datos de la columna correspondiente en la segunda tabla.
  - Ninguna de las dos tablas puede estar ordenadas con la cláusula ORDER BY. Sin embargo, los resultados combinados pueden ser ordenados, según se describe en la sección siguiente.
  - Los nombres de columna de las dos consultas combinadas mediante una UNION no tienen que ser idénticos. Puesto que las columnas de las dos tablas pueden tener nombres diferentes, lo habitual es mostrar el nombre del campo de la primera consulta, pero depende del gestor de base de datos. Por omisión, la operación UNION **elimina las filas duplicadas** como parte de su procesamiento.
  - La eliminación de filas duplicadas en los resultados de la consulta es un proceso que consume mucho tiempo, especialmente si los resultados contienen un gran número de filas. Si se sabe, en base a las consultas individuales implicadas, que la operación UNION no puede producir filas duplicadas, se debería utilizar específicamente la operación UNION ALL, ya que la consulta se ejecutará mucho más rápidamente.

- **Uniones y ordenación:** La cláusula ORDER BY no puede aparecer en ninguna de las dos sentencias SELECT combinadas por una operación UNION. No tendría mucho sentido ordenar los dos conjuntos de resultados de ninguna manera, ya que éstos se dirigen directamente a la operación UNION y nunca son visibles al usuario.
- Sin embargo, el conjunto combinado de los resultados de la consulta producidos por la operación UNION puede ser ordenado especificando una cláusula ORDER BY después de la segunda sentencia SELECT.
- Uniones múltiples: La operación UNION puede ser utilizada repetidamente para combinar tres o más conjuntos de resultados. De esta manera si tenemos una unión entre una tabla B y una tabla C, la podemos unir y el resultado de esa unión unirla a una tercera tabla A. Tendríamos lo siguiente: A UNION (B UNION C)
- Ejemplo: En este caso le unimos a dos tablas de directores, la tabla de clientes con sus nombres y apellidos.

```
SELECT nombre,apellidos FROM clientes
UNION
SELECT nombre,apellidos FROM directores
UNION
SELECT nombre,apellidos FROM directores1;
```

- Los paréntesis que aparecen en la consulta indican que UNION debería ser realizada en primer lugar. De hecho, si todas las uniones de la sentencia eliminan filas duplicadas, o si todas ellas mantienen las filas duplicadas, el orden en que se efectúan no tienen importancia. Estas tres expresiones son equivalentes:
  - A UNION (B UNION C)
  - (A UNION B) UNION C
  - (A UNION C) UNION B

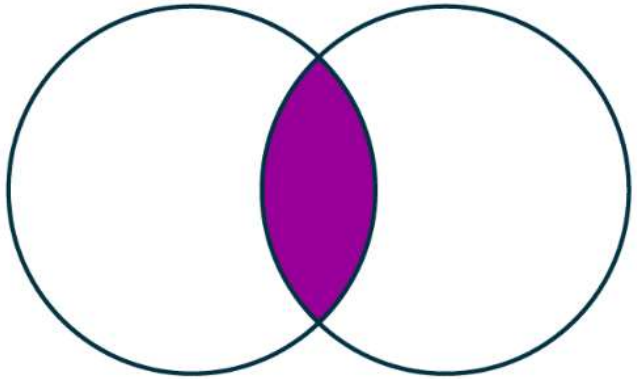
- **Intersect**: INTERSECT devuelve los valores comunes a ambas, con lo que obtenemos una intersección (sólo los registros que están entre los resultados de ambas consultas). La sintaxis sería:

```
SELECT columna1, columna2,... FROM tabla1
INTERSECT
SELECT columna1, columna2,... FROM tabla2
```

- La representación gráfica sería la siguiente:

Diagrama visual

La representación gráfica sería la siguiente:



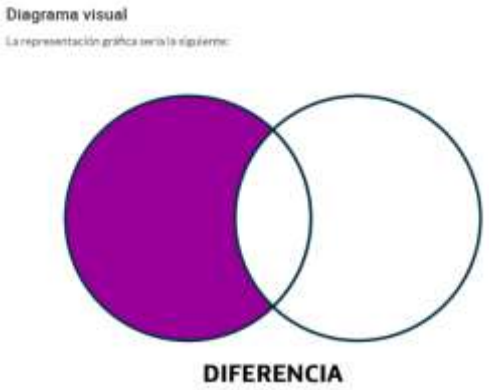
INTERSECCIÓN

- Ejemplo: Obtener los nombres que son comunes entre la tabla directores y clientes.

```
SELECT nombre FROM directores
INTERSECT
SELECT nombre FROM clientes;
```

- Al igual que ocurre con UNION la intersección también puede combinarse con la ordenación (ORDER BY).
- **Minus:** MINUS devuelve los valores de la primera consulta que no se encuentran en la segunda. Así podemos averiguar qué registros están en una consulta pero no en la otra, calculando la diferencia entre dos conjuntos de registros. Algo realmente útil en ocasiones y difícil de conseguir con instrucciones más simples. La sintaxis sería:

```
SELECT columna1, columna2,... FROM tabla1
MINUS
SELECT columna1, columna2,... FROM tabla2
```



- Ejemplo: Listar los nombres de directores que no se llamen igual que los clientes de las sucursales, sin comprobar los clientes de la sucursal 1001:

```
SELECT nombre FROM directores
MINUS
SELECT nombre FROM clientes where nsucursal<>1001;
```


- **Funciones de agrupación:** Hasta ahora hemos visto funciones que operan con valores simples; no obstante, hay otras funciones como SUM, AVG y COUNT, que actúan sobre un grupo de filas para obtener un valor. Estas funciones permiten obtener la edad media de un grupo de alumnos, el alumno más joven, el más viejo, el número total de miembros de un grupo, etc. Los valores nulos son ignorados por las funciones de grupos de valores y los cálculos se realizan sin contar con ellos.
- Estas funciones son:
  - SUM: suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
  - AVG: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
  - MIN: devuelve el valor mínimo del campo que especifiquemos.
  - MAX: devuelve el valor máximo del campo que especifiquemos.
  - COUNT: devuelve el número total de filas seleccionadas por la consulta.

- SUM: La función columna SUM() calcula la suma de una columna de valores de datos. Los datos de la columna deben tener un tipo numérico. El resultado de la función SUM() tiene el mismo tipo de datos básico que los datos de la columna, pero el resultado puede tener una precisión superior. Ejemplo: Suma de los sueldos de los directores:

```
SELECT SUM(sueldo) FROM directores;
```


- **AVG:** La función de columna AVG() calcula el promedio de una columna de valores de datos. Al igual que una función SUM(), los datos de la columna deben tener un tipo numérico. Ya que la función AVG() suma los valores de la columna y luego lo divide por el número de valores, su resultado puede tener un tipo de dato diferente al de los valores de columna.

- Si se aplica la función AVG() a una columna de enteros, el resultado será un número decimal o un número de coma flotante, dependiendo del gestor concreto que se esté utilizando. Ejemplo: Media de los sueldos de los directores

```
SELECT AVG(sueldo) FROM directores;
```


- **MIN Y MAX:** Las funciones de columna MIN() Y MAX() determinan los valores menor y mayor de una columna, respectivamente. Los datos de la columna pueden contener información numérica, de cadena o de fecha/hora. El resultado de la función MIN() y MAX() tiene exactamente el mismo tipo de dato que los datos de la columna. Ejemplo: Fecha de alta más antigua de los clientes

```
SELECT MIN(fecha_de_alta) FROM clientes;
```

- Cuando las funciones de columnas MIN() y MAX() se aplican a datos numéricos, SQL compara los números en orden algebraico (los números negativos grandes son menores que los números negativos pequeños, los cuáles son menores que cero, el cual a su vez es menor que todos los números positivos). Las fechas se comparan secuencialmente (las fechas más antiguas son más pequeñas que las fechas más recientes); cuando se utiliza MIN() YMAX() con datos de cadenas, la comparación de las cadenas depende del conjunto de caracteres que se esté siendo utilizado.

- **COUNT:** La función de columna COUNT() cuenta el número de valores de datos que hay en una columna. Los datos de la columna pueden ser de cualquier tipo. La función COUNT() siempre devuelve un entero, independientemente del tipo de datos de la columna. Ejemplo: Obtener el Número de clientes

```
SELECT count(dni) FROM clientes;
```

- La función COUNT, se suele usar con asterisco (*) como objeto de cálculo, lo que hace que cuente filas independientemente de su contenido. Si usamos una expresión, como COUNT(sueldo), no contaría las filas cuyo sueldo fuera nulo.

- Ejemplo: Si contamos según el campo fecha_de_alta de clientes obtendremos el valor 2 y si lo contamos con * obtendremos el valor 3, porque hay un cliente que no tiene fecha de alta.

```
/* Obtenemos 2 porque hay un cliente que no tiene fecha_de_alta */
SELECT count(fecha_de_alta) FROM clientes;

/* Obtenemos 3 */
SELECT count(*) FROM clientes;
```


- **Consultas agrupadas:** La cláusula GROUP BY unida a un SELECT permite agrupar filas según las columnas que se indiquen como parámetros, y se suele utilizar en conjunto con las funciones de agrupación, para obtener datos resumidos y agrupados por las columnas que se necesiten.

- Las columnas indicadas en la cláusula GROUP BY se denominan columnas de agrupación de la consulta ya que son las que determinan cómo se dividen las filas en grupos.

- Así por ejemplo si queremos saber la suma de créditos que tienen aprobadas las sucursales a sus clientes, lo podríamos hacer así:

```
SELECT nsucursal, SUM(credito)
FROM clientes
GROUP BY nsucursal;
```


- **Restricciones para GROUP BY:** Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula FROM de la consulta. No se pueden agrupar las filas basándose en el valor de una expresión calculada.
- También hay restricciones sobre los elementos que pueden aparecer en la lista de selección de una consulta agrupada. Todos los elementos de la lista de selección deben tener un único valor por cada grupo de filas. Básicamente, esto significa que un elemento de selección en una consulta agrupada puede ser:
  - Una constante.
  - Una función de columna.
  - Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del grupo.
  - Una expresión que afecte a combinaciones de los anteriores.
  - Lo más frecuente es que en una consulta agrupada se incluya siempre una columna de agrupación y una función de agrupación en su lista de selección.

UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS

- Una subconsulta forma parte de una condición de búsqueda en la cláusula WHERE o HAVING.

- SQL ofrece estas condiciones de búsqueda en subconsultas:

  - Test de comparación subconsulta. Compara el valor de una expresión con un valor único producido por una subconsulta. Este test se asemeja al test de comparación simple.
  - Test de pertenencia a conjunto subconsulta. Comprueba si el valor de una expresión coincide con uno del conjunto de valores producido por una subconsulta. Este test se asemeja al test de pertenencia a conjunto simple.
  - Test de existencia. Examina si una subconsulta produce alguna fila de resultados.
  - Test de comparación cuantificada. Compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta.

UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS POR TEST DE COMPARACIÓN

- **Test de comparación subconsulta** (=, <>, <, <=, >, >=): El test de comparación subconsulta es una forma modificada del test de comparación simple. Compara el valor de una expresión producido por una subconsulta, y devuelve un resultado TRUE si la comparación es cierta. Este test se utiliza para comparar un valor de la fila que está siendo examinada con un valor único producido por una subconsulta.

- El test de comparación subconsulta ofrece los mismos seis operadores de comparación (=, <>, <, <=, >, >=) disponibles con el test de comparación simple. La subconsulta especificada en este test debe producir una única fila de resultados. Si la subconsulta produce múltiples filas, la comparación no tiene sentido y SQL informa de una condición de error.

- Ejemplo: Seleccionar el nombre y los apellidos de los clientes cuyo número de sucursal sea distinto del máximo número de sucursal de los clientes dados de alta.

```
SELECT nombre, apellidos
FROM clientes
WHERE nsucursal <> (SELECT MAX(nsucursal)
                    FROM clientes);
```


- **Test de pertenencia a conjunto (IN):** El test de pertenencia a conjunto subconsulta (IN) es una forma modificada del test de pertenencia a conjunto simple. Compara un único valor de datos con una columna de valores producida por una subconsulta y devuelve un resultado TRUE si el valor coincide con uno de los valores de la columna. Se puede usar también negando la pertenencia. En este caso lo indicaríamos con NOT IN.

- La subconsulta produce una columna de valores, y la cláusula WHERE de la consulta principal, comprueba si un valor de una fila de la consulta principal coincide con uno de los valores de la columna.

- Ejemplo: Obtener el nombre y apellido de los clientes que tengan el mismo nombre que algún director:

```
SELECT nombre, apellidos
FROM clientes
WHERE nombre IN (SELECT nombre
                 FROM directores);
```


- **Test de pertenencia a conjunto (IN):** El test de pertenencia a conjunto subconsulta (IN) es una forma modificada del test de pertenencia a conjunto simple. Compara un único valor de datos con una columna de valores producida por una subconsulta y devuelve un resultado TRUE si el valor coincide con uno de los valores de la columna. Se puede usar también negando la pertenencia. En este caso lo indicaríamos con NOT IN.

- La subconsulta produce una columna de valores, y la cláusula WHERE de la consulta principal, comprueba si un valor de una fila de la consulta principal coincide con uno de los valores de la columna.

- Ejemplo: Obtener el nombre y apellido de los clientes que tengan el mismo nombre que algún director:

```
SELECT nombre, apellidos
FROM clientes
WHERE nombre IN (SELECT nombre
                 FROM directores);
```

```
SELECT nombre, apellidos
FROM clientes
WHERE nombre NOT IN (SELECT nombre
                     FROM directores);
```


- **Test de existencia (EXISTS):** El test de existencia (EXISTS) comprueba si una subconsulta produce alguna fila de resultados. No hay test de comparación simple que se asemeje al test de existencia; solamente se utiliza con subconsultas. Se puede usar también negando la existencia. En este caso lo indicaríamos con NOT EXISTS.

- La subconsulta de un test EXISTS siempre contienen una referencia externa que "enlaza" la subconsulta a la fila que actualmente está siendo examinada por la consulta principal.

- Ejemplo: Obtener los nombres de las sucursales cuyo director gane más de 1400 Euros:

```
SELECT nombre
FROM sucursales
WHERE EXISTS (SELECT *
                    FROM directores, sucursales
                    WHERE directores.id=sucursales.director
                    and sueldo>1400);
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS POR TEST CUANTIFICADOS ANY Y ALL

- **Test cuantificados (ANY y ALL):** La versión subconsulta del test IN comprueba si un valor de dato es igual a algún valor en una columna de los resultados de la subconsulta. SQL proporciona dos test cuantificados, ANY y ALL, que extienden esta noción a otros operadores de comparación, tales como mayor que (>) y menor que (<). Ambos tests comparan un valor de dato con la columna de valores producidos por una subconsulta.

- **Test ANY:** El test ANY se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con cada valor de datos de la columna, uno cada vez. Si alguna de las comparaciones individuales producen un resultado TRUE, el test ANY devuelve un resultado TRUE.

- Ejemplo: Listar los nombres de los clientes cuyo crédito es menor a alguno de los créditos de los clientes de la sucursal 1001.

```
SELECT nombre
FROM clientes
WHERE credito < ANY (SELECT credito
                     FROM clientes
                      WHERE nsucursal='1001');
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS POR TEST CUANTIFICADOS ANY Y ALL

- Test ALL: El test ALL se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores de datos producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con todos y cada uno de los valores de datos de la columna. Si todas las comparaciones individuales producen un resultado TRUE, el test ALL devuelve un resultado TRUE.

- Ejemplo: Listar los nombres de los clientes cuyo crédito es menor a todos los créditos de los clientes de la sucursal 1001.

```
SELECT nombre FROM clientes WHERE
credito < ALL (SELECT credito FROM clientes WHERE nsucursal='1001');
```


- **Subconsultas en la cláusula HAVING:** Aunque las subconsultas suelen encontrarse sobre todo en la cláusula WHERE, también pueden utilizarse en la cláusula HAVING de una consulta. Cuando una subconsulta aparece en la cláusula HAVING, funciona como parte de la selección de grupo de filas efectuada por la cláusula HAVING.

- Ejemplo: Obtener el número de sucursal y la media de sus créditos concedidos si la media de la sucursal es mayor a la media del resto de sucursales.

```
SELECT nsucursal, AVG(credito) FROM clientes
GROUP BY nsucursal HAVING AVG(credito) > (SELECT AVG(credito) FROM clientes);
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS EN INSERT

- **Subconsultas en INSERT:** Hay un tipo de consulta, llamada de adición de datos, que permite rellenar datos de una tabla copiando el resultado de una consulta; se hace mediante la instrucción INSERT y ese relleno se basa en una consulta SELECT que poseerá los datos a añadir. Lógicamente el orden de esos campos debe de coincidir con la lista de campos indicada en la instrucción INSERT y la nueva tabla debe estar previamente creada.

```
INSERT INTO tabla (campo1, campo2,...)
SELECT campoCompatibleCampo1, campoCompatibleCampo2,...
FROM lista DeTablas
[...otras cláusulas del SELECT...]
```

- Ejemplo: Insertar en una nueva tabla los clientes que tengan un crédito superior a 1200 Euros.

```
INSERT INTO clientes_credito(dni,nombre,apellidos,direccion)
SELECT dni,nombre,apellidos,direccion
FROM clientes
WHERE credito>1200;
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS EN INSERT

- **Subconsultas en INSERT:** Hay un tipo de consulta, llamada de adición de datos, que permite rellenar datos de una tabla copiando el resultado de una consulta; se hace mediante la instrucción INSERT y ese relleno se basa en una consulta SELECT que poseerá los datos a añadir. Lógicamente el orden de esos campos debe de coincidir con la lista de campos indicada en la instrucción INSERT y la nueva tabla debe estar previamente creada.

```
INSERT INTO tabla (campo1, campo2,...)
SELECT campoCompatibleCampo1, campoCompatibleCampo2,...
FROM lista DeTablas
[...otras cláusulas del SELECT...]
```

- Ejemplo: Insertar en una nueva tabla los clientes que tengan un crédito superior a 1200 Euros.

```
INSERT INTO clientes_credito(dni,nombre,apellidos,direccion)
SELECT dni,nombre,apellidos,direccion
FROM clientes
WHERE credito>1200;
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS EN UPDATE

- **Subconsultas en UPDATE:** La instrucción UPDATE permite modificar filas. Es muy habitual el uso de la cláusula WHERE para indicar las filas que se modificarán. Esta cláusula se puede utilizar con las mismas posibilidades que en el caso del SELECT, por lo que es posible utilizar subconsultas.

- Ejemplo: Esta instrucción aumenta un 10% el crédito de los clientes que son de la Sucursal Centro. En primer lugar se averigua qué código tiene esa sucursal y posteriormente se utiliza para actualizar el cliente.

```
UPDATE clientes SET credito=credito*1.10 WHERE nsucursal=
(SELECT nsucursal FROM sucursales WHERE nombre='Sucursal Centro');
```

- También podemos utilizar subconsultas en la cláusula SET de la instrucción UPDATE: Ejemplo, actualizando el crédito de los clientes de la sucursal 1002 al máximo de los créditos de la sucursal 1001.

```
UPDATE clientes SET credito =
(SELECT max(credito) FROM clientes WHERE nsucursal=1001)
WHERE nsucursal=1002;
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS EN DELETE

- **Subconsultas en DELETE:** Al igual que en el caso de las instrucciones INSERT o SELECT, DELETE dispone de cláusula WHERE y en dicha cláusula podemos utilizar subconsultas.

- Ejemplo: Borrar los clientes que sean de la Sucursal Centro:

```
DELETE clientes WHERE nsucursal IN
(SELECT nsucursal FROM sucursales WHERE nombre='Sucursal Centro');
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS EN DELETE

- **Subconsultas en DELETE:** Al igual que en el caso de las instrucciones INSERT o SELECT, DELETE dispone de cláusula WHERE y en dicha cláusula podemos utilizar subconsultas.

- Ejemplo: Borrar los clientes que sean de la Sucursal Centro:

```
USE BANCO;
DELETE
from clientes
where nsucursal  IN
                                        (SELECT nsucursal
        FROM sucursales
        WHERE nombre='Sucursal Centro');
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS CON SUBCONSULTAS EN CREATE

- Subconsulta en CREATE: También la instrucción CREATE TABLE tiene la capacidad de ser utilizada mediante una subconsulta. Lo que se crea es una tabla que contiene una copia de los datos de la consulta y cuya estructura se basa en función de dichos datos.

```
CREATE TABLE nombreTabla AS

 SELECT ...contenidoConsultaSELECT...;
```

- Ejemplo: Crea una nueva tabla llamada clientes_credito que va a contener el dni, nombre y apellidos de los clientes que tienen crédito superior a 500 Euros:

```
CREATE TABLE clientes_credito AS
(SELECT dni,nombre, apellidos FROM CLIENTES WHERE CREDITO>500);
```


- **Subconsulta como expresión:** Una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar (o una lista de valores de un campo). Las subconsultas que retornan un sólo valor escalar se utilizan con un operador de comparación o en lugar de una expresión:

```
SELECT campo operador (SUBCONSULTA) from TABLA;
```

- Ejemplo: Queremos saber los nombres de los clientes y la diferencia de su crédito respecto al máximos de los créditos de todos los créditos concedidos:

```
SELECT nombre, apellidos, credito - (select max(credito) from clientes)
AS Diferencia FROM clientes;
```


UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS

- Fases del procesamiento de una consulta: Una sentencia SQL no se ejecuta en una única operación, sino que el SGBD necesita dividirla en varias fases. La mayoría de estas fases requieren de la generación de una relación o tabla virtual, es decir, una tabla temporal generada exclusivamente para llevar a cabo un paso intermedio, que luego será destruida.

- Observa esta consulta:

```
SELECT Procedencia AS Pais, COUNT(Articulo) AS NumArticulos
FROM Articulo
WHERE CATEGORIA IN ('Pescado/Marisco','Condimentos','Bebidas','Lácteos')
GROUP BY Procedencia
HAVING COUNT(Articulo)>1
ORDER BY COUNT(Articulo);
```

- La pregunta sería: ¿En qué orden se ejecutan cada una de las partes de esa sentencia SQL?, y por otro lado, ¿qué coste tiene para el SGBD cada parte?

- El orden de procesamiento de una consulta de este tipo siempre será el mismo, si bien podría variar un poco en función del SGBD utilizado. La consulta SQL ejecutada con MySQL tendría las siguientes etapas:

- FROM: Lo primero que hace el SGBD es comprobar las tablas implicadas en la consulta que se ha de ejecutar.

- WHERE: Luego realiza los filtros necesarios sobre las filas implicadas obteniendo una primera relación virtual.

- SELECT: Aquí es donde se realiza la proyección de las columnas que realmente deseamos. A partir de ellas se modifica la primera relación, resultando una segunda relación virtual. También es en este punto donde se aplican los alias (AS) de las columnas.

- GROUP BY: Agrupa los resultados por el SELECT. Recuerda que todas las columnas que no estén afectadas por una función de agregado (COUNT, SUM, AVG, MAX, MIN, etc.) tendrán que estar aquí incluidas. Se genera una tercera relación virtual con los resultados ya agrupados.

- HAVING: Realiza un filtro sobre la tercera relación virtual, es decir, la relación de datos ya agrupados. Aquí es común realizar el filtro utilizando como criterio las consultas resultantes de una función de agregado (COUNT, SUM, AVG, MAX, MIN, etc.).

- ORDER BY: Se ordenan los resultados finales.

- Por lo tanto si hubiera un error en las tablas indicadas en el FROM y también en los campos indicados en el WHERE, el SGBD mostraría el primer error que encuentre y que en este caso es el de la tabla.

- Si nuestra consulta está basada en más de una tabla no está tan claro en qué orden accederá el SGBD a cada una de ellas. Podemos utilizar el comando EXPLAIN sobre una sentencia SQL para averiguar los pasos que sigue el SGBD. Lo veremos en el siguiente apartado.
- EXPLAIN: El comando EXPLAIN usado sobre una sentencia SQL permite averiguar los pasos que seguirá el SGBD para ejecutar dicha consulta.

- Esto resulta muy útil para analizar las fases que conlleva dicha consulta y optimizarla si es posible. Podemos obtener información como:

- El orden de las tablas a las que hace referencia la sentencia SQL.
- El método de acceso para cada tabla indicada en la sentencia
- El método de combinación para las tablas afectadas por operaciones de combinación en la sentencia.
- Operaciones de datos como filtro, clasificación o agregación.

- Cada SGBD tiene su propia manera de gestionar esta información. En este caso vamos a estudiar como se gestiona en ORACLE.

- EXPLAIN PLAN FOR

- EXPLAIN PLAN FOR es el comando de Oracle SQL que indica cual es el plan de consulta para una sentencia SQL determinada.

- EXPLAIN PLAN FOR: Por ejemplo si queremos ver el plan de consulta para una sentencia que nos muestre las sucursales con los nombres de los directores ordenados por número de sucursal, excluyendo la sucursal 1002, tendríamos que ejecutar la siguiente sentencia:

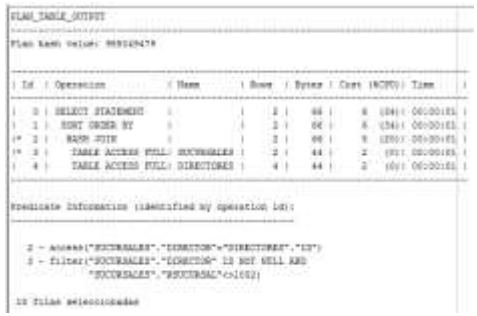
```
EXPLAIN PLAN FOR SELECT nsucursal,sucursales.nombre, directores.nombre
FROM sucursales JOIN directores
ON sucursales.director=directores.id
where nsucursal<>1002
order by nsucursal;
```

- Una vez ejecutada la sentencia el plan de consulta se almacena en una tabla especial de ORACLE llamada PLAN_TABLE. Para ver su contenido de manera formateada podemos utilizar la sentencia:

```
SELECT * FROM TABLE( dbms_xplan.display);
```


- Se muestran las operaciones realizadas y en qué orden. También podemos ver el coste en CPU y en tiempo. Entre los campos principales de la tabla PLAN_TABLE y que se muestran en el resultado obtenido son:

- Id: Identificador de cada uno de los pasos del plan.
- Operation: Operación realizada. Contiene SELECT, INSERT, UPDATE ó DELETE, u operaciones de ordenación, acceso a tablas, etc.
- Name: Nombre de los objetos de la base de datos a los que accede. En este caso muestra las dos tablas de la que consta la consulta SELECT.
- Rows: Filas a las que se accede en el objeto indicado
- Bytes: Bytes leídos
- Cost (%CPU): Coste relativo de la etapa.
- Time: Tiempo que conlleva la etapa.
- Un estudio detallado de todos estos aspectos y sobre todo el orden en el que se ejecutan cada una de las partes de la sentencia nos puede llevar a una optimización de la consulta.



- **Rendimiento de las consultas SQL: U**na vez que ya conocemos cómo se procesa una sentencia SQL en el SGBD es importante seguir unas pautas para obtener un mejor rendimiento de nuestra base de datos.

- Ten en cuenta que en los ejemplos con los que estamos trabajando estamos utilizando unos pocos de registros, pero lo normal es trabajar en una base de datos con miles de registros.

- La degradación del rendimiento se produce a menudo si el mantenimiento de la base de datos no es el adecuado o si las consultas se pueden reescribir de forma más eficaz. Por ello es muy importante la optimización de las consultas.

- Es importante seguir los procedimientos recomendados de SQL para garantizar la optimización de consultas:

- Indexa todos los predicados en cláusulas JOIN, WHERE, ORDER BY y GROUP BY

- Se recomienda indexar todas las columnas de predicados. La excepción es cuando los datos de columna tienen una cardinalidad muy baja.

- Evita el uso de funciones en los predicados:

- La base de datos no utiliza el índice si hay una función en la columna. Por ejemplo:

```
SELECT * FROM table1 WHERE UPPER(col1)='ABC’;
```

- Evita el uso del carácter comodín (%) al principio de un predicado

- El predicado LIKE '%abc' produce una exploración de tabla completa. Por ejemplo:

```
SELECT * FROM table1 WHERE col1 LIKE '%ABC';
```


- Evita columnas innecesarias en la cláusula SELECT

- Especifica las columnas en la cláusula SELECT en lugar de utilizar SELECT *. Las columnas innecesarias imponen cargas adicionales en la base de datos, lo que ralentiza no sólo el SQL específico, sino todo el sistema.

- Utiliza la unión interna en lugar de la unión externa, si es posible. La unión externa sólo debe utilizarse si es necesario. La utilización de la unión externa limita las opciones de optimización de la base de datos, lo que suele dar como resultado una ejecución más lenta del SQL.

- UNION sólo deben utilizarse si es necesario y utiliza UNION ALL en lugar de UNION, si es posible, ya que es mucho más eficaz, porque no tiene que comprobar las filas repetidas

- Impacto al rendimiento de la cláusula ORDER BY: Ten en cuenta el impacto del rendimiento que supone añadir la cláusula ORDER BY, ya que la base de datos necesita clasificar el conjunto de resultados, lo que se convierte en una de las operaciones más costosas de la ejecución de SQL.

- Consultas basadas en más de una tabla: Utiliza INNER JOIN en lugar de WHERE para obtener una relación basada en dos o más tablas.

UF-4 FORMULACIÓN DE CONSULTAS AVANZADAS

- Diferencia entre dos relaciones: Para obtener la diferencia entre dos relaciones R1-R2 utiliza LEFT JOIN en lugar de una consulta anidada.

- Intersección entre dos relaciones: Para la intersección entre dos relaciones, utiliza INNER JOIN en lugar de consultas anidadas.

- Filtrar primero por los criterios más restrictivos: Así se reducen el número de tuplas antes de efectuar los siguientes filtros:

  - DISTINCT en lugar de GROUP BY

  - Usa DISTINCT en lugar de GROUP BY cuando sea posible.

  - Realiza las selecciones o filtros tan pronto como sea posible

  - Realiza los filtros, si es posible, por campos indexados