CAPÍTULO 8°: SQL (RECUPERACIÓN DE DATOS)

1 ***La sentencia SELECT*** **.-**

La sentencia SELECT recupera datos de una base de datos y los devuelve en forma de resultados de la consulta. La sentencia SELECT, en su formato completo está compuesta por las seis cláusulas siguientes:

- SELECT.- Lista los datos a recuperar. Los ítems pueden ser columnas de una o varias Relaciones de la base de datos o columnas a calcular por SQL cuando se efectúe la consulta:

- FROM.- Lista las Relaciones que contienen los datos a recuperar por la consulta

- WHERE.- Indica a SQL las tuplas de datos a incluir en los resultados de la consulta. Se utiliza una condición de búsqueda para especificar las tuplas deseadas

- GROUP BY.- especifica una consulta resumen. Agrupa todas las tuplas similares y produce una tupla resumen de los resultados de cada grupo.

- HAVING.- Indica a SQL la inclusión únicamente ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE utiliza una condición de búsqueda para especificar los grupos deseados.

- ORDER BY.- Ordena los resultados de la consulta utilizando como criterio los datos de una o más columnas.

2 ***Consultas De Relación Única***.-

2.1 ***Consultas Sencillas***.-

Seleccionan las columnas de la Relación que se indiquen en la lista de selección:

SELECT Campo1, Campo2 FROM Relación

El resultado de la consulta contiene todos los datos de la Relación Relación correspondientes a los atributos Campo1 y Campo2.

SELECT * FROM Relación

El resultado de la consulta contiene todos los datos de la Relación **Relación** correspondientes a todos sus atributos.

2.2 ***Columnas Calculadas***.-

SQL puede incluir columnas cuyos valores se calculan a partir de los datos almacenados Pueden ser sumas, restas, multiplicaciones o divisiones (caso de datos almacenados numéricos) o concatenaciones (para datos almacenados alfanuméricos), así como funciones internas. Permite la utilización de paréntesis para la construcción de expresiones más complejas.

Numérico:

SELECT Campo1, Campo2, (Campo3 * Campo4) FROM Relación

Literales:

SELECT Campo1, Campo2 (Campo3 & Campo4) FROM Relación

Funciones internas:

SELECT Campo1, MONTH(Fecha1), YEAR(Fecha2) FROM Relación

2.3 ***Tuplas Duplicadas***.-

Pueden eliminarse las tuplas duplicadas en el resultado de una consulta utilizando la palabra reservada DISTINCT en la sentencia SELECT justo antes de la lista de selección de campos. Conceptualmente SQL genera primero el conjunto completo de resultados y elimina luego las tuplas que son duplicados exactos de alguna otra para formar los resultados finales.

SELECT DISTINCT Campo1, Campo2 FROM Relación

Si se omite la palabra clave DISTINCT, SQL no elimina las tuplas duplicadas. Si se quiere indicar explícitamente que las tuplas duplicadas sean incluidas se utiliza la palabra clave ALL aunque es innecesaria dado que éste es el comportamiento de SQL por omisión.

SELECT ALL Campo1, Campo2 FROM Relación

2.4 ***Selección de tupla***.-

La cláusula WHERE especifica el criterio o condición de búsqueda para seleccionar únicamente las tuplas de la Relación que sean consistentes con dicho criterio.

SELECT Campo1, Campo2 FROM Relación WHERE Condición de Búsqueda

Conceptualmente, la selección de las tuplas mediante la condición de búsqueda produce un resultado de uno de los tres tipos siguientes: TRUE (Verdadero), FALSE (falso) o NULL (nulo).

Únicamente cuando el resultado es verdadero (TRUE) la tupla se incluirá en el resultado de la consulta.

2.5 ***Tests Condición de Búsqueda.***(WHERE)**. -**

El Criterio o Condición de Búsqueda puede ser:

.- Test de Comparación(= , <> , < , > , <= , >=):

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 = Valor (Constante o Calculado)

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 <> Valor (Constante o Calculado)

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 < Valor (Constante o Calculado)

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 > Valor (Constante o Calculado)

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 <= Valor (Constante o Calculado)

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 >= Valor (Constante o Calculado)

.- Test de Rango (BETWEEN):

Establece la condición de búsqueda mediante un rango y comprueba si el valor de los datos se encuentra comprendido en el intervalo especificado:

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 BETWEEN Valor1 AND Valor2

La sentencia anterior es equivalente a:

SELECT Campo1, Campo2 FROM Relación

WHERE (Campo3 >= Valor1) AND (Campo3 <= Valor2)

.- *Test de pertenencia a conjunto (IN)*:

Establece la condición de búsqueda examinando si el dato coincide con uno de una lista de valores objetivo:

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 IN (Valor1, Valor2)

La sentencia anterior es equivalente a:

SELECT Campo1, Campo2 FROM Relación

WHERE (Campo3 = Valor1) OR (Campo3 = Valor2)

.- *Test de correspondencia con patrón* (LIKE):

Permite recuperar las tuplas en las que el contenido de un campo de texto se corresponde con un cierto texto particular:

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 LIKE 'texto'

Pueden utilizarse caracteres comodines:

% se corresponde con cualquier secuencia de cero o más caracteres

_ se corresponde con cualquier carácter simple

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 LIKE 't%'

Selecciona todas aquellas tuplas en las que la cadena de caracteres del campo Campo3 empiece por t

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 LIKE '%t'

Selecciona todas aquellas tuplas en las que la cadena de caracteres del campo Campo3 acabe por t

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 LIKE '%t%'

Selecciona todas aquellas tuplas en las que la cadena de caracteres del campo Campo3 contenga alguna t

Dada la posibilidad de que los caracteres comodines (% y _) sean utilizados como caracteres reales dentro de un dato, deben utilizarse caracteres de escape, definidos mediante la cláusula ESCAPE, según:

SELECT Campo1, Campo2 FROM Relación

WHERE Campo3 LIKE 'A$%BC%' ESCAPE '$'

en la que se indica que el primer signo de porcentaje en el patrón que sigue a un carácter de escape (es este caso $) es tratado como un literal , el segundo signo de porcentaje actúa como un comodín

.- *Test de valor nulo (IS NULL)*:

Establece la posibilidad de considerar valores nulos para un determinado atributo. Dado que una condición de búsqueda puede ofrecer tres valores: verdadera, falsa o desconocida (null) por lo cabe la posibilidad de gestionar dichos valores nulos mediante las palabras reservadas IS NULL o IS NOT NULL. Así:

SELECT Campo1, Campo2 FROM Relación

WHERE Campo1 IS NULL

Muestra aquellas tuplas de Relación en las que el valor del Campo1 es desconocido (Nulo) y

SELECT Campo1, Campo2 FROM Relación

WHERE Campo1 IS NOT NULL

Muestra aquellas tuplas de Relación en las que el valor del Campo1 está perfectamente determinado (No Nulo)

Debe tenerse en cuenta que Null no representa un valor, ni numérico ni literal, por lo que expresiones del tipo Campo1 = NULL carecen de sentido.

2.6 ***Condiciones de Búsqueda compuestas.*** **–**

Pueden utilizarse distintas condiciones de búsqueda, utilizando las reglas de la lógica, concatenando condiciones sencillas mediante las palabras clave AND, OR y NOT.

OR se utiliza para combinar dos condiciones de búsqueda cuando al menos una de ellas deba ser cierta.

AND se utiliza para combinar dos condiciones de búsqueda cuando ambas deban ser ciertas simultáneamente.

NOT se utiliza para seleccionar filas en donde la condición de búsqueda deba ser falsa.

2.7 ***Ordenación de las tuplas seleccionadas*** (ORDER BY). **–**

SQL permite ordenar las tuplas seleccionadas tomando como criterio un atributo o una serie preferencial de atributos mediante la cláusula ORDER BY.

Así:

SELECT Campo1, Campo2,Campo3 FROM Relación

ORDER BY Campo1, Campo3

Ordena las tuplas de la tabla Relación en orden ascendente por el atributo Campo1 y, en caso de igualdad en Campo1, por orden ascendente en Campo3.

La forma de ordenación de omisión es la ascendente pero puede considerarse una ordenación descendente utilizando la palabra reservada DESC:

SELECT Campo1, Campo2 FROM Relación

ORDER BY Campo1, DESC

2.8 ***Combinación de los resultados de una consulta*** (UNION). **–**

Pueden combinarse los resultados de dos o más consultas en una única tabla de resultados finales mediante la cláusula UNION.

La operación UNION produce una única tabla de resultados que combina las filas de la primera consulta con las filas de los resultados de la segunda consulta. La sentencia que especifica la UNION tiene el siguiente aspecto:

SELECT CampoA1, CampoA2 FROM RelaciónA

WHERE CriterioA

UNION

SELECT CampoB1, CampoB2 FROM RelaciónB

WHERE CriterioB

Para poder combinarse dos tablas mediante la operación UNION deben tenerse en cuenta las siguientes restricciones:

- Ambas tablas deben tener el mismo número de columnas

- El tipo de datos de cada columna de la primera tabla ha de ser el mismo tipo de datos de la columna correspondiente de la segunda tabla.

- Ninguna de las dos tablas deben estar ordenadas mediante la cláusula ORDER BY aunque si pueden estar ordenada la tabla final de resultados, tomando como criterio el orden relativo de presentación de los campos en el resultado:

SELECT CampoA1, CampoA2 FROM RelaciónA

WHERE CriterioA

UNION

SELECT CampoB1, CampoB2 FROM RelaciónB

WHERE CriterioB

ORDER BY 1, 2

- No se permiten expresiones en las listas de selección

- Contrariamente a como actúa la cláusula SELECT, la cláusula UNION por omisión elimina las filas duplicadas por lo que si se quiere explícitamente que dichas filas duplicadas aparezcan es preciso utilizar la palabra ALL:

SELECT CampoA1, CampoA2 FROM RelaciónA

WHERE CriterioA

UNION ALL

SELECT CampoB1, CampoB2 FROM RelaciónB

WHERE CriterioB

Por último, es posible realizar uniones de mas de dos tablas, siempre y cunda cumplan las restricciones anteriormente descritas:

SELECT CampoA1, CampoA2 FROM RelaciónA

WHERE CriterioA

UNION

(SELECT CampoB1, CampoB2 FROM RelaciónB

WHERE CriterioB

UNION

(SELECT CampoC1, CampoC2 FROM RelaciónC

WHERE CriterioC

UNION

SELECT CampoD1, CampoD2 FROM RelaciónD

WHERE CriterioD))

Los paréntesis indican qué UNION debe realizarse en primer lugar. Si todas las Uniones eliminan filas duplicadas o todas las Uniones retienen filas duplicadas el orden no tiene importancia. Así:

A UNION (B UNION C)

(A UNION B) UNION C

(A UNION C) UNION B

son expresiones equivalentes. Lo mismo ocurre con:

A UNION ALL(B UNION ALL C)

(A UNION ALL B) UNION ALL C

(A UNION ALL C) UNION ALL B

Pero es preciso respetar el orden si se eliminan o se retienen parcialmente filas duplicadas.

3 ***Consultas multitabla (Composiciones)***.-

Se definen como tal a aquellas consultas que solicitan datos procedentes de dos o mas tablas de la base de datos. SQL permite recuperar datos que responden a estas peticiones componiendo (JOIN) los datos procedentes de dichas tablas.

3.1 ***Composiciones. –***

Puesto que SQL gestiona las consultas multitabla mediante comparación de columnas de las distintas tablas, la sentencia SELECT deberá contener una condición de búsqueda que especifique la comparación de las columnas. A estas columnas de comparación se denominan columnas de emparejamiento para las dos tablas:

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA, TablaB

WHERE Campo1A = Campo1B
```

La consulta obtenida presenta sólo los pares de filas correspondientes a las dos tablas, en los que Campo1A de la primera tabla coincide con Campo1B de la segunda tabla.

La condición de búsqueda especificada por las columnas de emparejamiento en una consulta multitabla puede combinarse con otras condiciones de búsqueda para restringir el contenido de los resultados:

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA, TablaB

WHERE Campo1A = Campo1B

AND Campo2A > Campo2B
```

En general, para una consulta multitabla formada por N tablas habrá que especificar N-1 condiciones de búsqueda relativas a las correspondientes columnas de emparejamiento.

Un caso particular es la composición de una tabla consigo misma (Equicomposición)

En la sentencia SELECT, sin embargo, no se hace mención del "direccionamiento", esto es, con qué tabla debe SQL empezar a trabajar. Para eliminar esta ambigüedad el estándar para SQL2 provee de la cláusula JOIN para indicar que tabla es la subordinada y que tabla es la principal. La sintaxis de la sentencia SELECT es ahora:

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA JOIN TablaB

ON Campo1A = Campo1B
```

3.1.1 *Composiciones Internas.-*

En general, por defecto, SQL realiza enlaces internos (INNER) entre tablas, esto es, enlaces en los que si no se cumplen las dos partes de la igualdad en la condición de búsqueda determinada por las columnas de emparejamiento la pareja de datos resultante no aparece en el resultado. Esto es:

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA INNER JOIN TablaB

ON Campo1A = Campo1B
```

Ofrecerá exclusivamente como resultado las tuplas formadas por Campo1A, Campo2A, Campo1B y Campo2B pertenecientes Campo1A y Campo2A a la TablaA y Campo1B y Campo2B a la TablaB tales que el valor de Campo1A sea exactamente igual al valor de Campo1B

3.1.2 *Composiciones externas.-*

Sin embargo, puede ser interesante considerar en el resultado aquellas tuplas en las que bien Campo1A sea nulo, bien Campo1B sea nulo o bien sean nulos los dos simultáneamente. Se producirá entonces una composición externa (OUTER), cuya sintaxis es la siguiente:

a) Que se consideren los valores nulos de Campo1A y Campo1B simultáneamente o de cualquiera de ellos:

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA OUTER JOIN TablaB

ON Campo1A = Campo1B
```

b) Que se consideren todos los valores de Campo1A , esto es de la columna de emparejamiento de la tabla principal o tabla situada a la izquierda (LEFT) de la cláusula JOIN, independientemente de que existan o no registros de emparejamiento en la tabla subordinada (Campo1B puede ser nulo):

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA LEFT OUTER JOIN TablaB

ON Campo1A = Campo1B
```

c) Que se consideren todos los valores de Campo1B , esto es de la columna de emparejamiento de la tabla subordinada o tabla situada a la derecha (RIGHT) de la cláusula JOIN, independientemente de que existan o no registros de emparejamiento en la tabla principal (Campo1A puede ser nulo):

```
SELECT Campo1A, Campo2A, Campo1B, Campo2B

FROM TablaA RIGHT OUTER JOIN TablaB

ON Campo1A = Campo1B
```

4 ***Consultas Resumen*** .-

Se realizan consultas resumen cuando no se requiere un nivel de detalle proporcionado por las consultas descritas anteriormente sino mas bien un valor único o un pequeño número de valores que resuman el contenido de la base de datos. Para ello SQL suministra las cláusulas GROUP BY y HAVING que permiten realizar estas agregaciones de datos.

4.1 ***Funciones de columna. -***

Una función de columna SQL acepta una columna entera de datos como argumento y produce un único dato que resume la columna. Son funciones de columna:

SUM()	Calcula el valor total de una columna
AVG()	Calcula el valor promedio de una columna
MIN()	Calcula el valor mínimo de una columna
MAX()	Calcula el valor máximo de una columna
COUNT()	Cuenta el número de valores de una columna

COUNT(*) Cuenta las filas de una columna

El argumento de la función puede ser tanto el nombre de una columna como una expresión SQL:

```
SELECT SUM(Campo1A) AVG(Campo1A/Campo2A)

FROM Tabla.
```

Las funciones de columna, excepto COUNT(*) ignoran la existencia de valores nulos.

Las consultas resumen con funciones de columna admiten lógicamente condiciones de búsqueda:

```
SELECT SUM(Campo1A) AVG(Campo1A/Campo2A)

FROM Tabla.

WHERE Campo3A > (Campo1A * Campo1B)
```

4.2 ***Eliminación de filas duplicadas*** (DISTINCT). –

Las consultas resumen pueden eliminar filas duplicadas del resultado de la selección antes de aplicar una función de columna. La sintaxis de la expresión es:

```
SELECT COUNT(DISTINCT Campo1A)

FROM Tabla.
```

en la que se pide que se cuente cuantos valores distintos hay en la columna Campo1A.

4.3***Consultas agrupadas*** (GROUP BY). –

Las consultas resumen definidas hasta ahora suministran los valores totales de una columna. Puede interesar determinar los valores subtotales de la columna según el criterio de agrupación suministrado por otra columna. La cláusula GROUP BY de la sentencia SELECT proporciona esta capacidad:

```
 SELECT AVG(Campo2A)

FROM TablaA

GROUP BY Campo2A
```

SQL por sí mismo no permite obtener resultados a la vez detallados y resumen en una consulta simple. Para obtener resultados detallados con subtotales o para obtener subtotales multinivel es preciso utilizar SQL programado y calcular los subtotales dentro de la lógica del programa.

No obstante, SQL-Server suple esta limitación añadiendo una cláusula COMPUTE al final de la sentencia SELECT:

SELECT Campo1, Campo2, Campo3

FROM Tabla

ORDER BY Campo1, Campo2

COMPUTE SUM(Campo3) BY Campo1, Campo2

COMPUTE SUM(Campo3), AVG(Campo3) BY Campo1

Calcula subtotales para cada valor de Campo1 con relación a Campo2 y subtotales de Cada valor de Campo2 con relación a Campo3.

Las consultas agrupadas están sujetas a limitaciones bastante estrictas:

* Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula FROM de la consulta.

* No pueden agruparse filas basándose en el valor de una expresión calculada.

* Todos los elementos de la lista de selección deben tener un único valor para cada grupo de filas. Esto implica que un elemento de selección en una consulta agrupada puede ser:

- Una constante

- Una función de columna que produce un único valor que resume las filas del grupo

- 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

En la práctica, una consulta agrupada incluirá siempre una columna de agrupación y una función de columna en su lista de selección.

Por último cabe señalar que, en el estándar SQL considera que dos valores NULL son iguales a efectos de la cláusula GROUP BY.

4.4 ***Condiciones de búsqueda de grupos*** (HAVING). -

Al igual que la cláusula WHERE puede ser utilizada para seleccionar y rechazar filas individuales que participan en una consulta, la cláusula HAVING permite seleccionar o rechazar grupos de filas. Su formato es análogo al de la cláusula WHERE, especificando la condición de búsqueda después de la palabra HAVING:

```
SELECT AVG(Campo2A)

FROM TablaA

GROUP BY Campo2A

HAVING AVG(Campo2A) > 30000
```

La cláusula HAVING especifica por tanto una condición de búsqueda para grupos y, consecuentemente la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de a filas individuales. Esto significa que un elemento que aparezca en la cláusula HAVING puede ser:

- Una constante

- Una función de columna que produce un único valor que resume las filas del grupo

- 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

En la práctica, la condición de búsqueda de la cláusula HAVING incluirá siempre al menos una función de columna ya que, si no lo hace, se aplicaría a filas individuales y sería suficiente una cláusula WHERE.

5 ***Subconsultas***. -

SQL permite utilizar los resultados de una consulta como parte de otra. Una subconsulta es una consulta que aparece dentro de la cláusula WHERE o la cláusula HAVING de otra sentencia SQL.

5.1 ***Subconsultas en la cláusula WHERE.*** -

Cuando aparece una subconsulta en la cláusula WHERE ésta funciona como parte del proceso de selección de filas, ésto es, forman siempre parte de la condición de búsqueda, por ello tiene sentido establecer tests de comparación para subconsultas al igual que se hizo para consultas simples.

Además del test de comparación (= , <> , < , > , <= , >=) y del test de pertenencia a un conjunto (IN), se utiliza el *test de existencia* (EXISTS) que determina la existencia o no de alguna fila en la tabla o tablas objeto de la subconsulta y cumplan la condición requerida en la subconsulta:

```
SELECT Campo1A, Campo2A

FROM TablaA

WHERE  Campo1A = (SELECT Campo1B, Campo2B FROM TablaB

    WHERE Campo1B = Valor AND          NOT EXISTS (SELECT * FROM TablaC

        WHERE Campo1C = Valor2 ))
```

En la práctica, la subconsulta en un test EXIST se escribe siempre utilizando la notación SELECT *.

Otros tests de búsqueda utilizados son los test cuantificados (ALL y ANY). Estos tipos de test comparan un valor de dato con la columna de valores producidos por una subconsulta mediante un operador de comparación (= , <> , < , > , <= , >=).

```
SELECT Campo1A FROM TablaA

WHERE Campo1A > ANY (SELECT Campo1B FROM tablaB

    WHERE Campo1C = Valor)
```

Selecciona aquellas filas de la TablaA en las que el valor de Campo1A sea mayor que ALGUN valor de Campo1B de la TablaB correspondiente a las filas en las que Campo1C sea igual a un valor.

```
SELECT Campo1A FROM TablaA

WHERE Campo1A > ALL (SELECT Campo1B FROM tablaB

    WHERE Campo1C = Valor)
```

Selecciona aquellas filas de la TablaA en las que el valor de Campo1A sea mayor que TODOS los valores de Campo1B de la TablaB correspondientes a las filas en las que Campo1C sea igual a un valor.
