EL LENGUAJE SQL

6.1.- Concepto de SQL

SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos. Es la abreviatura de *Structured Query Language* (Lenguaje Estructurado de Consultas) y funciona con un tipo de bases de datos específico: Las bases de datos relacionales. SQL, además de ser una herramienta de consulta y recuperación de datos, se utiliza para controlar todas las funciones que suministra un Sistema Gestor de Bases de Datos Relacionales a sus usuarios, incluyendo:

Definición de datos.- Permite que el usuario defina la estructura y organización de los datos almacenados, así como las relaciones entre ellos.

Recuperación de datos.- Permite al usuario o a un programa recuperar y utilizar los datos almacenados en una base de datos.

Manipulación de datos.- Permite al usuario o a un programa actualizar la base de datos añadiendo datos nuevos, borrando los viejos y modificando los almacenados previamente.

Control de acceso.- Puede ser utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo los datos almacenados contra accesos no autorizados.

Compartición de información.- Se utiliza para coordinar la compartición de datos entre usuarios concurrentes, asegurando que no haya interferencias entre ellos.

Integridad de los datos.- Define restricciones de integridad en la base de datos, protegiéndola de alteraciones debidas a actualizaciones inconsistentes o fallos del sistema

6.2.- Elementos componentes de una sentencia de SQL

SQL es el lenguaje que se utiliza para pedir al SGBD que realice las operaciones deseadas sobre las tablas. Las sentencias de este lenguaje contienen los siguientes componentes:

1.- ***Palabras predefinidas***.- Palabras que tienen un significado predefinido en el lenguaje SQL (SELECT, WHERE, INTO, etc.)

2.- *Nombres de tablas* (Relaciones) y *columnas* (atributos).

3.- ***Constantes.-*** También llamadas literales, son secuencias de caracteres que representan un valor determinado. Cuando éste no es un número, debe de in entre apóstrofes (').

4.- ***Signos delimitadores.-*** Signos especiales que aparecen en las sentencias fuera de otros elementos. Sirven para delimitar dichos elementos cuando no van entre comillas o apóstrofes. El mas utilizado es el espacio en blanco, pero también actúan como delimitadores paréntesis, comas, signos de operaciones aritméticas o de comparación.

Las palabras predefinidas y los nombres de tablas y columnas pueden escribirse tanto en mayúsculas como en minúsculas o en una mezcla de ambas, si bien es costumbre utilizar siempre mayúsculas

6.3.- Tipos de Sentencias en SQL

De acuerdo con el tipo de operación que expresan, las sentencias en SQL pueden clasificarse en:

1.-*Sentencias de Manipulación de Datos (Data Manipulation Language).*- Permiten realizar consultas u mantenimiento de datos. Son:

SELECT.- Permite extraer datos de una o varias tablas. Se utiliza para realizar consultas.

INSERT.- Permite añadir una o varias filas (tuplas) a una tabla (relación)

UPDATE.- Permite modificar uno o varios valores de una o mas filas de una tabla.

DELETE.- Permite borrar una o varias filas de una tabla.

2.- Sentencias de Definición de Datos (Data Definition Language).-

Permiten definir nuevos objetos (CREATE) o destruir (DROP) objetos existentes en una base de datos.

3.- Sentencias de Control de Datos (Data Control Language).-

Permiten garantizar la confidencialidad de acceso a los datos, mediante la concesión (GRANT) de autorizaciones o denegación (REVOKE) de éstas.

6.4.- Tipos de datos

Al asignar, mediante una sentencia CREATE, el nombre a las columnas o atributos de una tabla, se le asigna también a cada una de ellas un determinado tipo de datos de los predefinidos en SQL, con ello se está definiendo:

- El conjunto de los valores posibles que puede tomar el atributo (Dominio)
- Las operaciones que pueden realizarse con los valores almacenados en el atributo.

Los tipos de datos permitidos en SQL pueden ser:

a) Numéricos o cantidades susceptibles de participar en cálculos aritméticos

b) Alfanuméricos que representan combinaciones de caracteres cualesquiera

Datos Numéricos.-

Pueden ser:

*** Enteros**.- Sus valores son números enteros positivos o negativos.

SMALLINT o comprendidos entre -32768 y 322767 ambos inclusive

INTEGER o comprendidos entre -2147483648 y 2147483647 ambos inclusive.

*** Decimales**.- Números positivos o negativos que pueden tener parte real y parte fraccionaria. Están comprendidos entre -999999999999999 y 999999999999999.

Se definen mediante la palabra DECIMAL seguida de dos números enteros sin signo, encerrados entre paréntesis y separados por una coma. El primer número es la *precisión* y representa el número total de dígitos (parte entera y parte fraccionaria) que puede tener un valor. El segundo es la *escala* y representa el número de dígitos que puede tener la parte fraccionaria Así DECIMAL(3,0) es el conjunto de los números decimales sin parte fraccionaria comprendidos entre -999 y 999.

*** En coma flotante**.- Números positivos o negativos que pueden tener parte real y parte fraccionaria. Están comprendidos aproximadamente entre:

$5.4x10^{-79}$ y $7.2x10^{75}$ para los valores positivos

$-7.2x10^{75}$ y $-5.4x10^{-79}$ para valores negativos.

Se definen mediante las palabras reservadas FLOAT o REAL siendo ambas equivalentes.

Datos Alfanuméricos.-

Sus valores son hileras de caracteres cualesquiera tomados del conjunto de caracteres disponibles. En general permite manejar textos de hasta 254 caracteres. Pueden ser:

*** De longitud fija.-** Se definen mediante la palabra CHAR seguida de un número entero sin signo (longitud) entre paréntesis. CHAR(7) indica una hilera formada por 7 caracteres.

*** De longitud variable.-** Se definen mediante la palabra VARCHAR seguida de un número entero sin signo (longitud) entre paréntesis. VARCHAR(7) indica una hilera formada por hasta 7 caracteres

6.5.- Falta de datos (valores NULL)

Sea cual sea el tipo de dato especificado para una columna puede permitirse o no que entre sus posibles valores figure un valor especial, el valor nulo (NULL) que indica la carencia de datos o falta de información sobre dichos datos. Es decir, el interpretación que se hace del valor nulo es la de "valor Desconocido".

Al definir una columna en una tabla mediante la sentencia CREATE, puede permitirse la inclusión de valores nulos, indicando en su dominio correspondiente la palabra NULL o excluir dicho valor del dominio mediante la palabra NOT NULL.

Debe tenerse en cuenta que en una columna de tipo numérico el valor NULL no significa 0 ni es lo mismo que 0. Su significado real es número desconocido.

En una columna de tipo alfanumérico NULL no significa campo en blanco ni cadena de longitud cero. Su significado real es literal desconocido.

6.6.- Constantes

Las constantes o literales son secuencias de caracteres que representan un valor determinado, numérico o alfanumérico. Se especifican de la forma siguiente:

Numéricas:

Enteras y Decimales: mediante el valor de la constante con su signo caso de ser negativo: -10 , 20 , 3.14 , -5.72 etc.

Exponenciales: mediante una secuencia de caracteres formada por una constante entera o decimal de hasta 17 dígitos seguida de la letra E y de una constante entera de hasta 2 dígitos: 1.23E45 , 314E-2 , -11.2E17 , etc.

Alfanuméricas

Se especifican como secuencias de caracteres cualesquiera dentro del juego de caracteres disponibles, que empiezan y terminan con un apóstrofe (') : 'Informática' , '10-11-2000' , 'Harrold''S' , etc.

Como se ve en el último ejemplo se utilizan dos apóstrofes consecutivos para indicar al SGBD que se utiliza el carácter ' como literal y no como fin de la constante.

El SGBD trata las constantes alfanuméricas como datos de tipo VARCHAR.

6.7.- SQL para recuperación de datos

6.7.1.- 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.

6.7.2.- Consultas De Relación Única

6.7.2.1 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.

6.7.2.2 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

6.7.2.3 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

6.7.2.4 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.

6.7.2.5 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.

6.7.2.6 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.

6.7.2.7 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

6.7.2.8 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 UNIONSELECT 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 UNIONSELECT 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.

6.7.3 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.

6.7.3.1 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

6.7.3.1.1 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

6.7.3.1.2 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

6.7.4 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.

6.7.4.1 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)

6.7.4.2 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.

4.3Consultas 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.

6.7.4.4 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.

6.7.5 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.

6.7.5.1 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.

6.8.- SQL para modificaciones en la base de datos

SQL como lenguaje de programación en Bases de Datos, tiene dos formas de actuación: Como Lenguaje de Manipulación de Datos (DML) mediante el cual se consultan datos a la Base de Datos y se actualizan éstos, y como Lenguaje de Definición de los Datos (DDL) que permite definir y diseñar la estructura de los datos dentro de la Base de datos.

6.8.1 Actualizaciones en las Tablas

Las actualizaciones de los datos de la Base se realizan por alguno de los tres caminos siguientes: Inserción (INSERT), Borrado (DELETE) o modificación (UPDATE).

6.8.1.1 Introducción de datos en una tabla

Puede ser, a su vez, de tres formas distintas:

- Inserción de una fila

La expresión general de la sentencia de inserción de datos en una tabla es la siguiente:

INSERT INTO Tabla (Campo1, Campo2, Campo3, ... , CampoN) VALUES (Valor1, Valor2, Valor3, ... , ValorN)

Conceptualmente, la sentencia INSERT construye una fila de datos que se corresponden con la estructura en columnas de la tabla. La nueva fila insertada no tiene por que ser la primera ni la última de la tabla. Es simplemente una fila mas de la tabla que el Sistema Gestor de la Base de Datos situará donde proceda.

Cuando SQL inserta una nueva fila en una tabla, automáticamente asigna el valor NULL a cualquier campo que no haya sido especificado en la lista de columnas de la sentencia INSERT. Así mismo, puede hacerse más explícita esta asignación de NULL a un campo incluyendo la columna correspondiente en la lista y especificando la palabra clave NULL en el valor correspondiente:

INSERT INTO Tabla (Campo1, Campo2, Campo3, ... , CampoN) VALUES (Valor1, Valor2, NULL, ... , ValorN)

La sentencia anterior asigna el valor NULL al Campo3 en la nueva fila insertada

Debe tenerse en consideración que, si en la definición de un campo específico en una tabla se ha especificado que éste debe tener un valor no nulo, debe actualizarse explícitamente en la sentencia INSERT, ya que, de no hacerlo, el sistema produciría un error.

Así mismo, pueden utilizarse en la inserción constantes del sistema si el SQL utilizado lo permite:

INSERT INTO Tabla (Campo1, Campo2, Fecha3, ... , CampoN) VALUES (Valor1, Valor2, CURRENT DATE, ... , ValorN)

La sentencia anterior introduce el valor de la fecha actual en el Campo Fecha3.

Cuando se omite la lista de columnas a actualizar SQL genera automáticamente una lista formada por todas las columnas de la tabla en secuencia de izquierda a derecha. Así:

INSERT INTO Tabla VALUES (Valor1, Valor2, Valor3, ... , ValorN)

Actualiza todos los campos de la fila. Debe tenerse en cuenta que la palabra clave NULL debe ser utilizada en la lista de valores para asignar explícitamente valores NULL al campo que lo requiera y, además, la secuencia de valores debe corresponderse exactamente con la secuencia de columnas de la tabla.

- Inserción multifila

En este caso, los valores de datos para las nuevas filas a actualizar no son especificados explícitamente en el texto de la sentencia, en su lugar, la fuente de los datos de las nuevas filas es una consulta a la base de datos especificada en la sentencia:

INSERT INTO Tabla1 (Campo1, Campo2, Campo3, ... , CampoN) SELECT Campo1, Campo2, Campo3, ... , CampoN FROM Tabla2 WHERE Condición de Búsqueda.

Existen una serie de *restricciones lógicas* sobre la consulta que aparece dentro de una sentencia INSERT multifila, así, para SQL1:

- La consulta no puede contener la cláusula ORDER BY
- El resultado de la consulta debe contener el mismo número de columnas que hay en la lista de columnas de la sentencia INSERT (o que la tabla destino completa, si se ha omitido la lista de columnas) y los tipos de los datos deben ser compatibles columna a columna.
- La consulta no puede ser la UNION de varias sentencias SELECT diferentes (Esto es, únicamente puede especificarse una única sentencia SELECT)
- La tabla destino de la sentencia INSERT no puede aparecer en la cláusula FROM de la consulta, o de ninguna subconsulta que ésta contenga. Esto prohíbe insertar parte de una tabla sobre sí misma.

SQL2 es, sin embargo más flexible en las dos últimas restricciones ya que permite en la consulta expresiones y operaciones de unión y composición además de permitir la autoinserción.

- Inserción por carga masiva

En general, todos los SGBD comerciales incluyen en sus prestaciones la capacidad de carga masiva de datos en una tabla procedentes de un archivo. El estándar SQL de ANSI/ISO no considera esta función y suele ser suministrada como un programa de utilidad autónomo (Importar datos) en lugar de formar parte del lenguaje SQL.

6.8.1.2 Supresión de datos en una tabla

Una fila de datos se suprime de una tabla de una Base de Datos cuando la entidad representada por la fila desaparece del mundo exterior, esto es, la fila se suprime para mantener la Base de Datos como un modelo preciso del mundo real. La unidad más pequeña de datos que puede ser suprimida de una Base de Datos relacional es una única fila.

La sentencia DELETE elimina las filas seleccionadas de una tabla en función del criterio de selección indicado:

DELETE FROM Tabla WHERE Criterio de Selección

La cláusula WHERE actúa en la sentencia DELETE de forma exactamente igual a como lo hace en la sentencia SELECT, por lo que las condiciones de búsqueda en ambos casos tienen el mismo formato. Como consecuencia de ello es preciso tener en consideración lo siguiente:

- La cláusula WHERE puede especificar una sola fila o un conjunto de filas. En ambos casos, al actuar la sentencia DELETE, la fila o filas seleccionadas serán eliminadas de la tabla.
- Si no se especifica la cláusula WHERE, esto es:

DELETE FROM Tabla

La sentencia DELETE borra todas las filas las filas de la tabla, elimina todos los datos pero mantiene la estructura de la tabla, es decir, desaparece el contenido de la tabla pero se mantiene ésta en la Base de Datos.

6.8.1.3 Modificación de datos en una tabla

Tiene por fin actualizar los valores de la Base de Datos para mantener a ésta como un modelo preciso del mundo real. La unidad mínima que puede modificarse en una Base de Datos es una única columna de una única fila.

El formato general de la sentencia de actualización es:

UPDATE Tabla SET Campo1 = Valor1, Campo2 = Valor2, ... , CampoN = ValorN

WHERE Condición de Selección

La cláusula SET asigna los valores Valor1, Valor2, ... , ValorN respectivamente a las columnas Campo1, Campo2, ... , CampoN y actualiza con dichos valores a la fila o filas de la Tabla que cumplan la Condición de Selección indicada en la sentencia.

La cláusula WHERE actúa en la sentencia UPDATE de forma exactamente igual a como lo hace en la sentencia SELECT, por lo que las condiciones de búsqueda en ambos casos tienen el mismo formato. Esto implica, en particular, que si no se especifica la cláusula WHERE, se actualizarán todas las filas de la tabla. Ello permite una actualización masiva de la tabla destino.

6.8.2 Actualizaciones en la Base de Datos

Como se ha indicado en varias ocasiones, las sentencias SELECT, INSERT, DELETE, UPDATE y sentencias de transacciones se refieren a la manipulación de los datos en una Base de Datos. Estas sentencias de denominan colectivamente *Lenguaje de Manipulación de Datos (Data Manipulation Language)* o DML. Estas sentencias no pueden alterar la estructura de la Base de Datos, esto es, no pueden crear o destruir tablas o columnas en una Base de Datos.

Los cambios de estructura se realizan con el conjunto de sentencias denominadas conjuntamente *Lenguaje de Definición de Datos (Data Definition Language)* o DDL que permiten:

- Crear o suprimir una Base de Datos
- Definir y crear una nueva tabla
- Suprimir una tabla que ya no se necesita
- Cambiar la definición de una tabla existente
- Definir una tabla virtual (o Vista) de datos
- Establecer controles de seguridad para una Base de Datos
- Construir índices para hacer más rápido el acceso a la tabla
- Controlar el almacenamiento físico de los datos por parte del SGBD.

El núcleo del Lenguaje de Definición de Datos está basado en las sentencias siguientes:

- CREATE, que define y crea un objeto en la Base de Datos.

• DROP, que elimina un objeto existente en la Base de Datos

• ALTER, que modifica la definición de un objeto en la Base de Datos

En general, los principales productos comerciales de SGBD basados en SQL permiten utilizar el DDL mientras el SGBD está ejecutándose. La estructura de la Base de Datos es por tanto dinámica, permitiendo crear, eliminar o modificar la definición de las tablas de la base de datos mientras simultáneamente proporciona acceso a la base de datos a sus usuarios. Además, aunque el DDL y el DML son dos partes distintas del lenguaje SQL en la mayoría de los SGBD basados es SQL, la división es solamente conceptual. Las sentencias de DDL y de DML se remiten al SGBD de forma exactamente igual y pueden ser libremente entremezcladas en distintas sesiones de SQL.

Es preciso tener en consideración que el estándar de SQL de ANSI/ISO no exige la exige el soporte del DDL. Esto implica que las sentencias que se describen a continuación no están estandarizadas.

6.8.2.1 Creación de una Base de Datos

Al no existir un estándar para la creación de una Base de Datos, cada SGBD comercial adopta un planteamiento en la creación de una base ligeramente diferente:

• Oracle crea una base de datos como parte del proceso de la instalación del software Oracle. Normalmente las tablas de usuario se colocan en esta base de datos única global.

• Ingres incluye la utilidad especial CREATEDB que crea una nueva base de datos Ingres. El programa adicional DESTROYDB suprime una base de datos no necesaria.

• SQL Server y OS/2 Extended Edition incluyen la sentencia:

```
CREATE DATABASE Base
```

Como parte de su DDL. La sentencia adicional:

```
DROP DATABASE Base
```

destruye la base de datos creada previamente.

• SQLBase utiliza la orden de MS-DOS *COPY* para crear una nueva Base de Datos. El usuario simplemente hace una copia de una plantilla de base de datos vacía suministrada con el software SQLBase. Para suprimir una base de datos existente se utiliza la orden de MS-DOS *DEL*.

6.8.2.2 Definiciones de Tablas

La sentencia CREATE TABLE define una nueva tabla en la Base de datos y la prepara para aceptar datos.. Las diferentes cláusulas de la sentencia especifican los elementos de la definición de la tabla.

Cuando se ejecuta una sentencia CREATE TABLE, el usuario que la realiza (si tiene autorización para ello) se convierte en *propietario* de la tabla recién creada, a la cual se le da el nombre especificado en la sentencia , que debe ser un nombre SQL legal y no entrar en conflicto con el nombre de alguna tabla ya existente. La tabla recién creada está vacía pero el SGBD la prepara para aceptar datos añadidos con la sentencia INSERT.

Las columnas de la tabla se definen en el cuerpo de la sentencia CREATE TABLE y aparecen en una lista separada por comas e incluida entre paréntesis. El orden de la lista de

definiciones de columnas determina el orden de izquierda a derecha de las columnas de la tabla. Cada definición de columna especifica:

* *El nombre de la columna*. Cada tabla debe tener un nombre único, pero los nombres de las columnas pueden ser iguales a los nombres de columnas de otras tablas.

* *El tipo de datos de la columna*. Establece la clase de datos que la columna puede almacenar. Algunos tipos de datos, como VARCHAR o DECIMAL requieren información adicional, como la longitud o el número de decimales de los datos. Esta información adicional se incluye entre paréntesis a continuación de la palabra clave que especifica el tipo de datos.

* *El requerimiento de los datos*. La cláusula NOT NULL impide que aparezcan valores NULL en la columna. En caso contrario se permiten valores nulos.

* *El valor por omisión*. Opcional para la columna, indica el valor que el SGBD debe asignar a la columna cuando en una sentencia INSERT aplicada a la tabla no se especifica un valor para la columna.

Además el SQL2 suministra varias partes diferentes para la definición de una columna, como que contenga valores únicos, que sea una clave primaria o una clave ajena o restringir los valores que puede contener.

La forma general de la sentencia CREATE TABLE es la siguiente:

```
CREATE TABLE Tabla
(Campo1 INTEGER          NOT NULL,
Campo2  VARCHAR(XX) NOT NULL,
Campo3  DATE                    ,
Campo4  MONEY            NOT NULL)
```

La sentencia CREATE TABLE ofrece ligeras variantes según el SGBD utilizado ya que cada SGBD utiliza sus propias palabras clave para identificar los distintos tipos de datos en las definiciones de columna. Además, Sybase y SQL Server difieren mucho de otros productos SGBD y del estándar ANSI/ISO en el manejo de los valores NULL, ya que el estándar indica que una columna puede contener valores nulos a menos que específicamente se declare NOT NULL. Sybase y SQL Server utilizan el criterio opuesto, suponiendo que los valores NULL no son permitidos a menos que la columna se declare explícitamente como NULL.

En cuanto a los "valores por omisión", tanto el estándar ANSI/ISO como productos SQL soportados por IBM soportan este tipo de valores para las columnas, pero lo hacen de forma diferente: El estándar ANSI/ISO permite especificar un valor por omisión para cada columna. Así:

Estándar ANSI/ISO:

```
CREATE TABLE Tabla
(Campo1 INTEGER          NOT NULL     DEFAULT 106  ,
Campo2  VARCHAR(XX) NOT NULL     DEFAULT 'Nombre',
Campo3  DATE                                          ,
Campo4  MONEY            NOT NULL)
```

Los productos SQL de IBM no permiten especificar un valor diferente para cada columna sino que proporcionan un valor específico de omisión para cada tipo de dato utilizado, así el valor de omisión para datos numéricos es 0, para datos de cadena (VARCHAR) es la cadena vacía, para datos de caracteres (CHAR) es el blanco y para datos de fecha y hora es la fecha y hora actual:

Sintaxis IBM:

```
CREATE TABLE Tabla
(Campo1 INTEGER              NOT NULL     WITH DEFAULT      ,
Campo2  VARCHAR(XX) NOT NULL     WITH DEFAULT   ,
Campo3  DATE                                                ,
Campo4  MONEY                NOT NULL)
```

6.8.2.2.1 Definiciones de clave primaria y ajena y Restricciones de Unicidad

Además de la definición de las columnas de una tabla, la sentencia CREATE TABLE identifica la clave primaria de la tabla y las relaciones de la tabla con otras tablas de la Base de datos mediante las cláusulas PRIMARY KEY y FOREIGN KEY respectivamente.

La cláusula PRIMARY KEY especifica la columna o columnas que forman la clave primaria de la tabla. Esta columna (o combinación de columnas) sirve como identificador único para cada fila de la tabla. El SGBD requiere automáticamente que el valor de clave primaria sea único para cada fila de la tabla. Además, la definición de columna para todas y cada una de las columnas que forman la clave primaria debe especificar que la columna es NOT NULL.

La cláusula FOREIGN KEY especifica la clave ajena de la tabla y la relación que crea con otra tabla (Tabla Padre) de la Base de Datos. La cláusula especifica:

* La columna o columnas que forman la clave ajena, todas las cuales son columnas de la tabla que está siendo creada. Esto es, la columna de la tabla padre y la columna de la tabla hijo que se relaciona con la anterior deben tener el mismo nombre.

* La tabla que es referenciada por la clave ajena. Esta es la tabla padre en la relación, la tabla que se está definiendo es la tabla hija.

* Un nombre opcional para la Relación entre las dos tablas. Este nombre es opcional, no se utiliza en ninguna sentencia SQL pero es necesario si se desea poder suprimir la clave ajena posteriormente y puede aparecer en los mensajes de error.

* Como debe tratar el SGBD un valor NULL en una o mas columnas de la clave ajena, cuando compare filas con la tabla padre.

* Una regla de supresión opcional en la Relación (CASCADE, SET NULL, RESTRICT, SET DEFAULT o NO ACTION) que determina la acción que se debe realizar cuando se suprime una fila en la tabla padre

* Una regla de actualización opcional en la Relación (CASCADE, SET NULL, RESTRICT, SET DEFAULT o NO ACTION) que determina la acción que se debe realizar cuando se actualiza una parte de la clave primaria de la tabla padre.

Deben tenerse en cuenta las siguientes precisiones:

- La sintaxis definida no es estándar en ANSI/ISO

- No puede definirse una clave ajena que relacione la tabla que se est{a creando con una que no se ha creado todavía, es decir, debe crearse previamente la tabla padre antes de crear la tabla hijo. De no ser así, debe crearse la tabla hijo sin definición de clave ajena y posteriormente añadir dicha clave ajena utilizando la sentencia ALTER TABLE.

El estándar ANSI/ISO permite establecer en la sentencia CREATE TABLE restricciones de unicidad mediante la cláusula UNIQUE que exige que determinada columna o columnas tomen valores únicos. Sin embargo, prácticamente todos los SGBD mas populares hacen que las restricciones de unicidad formen parte de la sentencia CREATE INDEX que se describirá posteriormente.

La sintaxis general de creación de una tabla con claves primaria y ajena es:

```
CREATE TABLE Tabla1
(Campo1 INTEGER              NOT NULL,
Campo2  INTEGER              NOT NULL,
Campo3  VARCHAR(X)  NOT NULL,
Campo4  VARCHAR(X)  NOT NULL,
PRIMARY KEY (Campo1),
FOREIGN KEY SeRelacionaCon (Campo3)
REFERENCES Tabla2 ON DELETE SET NULL ON UPDATE CASCADE,
UNIQUE (Campo4))
```

Por último, si una clave primaria, una clave ajena o una restricción de unicidad afecta a una sola columna respectivamente, el estandar ANSI/ISO permite una forma abreviada de la sentencia cuya sintaxis es la siguiente:

```
CREATE TABLE Tabla1
(Campo1 INTEGER      NOT NULL     PRIMARY KEY,
Campo2  VARCHAR(X)  NOT NULL     UNIQUE,
Campo3  INTEGER      REFERENCES Tabla2,
Campo4  MONEY)
```

6.8.2.3 Eliminación de una Tabla

Para eliminar una tabla existente en una Base de Datos se utiliza la sentencia DROP TABLE, seguida del nombre de la tabla a eliminar:

```
DROP TABLE Tabla
```

Cuando esta sentencia suprime una tabla de la Base de Datos, su definición y todos sus contenidos se pierden y no es posible recuperarlos. Habría que crear de nuevo la tabla y posteriormente introducir sus datos. Debido a estas consecuencias, la sentencia DROP TABLE debe utilizarse con extremo cuidado.

El estándar SQL2 establece que una sentencia DROP TABLE incluye una sentencia CASCADE o una sentencia RESTRICT que especifica el impacto que tiene la eliminación de una determinada tabla sobre otros elementos de la base de datos que dependen de ella. No obstante la mayoría de los SGBD aceptan la sentencia DROP TABLE sin ninguna opción.

6.8.2.4 Modificación de una Definición de Tabla

La modificación de la definición de una tabla conllevaría alguna de las siguientes posibilidades:

- Añadir una definición de columna a una tabla
- Suprimir una columna de una tabla
- Cambiar el valor por omisión de una columna
- Añadir o eliminar una clave primaria para una tabla
- Añadir o eliminar una clave ajena para una tabla
- Añadir o eliminar una restricción de unicidad para una tabla
- Añadir o eliminar una restricción de comprobación para una tabla

6.8.2.4.1 Adición de una columna

El uso mas común de la sentencia ALTER TABLE es el de añadir una columna a una tabla existente. La nueva columna se añade al final de las definiciones de columnas de la tabla y se sitúa mas a la derecha de la tabla. La sentencia ALTER TABLE es prácticamente igual a la sentencia CREATE TABLE y su sintaxis es:

```
ALTER TABLE Tabla
ADD CampoN     VARCHAR(XX) NOT NULL
```

6.8.2.4.2 Supresión de una columna

La sentencia ALTER TABLE no puede ser utilizada para eliminar una columna de una tabla. Si realmente se quiere eliminar una columna de una tabla, los pasos a seguir son:

1° Descargar los datos de la tabla (por ejemplo mediante una Vista)

2° Utilizar la sentencia DROP TABLE para eliminar la definición de la tabla

3° Utilizar la sentencia CREATE TABLE para redefinir la tabla sin la columna no deseada

4° Volver a cargar los datos anteriormente descargados.

6.8.2.4.3 Modificación de Claves primaria y ajena

La sentencia ALTER TABLE también se utiliza para cambiar o añadir definiciones de claves primaria y ajena a una tabla. A diferencia de las definiciones de columna, las definiciones de clave primaria y ajena pueden ser añadidas y suprimidas con la sentencia ALTER TABLE. Las cláusulas que añaden definiciones de claves son exactamente las mismas que las contempladas en la sentencia CREATE TABLE y las cláusulas que suprimen las claves van antecedidas por la palabra reservada DROP:

Creación de una clave primaria:

```
ALTER TABLE Tabla
PRIMARY KEY (Campo1)
```

Creación de una clave ajena:

```
ALTER TABLE Tabla1
FOREIGN KEY NombreRelación (Campo2)
REFERENCES Tabla2
```

Eliminación de una clave primaria:

ALTER TABLE Tabla

DROP PRIMARY KEY

Eliminación de una clave ajena:

ALTER TABLE Tabla1

DROP FOREIGN KEY NombreRelación

La destrucción y creación de una clave puede hacerse en la misma sentencia SQL:

ALTER TABLE Tabla

DROP PRIMARY KEY

PRIMARY KEY (Campo2)

6.8.2.5 Definiciones de Restricción

En el estándar SQL2 la definición de la estructura de la Base de Datos se ha ampliado para incluir un nuevo área: *"Restricciones sobre los datos que se pueden introducir en la Base de Datos"*. Estas restricciones, denominadas *constraints*, son fundamentalmente las siguientes:

Restricción de unicidad.

Restricción de clave primaria

Restricción de clave ajena

Restricciones de comprobación

Aserciones

Definiciones de Dominio

Las tres primeras ya han sido tratadas anteriormente y se aplican en la sentencia CREATE TABLE y pueden ser modificadas por la sentencia ALTER TABLE.

6.8.2.5.1 Restricciones de comprobación

La restricción de comprobación (CHECK CONSTRAINT) limita el contenido de una tabla particular. En el estándar SQL2 una restricción de comprobación se especifica como una condición de búsqueda y aparece como parte de la definición de la tabla:

```
CREATE TABLE Tabla
(Campo1 INTEGER       NOT NULL,
Campo2  VARCHAR(X)    NOT NULL,
...  ...  ...
Fecha1   DATETIME     ,
CHECK (( Fecha1 < "01-Ene-2001) OR (Campo1 <= 3000)))
```

La restricción de comprobación se verifica cada vez que una sentencia SQL intenta actualizar o insertar datos en la tabla.

6.8.2.5.2 Aserciones

Una aserción (ASSERTION) es una restricción de la base de datos que restringe los contenidos de la base de datos en su conjunto. Se especifica como una condición de búsqueda, pero, a diferencia de la restricción de comprobación, la condición de búsqueda de una aserción puede restringir el contenido de múltiples tablas y los de los datos de las relaciones entre ellas. Se especifica, por tanto, como parte de la definición de la base de datos completa utilizando la sentencia CREATE ASSERTION. Un ejemplo sería:

```
CREATE ASSERTION ValorMaximo
CHECK ((Tabla1.Campo1 = Tabla2.Campo3) AND
(SUM(Tabla1.Campo4) <= ValorMaximo))
```

6.8.2.5.3 Definiciones de Dominio

Existe una restricción a priori en las columnas de una tabla, establecida por el tipo de los datos (y la extensión si procede) de la columna. Sin embargo, cuando se quiere especificar un dominio mas concreto para una determinada columna de una tabla, SQL2 implementa el concepto formal de dominio como una parte de la definición de la base de datos. Según SQL, un dominio es un conjunto nominado de valores de datos que realmente funciona como un tipo de datos adicional para utilizarlo en la definición de la base de datos. Se crea con la sentencia CREATE DOMAIN y puede ser referenciado dentro de la definición de la Tabla como si fuese un tipo de dato.

6.8.2.6 Indices

Un índice es una estructura que proporciona un acceso rápido a las filas de una tabla en función de los valores de una o mas columnas.

El SGBD utiliza el índice de la misma manera que se usa el índice de un libro: El índice almacena los valores y punteros a las filas en las que los valores se producen. En el índice los valores de datos están ordenados de forma ascendente o descendente para que el SGBD pueda buscar rápidamente el índice para encontrar un valor particular; posteriormente puede seguir el puntero para localizar la fila que tiene el valor.

La presencia o ausencia de índice es completamente transparente al usuario de SQL que acceda a una tabla. El funcionamiento de los índices es el siguiente:

Si el usuario realiza una consulta a la base de datos como:

```
SELECT Campo1, Campo2
FROM Tabla
WHERE Campo3 = 'Condición de Búsqueda'
```

Si no existe un índice asociado a la columna Campo3 el SGBD realizará la consulta exista o no exista información en ella. Procesará la consulta recorriendo secuencialmente la tabla fila a fila, para todas y cada una de las filas, verificando si se cumple 'Condición de Búsqueda' en Campo3. Si la tabla es muy grande la exploración puede llevar minutos, incluso horas.

Si existiera un índice asociado a la columna Campo3, El SGBD examina el índice para encontrar el valor solicitado ('Condición de Búsqueda') y luego sigue el puntero para encontrar la fila o filas solicitadas de la tabla. La búsqueda en índices es mucho mas rápida ya que el índice está ordenado y sus filas son muy pequeñas. Pasar del índice a la fila correspondiente es también muy rápido ya que el índice informa al SGBD el lugar del disco en el que está localizada la fila.

La gran ventaja de la utilización de índices se centra en que se acelera enormemente la ejecución de sentencias SQL con condiciones de búsqueda que se refieren a columnas indexadas.

Los inconvenientes se refieren a que consumen un espacio de disco adicional y a que deben ser actualizados cada vez que se añada a la tabla una fila nueva o se actualice el contenido de una fila existente.

Para establecer un índice a una columna de una tabla deben tenerse en cuenta las consideraciones siguientes:

- La indexación es apropiada a columnas que son utilizadas frecuentemente en condiciones de búsqueda.
- La indexación es aconsejable cuando las consultas de una tabla son mas frecuentes que las inserciones y las actualizaciones.
- El SGBD siempre establece un índice para la clave primaria ya que presupone que el acceso a la tabla se efectuará mas frecuentemente a través de dicha clave.

Para crear un índice se utiliza la sentencia CREATE INDEX mediante la cual se asigna un nombre al índice y se especifica la tabla para la cual se crea el índice, así como la columna o columnas a indexar y el orden, ascendente o descendente en que debe hacerse la indexación:

```
CREATE INDEX Campo3IDX
ON Tabla (Campo3)
```

La palabra clave UNIQUE que se utiliza para indicar que la columna a indexar debe tener valores únicos. el estándar ANSI/ISO la asocia a la sentencia CREATE TABLE, sin embargo el SQL basado en DB2 la asocia con el índice:

```
CREATE UNIQUE INDEX Campo3IDX
ON Tabla (Campo3)
```

Por último, la sentencia:

```
DROP INDEX Campo3IDX
```

Suprime un índice creado anteriormente.

6.9.- SQL en modo programación

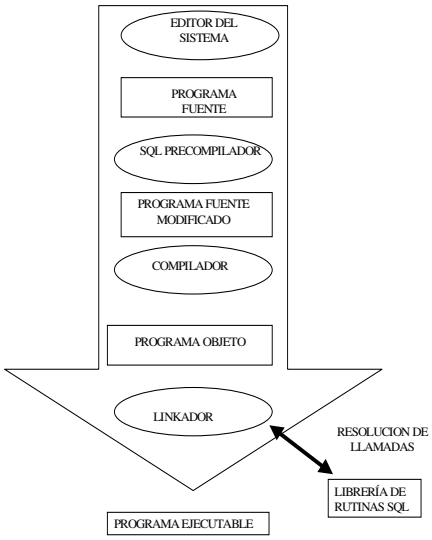
6.9.1 SQL Embebido

SQL también puede utilizarse para acceder a las tablas de una base de datos desde un programa de aplicación escrito en lenguajes como Pascal, Cobol, PL/I, Ensamblador, Basic, C , …

El lenguaje de programación que contiene sentencias SQL se denomina lenguaje *Anfitrión* y al SQL que forma parte del programa se llama lenguaje *Embebido*.

No existe limitación alguna respecto al tipo o número de sentencias SQL que se pueden embeber en un programa pudiéndose utilizar todas las sentencias y directivas vistas anteriormente tanto en Lenguaje de Definición de Datos (DDL) como en Lenguaje de Manipulación de Datos (DML).

Los programas con SQL embebido deben ser procesados por un *precompilador* antes de utilizar el compilador normal del lenguaje correspondiente. La misión del *precompilador* consiste en traducir las sentencias SQL en llamadas a rutinas de biblioteca que se encargan de realizar las acciones correspondientes sobre la Base de Datos. Por ello, dichas sentencias deben aparecer en el programa precedidas por un carácter o comando especial que permite al precompilador reconocerlas.



Igualmente, si dentro de la sentencia SQL embebida se necesitan utilizar variables definidas en el programa anfitrión, éstas se deben distinguir mediante un símbolo especial que las preceda. Estos símbolos o comandos especiales serán diferentes dependiendo del lenguaje de programación que se utilice. Por este motivo, se ha optado aquí por una codificación en Pseudocódigo, usando EXEC-SQL y END-EXEC para delimitar las sentencias SQL embebidas y el símbolo ":" para distinguir las variables anfitrión dentro de ellas.

6.9.2 Ejemplo de Aplicación

Como ejemplo de aplicación se desarrollara un programa que permita cómodamente a un operador realizar una reserva de plaza para un vuelo y vender el billete correspondiente.

La base de datos sobre la que actuará la aplicación esta formada por tres tablas:

Vuelos

Num_Vuelo	Origen	Destino	Hora_Salida	Tipo_Avion

Reservas

Num_Vuelo	Fecha_Salida	Plazas_Libres

Aviones

Tipo	Capacidad	Longitud	Envergadura	Veloc_Crucero

Las operaciones podrían incluir obtener el número de plazas libres de un vuelo y fecha determinados, averiguar todos los vuelos que hay entre un origen y un destino dados, reservar una plaza, etc. Cada una de ellas podría ser una *opción de menú* de manera que cuando el operador seleccione una opción determinada, se ejecute una subrutina del programa encargada de esta tarea.

Así, si el operador selecciona la opción "Obtener número de plazas" se iniciaría una subrutina que deberá hacer lo siguiente:

Pedir al operador las ciudades *origen* y *destino* del vuelo así como la fecha y hora de éste.

Hacer la consulta correspondiente para recuperar el número de plazas.

Presentar los resultados en pantalla.

El primer y tercer paso se realizarán mediante el lenguaje anfitrión elegido, en cuanto al segundo, la sentencia SELECT correspondiente sería:

```
SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES
FROM VUELOS, RESERVAS
WHERE ORIGEN = :origen AND DESTINO = :destino AND FECHA_SALIDA =
:fecha
AND HORA_SALIDA = :hora AND RESERVAS.NUM_VUELO =
VUELOS.NUM_VUELO
```

Las variables :origen, :destino, :fecha y :hora son los parámetros de entrada de la sentencia SELECT y contendrán los valores leídos en el apartado 1. Están precedidas por el signo ":" como se ha indicado anteriormente.

El resultado de la consulta son dos valores RESERVAS.NUM_VUELO y PLAZAS_LIBRES. A diferencia de lo que ocurre con SQL interactivo, el resultado de un SELECT de SQL embebido no se muestra automáticamente en pantalla sino que debe ser recogido por variables del programa en el que se encuentra inmerso. Estas variables se especifican en la cláusula INTO (exclusiva de SQL embebido) y corresponden a los parámetros de salida de la consulta, esto es, a los valores que devuelve.

La cláusula INTO va inmediatamente antes de la cláusula FROM dentro de una sentencia SELECT. En ella se especifican las variables a las que se deben asignar los valores devueltos.

Debe haber tantas variables como columnas se recuperen en la sentencia SELECT. En el caso anterior, la sentencia SELECT quedaría así:

```
SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES
INTO :vuelo, :plazas
FROM VUELOS, RESERVAS
WHERE ORIGEN = :origen AND DESTINO = :destino AND FECHA_SALIDA =
:fecha
AND HORA_SALIDA = :hora AND RESERVAS.NUM_VUELO =
VUELOS.NUM_VUELO
```

Esto significa que el valor devuelto para la columna RESERVAS.NUM_VUELO debe ser asignado a la variable :vuelo y el de la columna PLAZAS_LIBRES a la variable :plazas.

La subrutina completa, en Pseudocódigo tendría el aspecto siguiente:

```
VARIABLES
Origen, Destino, Fecha, Hora, Vuelo : STRING
Plazas : ENTERO

COMIENZO
  ESCRIBIR "Introduzca la ciudad de Origen:"
  LEER Origen
  ESCRIBIR "Introduzca la ciudad de Destino:"
  LEER Destino
  ESCRIBIR "Introduzca la Fecha:"
  LEER Fecha
  ESCRIBIR "Introduzca la Hora:"
  LEER Hora
EXEC-SQL
SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES
INTO :Vuelo, :Plazas
FROM VUELOS, RESERVAS
WHERE ORIGEN = :Origen AND DESTINO = :Destino AND FECHA_SALIDA =
:Fecha
AND HORA_SALIDA = :Hora AND RESERVAS.NUM_VUELO =
VUELOS.NUM_VUELO
END-EXEC
ESCRIBIR "Número de vuelo: ", Vuelo
ESCRIBIR "Número de Plazas Libres: ", Plazas
FIN
```

Debe tenerse en cuenta las siguientes consideraciones:

.- Las variables del lenguaje utilizado aparecen precedidas del símbolo ":" *exclusivamente* dentro de las sentencias SQL pero no en el resto del código.

.- Los tipos de las variables han de ser compatibles con los tipos de datos de las columnas cuyos valores van a recoger. Son particularmente importantes las funciones de conversión de tipos integradas en los lenguajes anfitrión utilizados, para establecer dicha compatibilidad.

.- La solución expresada anteriormente sólo es válida en el caso en que exista *una única fila que satisfaga las condiciones impuestas por la cláusula WHERE* ya que las variables anfitrión sólo pueden recibir un único valor, dado que son variables simples.

Si el resultado de la cláusula WHERE es de más de una fila, es preciso definir una nueva estructura de datos que almacene un número de filas a priori indeterminado. Esta nueva estructura es el *CURSOR* y funciona como un puntero que apunta a cada fila resultado.

6.9.3 Manejo de Cursores

El manejo del cursor se realiza en cuatro etapas:

1.- *Declarar el cursor.-* Como cualquier otro tipo de variable un cursor necesita ser descrito. Para ello se le asocia a la sentencia SQL que lo va a utilizar. La sintaxis es:

DECLARE <nombre-cursor> CURSOR FOR <sentencia-Select>

2.- *Abrir el cursor.-* El Sistema Gestor de Base de Datos se encarga de construir la tabla resultante de la consulta asociada al cursor. El cursor queda apuntando a una posición anterior a la primera fila. La sentencia requerida es:

OPEN <nombre-cursor>

3.- *Recuperar la fila.-* El SGBD avanza el cursor una fila y se la devuelve al programa dentro de las variables anfitrión que se especifiquen. Esta operación se ejecutará cuantas veces sea necesario hasta que se recuperen todas las filas. La sentencia a utilizar es:

FETCH <nombre-cursor> INTO <lista-de-variables>

Cuando se realiza un FETCH y no quedan mas filas que recuperar la sentencia actualiza una variable especial, llamada SQLCODE asignándole el valor de 100. Esto permite detectar el final del cursor.

4.- *Cerrar el cursor.-* Cuando se ha recuperado la última fila del cursor hay que cerrarlo:

CLOSE <nombre-cursor>

Como ejemplo de manejo de cursores se desarrollará una subrutina correspondiente a otra opción de menú de la aplicación anterior: Aquella que permite averiguar todos los vuelos que hay entre un origen y un destino dados. Las acciones necesarias para la subrutina son:

1.- Pedir al operador el origen y el destino.

2.- Realizar la consulta correspondiente.

3.- Recuperar la primera fila seleccionada.

4.- Finalizar si la fila pedida no existe,

5.- Escribir la fila recuperada en la pantalla.

6.- Recuperar la siguiente fila.

7.- Volver al paso 4.

8.- Finalizar

VARIABLES

Origen, Destino, Hora, Vuelo : STRING

```
EXEC-SQL
  DECLARE Lista_Vuelos CURSOR FOR
        SELECT Num_Vuelo, Hora_Salida
        FROM VUELOS
        WHERE Origen = :Origen AND Destino = :Destino
END-EXEC
COMIENZO
ESCRIBIR "Introduzca la Ciudad de Origen: "
LEER Origen
ESCRIBIR "Introduzca la Ciudad de Destino: "
LEER Destino
EXEC-SQL OPEN Lista_Vuelos END-EXEC
EXEC-SQL FETCH Lista_Vuelos INTO :Vuelo, :Hora END-EXEC
MIENTRAS SQLCODE <> 100 HACER
  ESCRIBIR Vuelo, Hora
  EXEC-SQL FETCH Lista_Vuelos INTO :Vuelo, :Hora END-EXEC
FIN MIENTRAS
EXEC-SQL CLOSE Lista_Vuelos END-EXEC
FIN
```

4 Utilización de sentencias SQL de actualización .-

Como ejemplo de uso de sentencias de actualización con SQL embebido se desarrolla una subrutina que realice reservas de plazas. Dicha subrutina deberá:

1.- Pedir al operador el número de vuelo, la fecha y el número de billetes a reservar.

2.- Obtener el número de plazas libres para dicho vuelo y fecha

3.- Realizar la reserva, si hay suficientes plazas, emitiendo en caso contrario un mensaje.

El Pseudocódigo de la subrutina podría ser:

```
VARIABLES
  Vuelo, Fecha : STRING
  Billetes, Plazas : ENTERO
COMIENZO
  ESCRIBIR "Introduzca el Número de Vuelo: "
  LEER Vuelo
  ESCRIBIR "Introduzca la fecha: "
  LEER Fecha
```

```
  ESCRIBIR "Introduzca el Número de Billetes Solicitados: "
  LEER Billetes
EXEC-SQL
  SELECT Plazas_Libres INTO :Plazas
        FROM RESERVAS
        WHERE Num_Vuelo = :Vuelo AND Fecha_Salida = :Fecha
END-EXEC
SI Plazas >= Billetes ENTONCES
  EXEC-SQL
        UPDATE RESERVAS
              SET Plazas_Libres = :Plazas - :Billetes
              WHERE Num_Vuelo = :Vuelo AND Fecha_Salida = :Fecha
  END-EXEC
  ESCRIBIR "Reserva Realizada"
SI NO
  ESCRIBIR "No hay Suficientes Plazas para Reservar"
FIN SI
FIN
```

Con los ejemplos anteriores se han mostrado las principales características y las aplicaciones mas habituales de SQL embebido. La inclusión de otras sentencias SQL dentro de un programa de aplicación se realiza de forma similar.

6.10. Vistas

6.10.1 Concepto de Vista

Una vista es una "tabla virtual" en una Base de Datos, cuyos contenidos están definidos por una consulta. Para el usuario la vista aparece como una tabla real formada por columnas designadas y filas de datos. Sin embargo, a diferencia de la tabla real, la vista no existe en la base de datos como un conjunto almacenado de valores, esto es, los datos y columnas visibles en la vista son los resultados de la consulta que define la vista. SQL suministra a la vista un nombre semejante al de una tabla y almacena la definición de la vista en la base de datos.

Las vistas permiten:

Acomodar el aspecto de la base de datos de manera que diferentes usuarios vean la base desde distintas perspectivas.

Restringir el acceso a los datos, de modo que diferentes usuarios únicamente vean ciertas filas o ciertas columnas de una tabla.

Simplificar el acceso a la base de datos permitiendo una presentación de la estructura de los datos almacenados de modo que sea más natural a cada usuario.

Cuando el SGBD encuentra una referencia a una vista en una sentencia SQL determina la definición de la vista almacenada en la base de datos, traduce la petición de consulta indicada en la vista a una petición equivalente con las tablas fuente de la vista y ejecuta la sentencia SQL. De este modo el SGBD mantiene la integridad de las tablas fuente.

6.10.2 Ventajas y desventajas de las Vistas

La utilización de vistas suministra importantes **beneficios**:

Seguridad.- Cada usuario puede obtener permiso para acceder a la base de datos mediante determinadas vistas que contienen los datos específicos que dicho usuario está autorizado a ver.

Simplicidad de Consulta.- La vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla, haciendo que consultas multitabla pasen a ser consultas sobre una sola vista.

Simplicidad estructurada.- Pueden dar al usuario una visión personalizada de la base de datos ofreciéndole a éste un conjunto de tablas virtuales que son precisamente las que tienen sentido para dicho usuario.

Integridad de los datos.- Si se accede a los datos y se introducen a través de una vista, el SGBD puede comprobar automáticamente los datos para asegurarse que satisfacen restricciones de integridad especificadas.

Aunque las vistas presentan ventajas sustanciales hay que tener en cuenta dos **desventajas** importantes de cara a utilizar una vista en vez de una tabla real:

Rendimiento.- Las vistas crean una apariencia de una tabla, pero el SGBD debe traducir las consultas definidas sobre una vista en consultas sobre las tablas fuente subyacentes a la vista. Ello hace que si la vista se define mediante una consulta multitabla compleja, aunque la consulta realizada sobre la vista sea sencilla, se convierte en una composición complicada para el SGBD y su realización puede tardar mucho tiempo en completarse.

Restricciones de actualización.- Al tratar de actualizar filas en una vista, el SGBD debe traducir la petición a una actualización en las tablas fuente origen de la vista. No hay problemas si la vista es sencilla, pero vistas complejas pueden no ser actualizadas y convertirse en de "sólo lectura".

6.10.3 Creación de una vista

La sentencia CREATE VIEW permite crear una vista. Asigna a la vista un nombre y especifica la consulta que define la vista. La creación de una vista implica el tener permiso de acceso a todas las tablas referenciadas en la consulta. Además puede asignar un nombre a cada columna de la vista recién creada.

Aunque todas las vistas se crean del mismo modo, en la práctica se utilizan tipos distintos de vista para propósitos diferentes:

6.10.3.1 Vistas Horizontales

Tiene por objeto restringir al usuario el acceso a determinadas filas de una tabla. Se crea, por tanto mediante una consulta sobre todas las columnas de la tabla y cuya condición de búsqueda limita las filas únicamente a las deseadas. Son ejemplos de vistas horizontales:

CREATE VIEW Vista AS

```
SELECT *
FROM Tabla
WHERE Campo1 IN (Valor1, Valor2, … , ValorN)
----
CREATE VIEW Vista AS
SELECT *
FROM Tabla
WHERE Campo1 = 'Texto'  OR Campo2= Valor
```

6.10.3.2 Vistas Verticales

Tiene por objeto restringir al usuario el acceso a determinadas columnas de una tabla. Su sintaxis general más elemental sería:

```
CREATE VIEW Vista AS
SELECT Campo1, Campo2, … , CampoN
FROM Tabla
```

Donde Campo1, Campo2, … , CampoN son las columnas que se quieren contemplar en la vista.

6.10.3.3 Vistas con subconjuntos fila/columna

Son aquel tipo de vistas en las que se restringe tanto el acceso a determinadas columnas de una tabla como a ciertas filas que no cumplen una determinada condición de búsqueda:

```
CREATE VIEW Vista AS
SELECT Campo1, Campo2, … , CampoN
FROM Tabla
WHERE Campo1 = Valor AND Campo2 = 'Texto'
```

Donde Campo1, Campo2, … , CampoN son las columnas que se quieren contemplar en la vista.

Y únicamente se verán aquellas filas en las que el valor de Campo1 sea Valor y el contenido de Campo2 sea Texto.

6.10.3.4 Vistas Agrupadas

Son aquellas en las que los datos visualizados proceden de una consulta agrupada. Agrupa filas relacionadas de datos y producen una fila de resultados de consulta para cada grupo resumiendo los datos de ese grupo.

```
CREATE VIEW Vista (Vcampo1, Vcampo2, Vcampo3, Vcampo4, Vcampo5, Vcampo6)
AS
SELECT Campo1, COUNT(*), SUM(Campo2), MIN(Campo3), MAX(Campo4),
AVG(Campo5)
FROM Tabla
GROUP BY Campo1
```

La vista creada agrupa, en función de Campo1 de la Tabla las operaciones de columna (COUNT(*), SUM(), MIN(), MAX(), AVG()) respectivamente, de los campos Campo2, Campo3, Campo4 y Campo5 y las asigna respectivamente a las columnas de la consulta Vcampo1, Vcampo2, Vcampo3, Vcampo4, Vcampo5 y Vcampo6.

6.10.3.5 Vistas compuestas

Una de las principales razones para la utilización de vistas es simplificar las consultas multitabla. Especificando una consulta de dos o tres tablas en la definición de la vista, se puede crear una vista compuesta que extrae sus datos de dos o tres tablas diferentes y presenta los resultados como una tabla virtual.

Una vez definida la vista, pueden realizarse consultas simples sobre dicha vista con peticiones que, en caso contrario requerirían una composición de dos o tres tablas.

CREATE VIEW Vista Vcampo1, Vcampo2, Vcampo3 AS

SELECT CampoA1, CampoB1, CampoC1

FROM TablaA, TablaB, TablaC

WHERE CampoA2 = CampoB2 AND CampoB3 = CampoC3

6.10.4 Actualización de una vista

Determinadas vistas permiten insertar datos, modificarlos o suprimir filas de la vista, estas operaciones son obviamente traducidas a operaciones equivalentes respecto de las tablas fuente de la vista. Esto es, una actualización en los datos de una vista comporta la actualización de los datos (eliminación, modificación o inserción) de la o las tablas subyacentes a la vista.

El estándar ANSI/ISO especifica las vistas que pueden ser actualizables. Según este estándar, una vista puede ser actualizada si la consulta que la define satisface todas las restricciones siguientes:

- No debe especificarse la cláusula DISTINCT esto es, las filas duplicadas no deben ser eliminadas de los resultados de la consulta .
- La cláusula FROM de la consulta subyacente debe especificar únicamente una tabla actualizable, esto es debe tener una única tabla fuente para la cual el usuario tiene las autorizaciones y privilegios requeridos. Si la tabla fuente es así mismo una vista, ésta debe satisfacer estos criterios.
- Cada elemento de selección debe ser una referencia de columna simple. La lista de selección no puede contener expresiones, columnas calculadas o funciones de columna.
- La cláusula WHERE no debe incluir una subconsulta, sólo pueden aparecer condiciones de búsqueda simples fila a fila.
- La consulta subyacente no debe incluir una cláusula GROUP BY o HAVING

El resumen de estas restricciones es el siguiente: Para que una vista sea actualizable, el SGBD debe ser capaz de relacionar cualquier fila de la vista con su respectiva fila fuente de la tabla fuente. Análogamente debe ser capaz de relacionar cada columna individual de la vista con la columna individual a actualizar de la tabla fuente.

En general, las actualizaciones de vistas en los productos comerciales basados en SQL son bastante menos restrictivas que el estándar fijado por el ANSI/ISO.

6.10.5 Eliminación de una vista

La eliminación de una vista según el estándar ANSI/ISO para SQL2 se realiza mediante la sentencia DROP VIEW seguida del nombre de la vista y del método de eliminación (CASCADE, RESTRICT, etc.). Según:

DROP VIEW NombreVista CASCADE

Aunque la mayoría de los productos comerciales soporta la versión de la sentencia DROP VIEW sin establecer explícitamente el método de eliminación.