Temas Generales para la preparación de la Oposición al Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado.

Cuerpo Superior de Estadísticos del Estado
Especialidad de Estadística-Ciencia de Datos.

Almacenamiento y modelos de datos

Tema 4 - SQL: manipulación de datos

AUTOR: ANDONI PÉREZ DE LEMA SÁENZ DE VIGUERA

Asociación Profesional de Cuerpos Superiores de Sistemas y Tecnologías de la Información de las Administraciones Públicas

Creación: Junio 2021

ÍNDICE

1 INTRODUCCION

SQL (abreviatura de “Structured Query Language”) es un lenguaje especializado de programación, de tipo declarativo y basado en el modelo relacional, que se emplea para administrar, recuperar y actualizar información de sistemas de gestión de bases de datos relacionales, a distintos niveles de utilización: usuarios, programadores y administradores de la base de datos. Fue desarrollado en los años 70 en los laboratorios de IBM, y se ha convertido en un estándar ISO (International Standard Organization) y ANSI (American National Standards Institute). Actualmente más de 100 sistemas de gestión de bases de datos soportan SQL, que se ha convertido en el lenguaje de consulta relacional más popular a nivel mundial. Tras esta visión general, revisamos a continuación el proceso histórico de evolución del lenguaje SQL.

Donald Chamberlain, del laboratorio IBM de San José, definió en 1974 el lenguaje SEQUEL (Structured English Query Language). En 1976 se definió una versión revisada que se denominó SEQUEL/2, pero el nombre se cambió posteriormente a SQL por razones legales.

En 1977, con el fin de validar la viabilidad del modelo relacional , IBM desarrolló un nuevo prototipo llamado System R, que supuso la primera implementación de SQL. Durante los años siguientes, diversos vendedores americanos fueron anunciando productos basados en SQL, tales como : ORACLE (Oracle Corp.), SQL/DS (IBM) –para entornos DOS/VSE y VM/CMS-, DG/SQL (Data General Corp.), DB2 (IBM) –para entornos MVS-, DG/SQL (Data General Cor.), INGRES (University of California, Berkley).

En 1982, la ANSI abordó el desarrollo de una propuesta de lenguaje relacional estándar, y la ISO se sumó a este proyecto en 1983. La propuesta, que consistía esencialmente en el dialecto SQL de IBM, fue ratificada como estándar en 1986 por la ANSI, y en 1987 por parte de la ISO.

Desde entonces, el lenguaje ha sido revisado para incluir un conjunto más amplio de prestaciones, habiéndose publicado las siguientes revisiones del estándar a lo largo del tiempo : SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL: 2016.

Las funcionalidades adicionales a las que incluye el estándar de SQL, y que son proporcionadas por los diferentes proveedores de bases de datos, se denominan “extensiones al estándar”. Cada implementación de SQL se denomina “dialecto”. Es preciso tener en cuenta que si se usan extensiones al estándar, el código no será portable a otros sistemas de gestión de bases de datos relacionales.

2 CARACTERISTICAS DEL LENGUAJE SQL

SQL se ha convertido en el lenguaje estándar para trabajar con bases de datos relacionales, y es soportado por la práctica totalidad de los productos disponibles en el mercado. Al igual que el resto de lenguajes relacionales, SQL es un lenguaje declarativo que está basado en el cálculo relacional de tuplas. En consecuencia, toda consulta formulada utilizando el cálculo relacional de tuplas (o equivalentemente, el álgebra relacional) se puede realizar también mediante SQL. Sin embargo, SQL dispone igualmente de agunas capacidades que van más allá del álgebra y el cálculo relacional, entre las que destacan :

- Comandos para la inserción, borrado o manipulación de datos.
- Capacidades aritméticas : en SQL se pueden incluir operaciones aritméticas y comparadores. Por ejemplo : X – Y > 5.
- Asignación y comandos de impresión : es posible asignar una tabla calculada a un nombre de tabla, así como representar/imprimir una tabla construida mediante una consulta.
- Operaciones para cadenas de caracteres, tales como : concatenación de cadenas, extracción de subcadenas, longitud de cadenas, conversión de cadenas a mayúsculas o minúsculas, eliminación de espacios al final de la cadena, etc.
- Funciones agregadas : se pueden aplicar operaciones tales como promedio, suma, máximo, etc., a las columnas de una tabla, con el fin de obtener una cantidad única o agregada.

SQL es un “lenguaje orientado a la transformación”, y de cuarta generación (4GL), es decir, un lenguaje que usa relaciones para transformar unas entradas en las salidas deseadas. En los lenguajes procedimentales de tercera generación se debían especificar todos los pasos que hay que dar para conseguir el resultado. Sin embargo, en SQL tan sólo deberemos indicar al Sistema de Gestión de Base de Datos (SGBD) qué es lo que queremos obtener - pero no cómo hacerlo-, y usando una sintaxis relativamente simple. La estructura de los comandos se basa en palabras del inglés estándar, como : CREATE TABLE, SELECT, INSERT, UPDATE o DELETE.

SQL es fundamentalmente un lenguaje de formato libre, lo que implica que no exige que las diferentes partes de las instrucciones se escriban en localizaciones determinadas de la pantalla. Es decir, SQL no requiere la indentación o sangrado para delimitar la estructura del código, pero se puede emplear opcionalmente para facilitar su legibilidad.

Por otro lado, las bases de datos relacionales permiten la ejecución de instrucciones SQL de dos formas : de manera directa o interactiva, y también como parte de un programa de aplicación (esto es, las instrucciones SQL pueden estar incrustadas o embebidas, lo que significa que pueden encontrarse entremezcladas con las instrucciones del lenguaje de programación de la aplicación, que se denomina “lenguaje anfitrión”). Pueden existir algunas diferencias de detalle entre las instrucciones SQL interactivas y su equivalente en SQL embebido, pero el principio general del “modo dual” de SQL (posibilidad de utilizarlo tanto en modo interactivo como embebido) sigue siendo válido.

El lenguaje SQL consta de los siguientes componentes :

- **Lenguaje de definición de datos (DDL)** : que permite definir la estructura de la base de datos y controlar el acceso a los datos. Proporciona comandos para la definición y modificación de esquemas de relación, así como para el borrado de relaciones. DDL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos. Asimismo, DDL incluye comandos para la definición de vistas, y comandos para especificar los derechos de acceso a las relaciones y las vistas.
- **Lenguaje interactivo de manipulación de datos (DML)** : el DML incluye un lenguaje de consultas (que permiten recuperar y manipular datos), basado tanto en el álgebra relacional como el cálculo relacional de tuplas. Asimismo, contiene comandos adicionales, como se ha indicado anteriormente. El DML puede operar tanto a nivel externo (sobre vistas) como en el nivel conceptual (sobre tablas base).
- **Lenguaje de control de comandos (DCL) :** está constituido por comandos que permiten al Administrador del sistema gestor de base de datos controlar el acceso a los objetos, es decir, otorgar o denegar permisos a uno o más roles para realizar determinadas tareas.

- **Lenguaje de control de transacciones (TCL)** : SQL incluye comandos para especificar el inicio y final de las transacciones, así como para hacer definitivos los cambios o deshacer las transacciones, volviendo al estado inicial.

3 LENGUAJE DE MANIPULACION DE DATOS (DML)

Una instrucción SQL consta de : **palabras reservadas** y **palabras definidas por el usuario**. En general, la mayor parte de componentes de una instrucción SQL no distinguen mayúsculas y minúsculas, exceptuando los datos de literales no numéricos, que deben indicarse tal y como aparezcan en la base de datos. Los "literales" son constantes que se usan en instrucciones SQL. Los literales correspondientes a valores no numéricos irán rodeados de comillas simples, mientras que los literales numéricos no deberán estar delimitados con comillas. Si bien no se requiere en el estándar, muchos dialectos o implementaciones de SQL exigen que las instrucciones SQL finalicen en ";", y adoptaremos esa convención en los ejemplos proporcionados en este tema.

El lenguaje DML tiene las siguientes sentencias :

- **SELECT** : permite realizar la consulta de datos almacenados en tablas de base de datos. Es la sentencia de DML más potente, con sintaxis más compleja, y también la que se ejecuta más frecuentemente.
- **INSERT** : agrega uno o más registros a una (y sólo una) tabla de una base de datos relacional.
- **UPDATE** : es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.
- **DELETE** : borra uno o más registros existentes en una tabla.

Todos los ejemplos presentados en este tema, que servirán para ilustrar el uso de las sentencias SQL de DML, actuarán sobre las siguientes tablas :

Proveedor (P_No, P_Nombre, CIF, Ciudad, Sector, Correo, Telefono, Cuenta, Facturacion, Direccion, Pais)

Articulo (A_No, A_Nombre, A_Precio, Peso, Categoria)

Vende (P_No, A_No)

Personal (Per_No, Per_Nombre, Per_Apellido1, Per_Apellido2, Per_Salario, Puesto, P_No, Dep_No)

Departamento (Dep_No, Dep_Nombre)

Cliente (C_No, C_Nombre, Ciudad, Pais)

Pedido_Cliente (PC_No, Fecha, PC_Precio, C_No)

3.1. Consultas simples

La instrucción "SELECT" se emplea en SQL para realizar consultas. Tiene la capacidad de combinar, en un solo comando, las operaciones del algebra relacional de Selección, Proyección y "Join", entre otras funcionalidades. Esta sentencia devuelve una tabla lógica al cliente, ya que los resultados de la consulta se presentan en forma de un conjunto de filas y columnas, a semejanza de las tablas de base de datos. Pero una tabla de resultados sólo estará en memoria mientras la utilicemos, y posteriormente se descartará, a diferencia de las tablas de la base de datos.

El comando SELECT tiene la siguiente sintaxis, expresada en notacion “Backus Naur Form” (BNF) :

SELECT [DISTINCT | ALL] {* | [columnExpression [**AS** newName]] [, . . .]}
FROM TableName [alias] [, . . .]
[WHERE condition]
[GROUP BY columnList] **[HAVING** condition]
[ORDER BY columnList]

El orden de las cláusulas en un comando SELECT no puede cambiarse, y es el que se indica en la expresión anterior, en la que se puede observar que las únicas clausulas obligatorias son “SELECT” y “FROM”.

Al objeto de clarificar el significado de la notación BNF, que usaremos para definir la sintaxis de las sentencias SQL, reseñamos sus aspectos principales :

- las letras mayúsculas representan **palabras reservadas**,
- las letras minúsculas representan **palabras definidas por el usuario**,
- una barra vertical representa una **elección** entre alternativas,
- las llaves ({}) representan un **elemento obligatorio**,
- los corchetes ([]) representan un **elemento opcional**,
- los tres puntos (…) representan la **repetición opcional de un elemento**, cero o más veces.

A continuación de la palabra “SELECT”, se identifican los nombres de las columnas (o datos calculados) de las tablas o vistas de las que se quiere extraer datos. Si se desea seleccionar todos los campos, se puede emplear el operador “*”, que evita tener que identificarlos uno a uno. La cláusula “WHERE” se utiliza para seleccionar únicamente aquellas filas que cumplan la condición especificada. La cláusula “ORDER BY” permite ordenar los registros resultantes de una consulta, en función de los valores de una o varias columnas, en orden ascendente o descendente. “GROUP BY” tiene como propósito organizar los resultados de una consulta por grupos de filas que tengan el mismo valor para una columna o conjunto de columnas. “HAVING” se usa para filtrar los grupos, estableciendo una condición de selección.

Vamos a revisar a continuación los principales tipos de consultas simples que se pueden realizar, proporcionando ejemplos para cada una.

A.- Recuperación de todas las filas y columnas de una tabla.

La siguiente consulta recuperaría el valor de todas las columnas del conjunto de registros de la tabla “Proveedor” :

```
SELECT *
FROM Proveedor;
```

B.- Recuperación de columnas específicas, para todas las filas de una tabla.

La siguiente consulta obtendría el valor de las columnas “P_No” y “P_Nombre” de la tabla “Proveedor” :

```
SELECT P_No, P_Nombre
FROM Proveedor;
```

C.- Recuperación de valores únicos con SELECT DISTINCT

La claúsula “DISTINCT” nos devuelve valores únicos. En una tabla, una columna (exceptuando el caso de que sea una clave) puede contener generalmente valores duplicados; y algunas veces sólo se necesita un listado de los valores diferentes. La siguiente consulta devolvería los valores únicos de la columna “A-No” de la tabla “Vende” :

```
SELECT DISTINCT A_No
FROM Vende;
```

Si se especifica la palabra clave “ALL”, la consulta no eliminará los valores duplicados. Éste es el comportamiento por defecto cuando no se especifica expresamente ninguno de los dos (“ALL” o “DISTINCT”).

D.- Consulta con Valores Calculados

Es posible realizar “consultas con valores calculados”, en las que se incluya una expresión con operaciones como suma, resta, multiplicación o división, aplicadas sobre una o varias columnas de tipo numérico. Se puede nombrar la columna calculada con un “alias”, usando la cláusula “AS”. En general, los “alias” son nombres alternativos temporales que se dan a una tabla o una columna de una tabla, en el marco de una consulta, con la finalidad de hacer más legibles los nombres de las columnas o tablas, o para simplificar las sentencias SQL cuando los nombres de tablas o columnas son largos o complicados. El tiempo de vida de un “alias” se limita a la ejecución de la consulta en la que aparece.

Ejemplo :

```
SELECT A_No, A_Precio*1.2 AS  PRECIO_DOLARES
FROM Articulo;
```

E.- Consultas con selección de filas (cláusula WHERE):

Es posible realizar consultas en las que se restrinja el número de filas a recuperar, mediante el uso de la cláusula “WHERE”, que permite 5 tipos de condiciones de búsqueda : **comparación, rango, pertenencia a conjuntos, coincidencia de patrones, condición Null.**

E.1.- Consulta con condición de búsqueda compuesta :

La siguiente consulta devolvería el nombre y CIF de aquellos proveedores cuya sede está en Murcia o Jaén :

```
SELECT P_Nombre, P_CIF
FROM Proveedor
WHERE Ciudad = ‘Murcia’ OR Ciudad = ‘Jaén’;
```

E.2.- Consulta con condición de búsqueda por rango :

Obtención del nombre de todos los artículos cuyo precio esté entre 500 y 1500 :

```
SELECT A_Nombre
FROM Articulo
WHERE A_Precio BETWEEN 500 AND 1500;
```

La consulta anterior también se podría expresar así en SQL :

```
SELECT A_Nombre
FROM Articulo
WHERE A_Precio >=500 AND A_Precio <=1500;
```

Existe también en SQL el operador “NOT BETWEEN”, que verifica los valores que se encuentren fuera del rango indicado.

E.3.- Consulta con condición de búsqueda por pertenencia a conjuntos (IN/NOT IN):

Obtención del nombre y teléfono de todos los proveedores que pertenecen a los sectores “Material de Oficina” o “Electrónica” :

```
SELECT P_Nombre, Telefono
FROM Proveedor
WHERE Sector IN (‘Material de Oficina’, ‘Electrónica’);
```

E.4.- Consulta con condición de búsqueda por coincidencia con patrones (LIKE/NOT LIKE):

En las cláusulas “WHERE”, se usa el operador “LIKE” para buscar en función de la concordancia con un determinado patrón en una columna. Hay dos símbolos especiales que se usan en combinación con el operador “LIKE” :

- El signo de porcentaje “%” representa cero, uno o múltiples caracteres.
- El signo guión bajo “_” representa un único carácter.

Obtención del nombre y correo electrónico de todos los proveedores cuya dirección contenga la palabra “PLAZA” :

```
SELECT P_Nombre, Correo
FROM Proveedor
WHERE Direccion LIKE ‘%PLAZA%;
```

En contraposición, el operador “NOT LIKE” permite buscar los valores que no se ajusten al patrón indicado.

E.5.- Consulta con condición de búsqueda por valores nulos o no nulos (IS NULL/IS NOT NULL):

Cuando se diseña una tabla en la base de datos, una de las propiedades que se establecen para los campos de la tabla –dependiendo de la información que guardan- es si pueden contener o no un valor nulo. El operador "NULL" permite establecer en la cláusula "WHERE" de una consulta SQL condiciones para filtrar por campos de valor nulo o ausente.

Sirva para ilustrar este punto el siguiente ejemplo, que es una consulta que obtiene el nombre y precio de todos aquellos artículos cuyo precio sea mayor que 100 y de los que no se disponga de información sobre su peso :

```
SELECT  A_Nombre, A_Precio
FROM Articulo
WHERE  A_Precio>100 AND Peso is NULL;
```

3.2.- Uso de funciones agregadas

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos. Una función de agregación es una función que resume las filas de un grupo en un solo valor. El estándar ISO define 5 funciones agregadas - que se encuentran en todos los gestores de bases de datos relacionales-, las cuales toman por argumento una columna, y devuelven un valor único como resultado:

- **COUNT**: devuelve el número total de filas seleccionadas por la consulta.
- **MIN**: devuelve el valor mínimo del campo que especifiquemos en el argumento.
- **MAX**: devuelve el valor máximo del campo que especifiquemos en el argumento.
- **SUM**: suma los valores del campo que especifiquemos en el argumento. Sólo se puede utilizar en columnas numéricas.
- **AVG**: devuelve el valor promedio del campo que especifiquemos en el argumento. Sólo se puede utilizar en columnas numéricas.

A excepción de COUNT(*), todas las funciones agregadas eliminan en primer lugar los valores nulos, y actúan sólo sobre los valores restantes. COUNT(*) es un caso especial de la función COUNT, que cuenta todas las filas o registros de una tabla, independientemente de si contienen valores nulos o no.

Es de destacar que no se pueden usar funciones agregadas directamente en las clasusulas "WHERE", pero para ese propósito se emplean las clasulas "HAVING" (que requieren efectuar una partición en grupos mediante "GROUP BY"). Sí se permite incluir, dentro de una clasusula "WHERE", subconsultas que contengan una función agregada.

El siguiente ejemplo muestra una consulta para la obtención del número total de proveedores cuya facturación supera 1.000 :

```
SELECT COUNT(*) AS Total
FROM Proveedor
WHERE Facturacion > 1000;
```

En este otro ejemplo, se obtendría la facturación máxima, mínima y media de todos los proveedores :

```
SELECT MAX(Facturacion) as factMAX, MIN(Facturacion) as factMIN, AVG(Facturacion) as avgFACT
FROM Proveedor
```

Un detalle a tener en cuenta es que las funciones agregadas sólo se pueden usar : en la lista del "SELECT" y en la cláusula "HAVING". Por otro lado, no se permite entremezclar en la lista del "SELECT" funciones agregadas con nombres de columnas, a menos que se incluya en el SELECT una cláusula "GROUP BY" (la cual divide las columnas de una tabla en grupos), en cuyo caso se podrá incluir en la lista del "SELECT" : cualquiera de las columnas usadas para la agrupación, así como funciones agregadas. Veremos ejemplos que permitirán clarificar esto último en el siguiente punto.

3.3.- Agrupamiento de Registros (GROUP BY...HAVING)

SQL nos permite particionar los registros de una tabla en grupos. En estas condiciones, las funciones agregadas no se calculan sobre todos los valores de una columna especificada, sino únicamente sobre los valores de un solo grupo. Es decir, la función agregada se calcula individualmente para cada grupo.

La cláusula "GROUP BY…HAVING" permite agrupar en un solo registro todos aquellos registros cuyo valor del campo o campos indicados como parámetro sean idénticos. Opcionalmente, concede la posibilidad de filtrar únicamente los registros resultantes de la agrupación que cumplan una condición dada, expresada mediante la cláusula "HAVING".

Cabe destacar que cuando se usa "GROUP BY", cada elemento de la lista que sucede al "SELECT" deberá tener un único valor por cada grupo. Es decir, la cláusula "SELECT" sólo podrá contener una lista de :

- nombres de columnas,
- funciones agregadas,
- constantes,
- una expresión constituida por combinaciones de las anteriores.

Para asegurar esa unicidad de los valores, todos los nombres de columnas que aparezcan en la lista del "SELECT" deberán estar presentes en la cláusula "GROUP BY", a menos que se empleen únicamente en una función agregada. Sin embargo, podría haber nombres en la cláusula "GROUP BY" que no se encuentren presentes en la lista de elementos del "SELECT". El estándar ISO de SQL considera que, en el ámbito de la cláusula GROUP BY, dos NULLS son iguales.

En el ejemplo ilustrativo que figura a continuación, se muestra una consulta que devuelve para cada sector con más de 2 proveedores: el nombre del sector, el total de proveedores y la suma de sus facturaciones.

```
SELECT Sector, COUNT(P_No) AS TotalProv, SUM(Facturacion) AS TotalFact
FROM Proveedor
GROUP BY Sector
HAVING COUNT(P_No > 2);
```

La cláusula "WHERE" y la cláusula "HAVING" hacen funciones similares, pero para propósitos diferentes, y una consulta puede contener ambas cláusulas. La cláusula "WHERE" (opcional) se aplica como un filtro inicial que opera sobre las filas individuales de las tablas, y sólo pasarán a la etapa de agrupamiento las filas que cumplan las condiciones de la cláusula "WHERE". A su vez, la cláusula "HAVING" se aplica a continuación a las filas del conjunto de resultados del "WHERE". "HAVING" se utiliza para restringir la búsqueda a aquellos grupos que satisfagan la condición indicada en la misma, en la que sólo se podrán usar aquellas columnas que : aparezcan en la cláusula "GROUP BY", o bien

estén contenidas en una función de agregación. En definitiva, todo ello se traduce en que : sólo aparecerán en el resultado de la consulta los grupos que cumplan las condiciones del “HAVING”, y para las filas que hayan pasado el filtro previo de la clausula “WHERE”.

En consecuencia con esto, y a efectos prácticos, cada expresión de restricción de búsqueda que utilice únicamente atributos planos deberá ser incluida en la cláusula “WHERE”, mientras que todas las expresiones que empleen funciones agregadas deberemos ubicarlas en la clausula “HAVING”. Esto es así porque “WHERE” no permite el uso de funciones agregadas, y por otro lado, las cláusulas “HAVING” siempre incluirán en la práctica al menos una función agregada, puesto que en caso contrario la condición de búsqueda podría desplazarse a la cláusula “WHERE”, y sería innecesario incluir un “HAVING”.

Recogemos a continuación otro ejemplo más completo, en el que se usan tanto la cláusula “WHERE” como “GROUP BY” y “HAVING”. Si deseamos obtener la cantidad de artículos - sin considerar los que tienen precio nulo- por cada categoría, excluyendo las categorías que tengan un peso medio inferior o igual a 50, se realizará la siguiente consulta :

```
SELECT Categoría, COUNT(*)
FROM Articulo
WHERE PRECIO IS NOT NULL
GROUP BY Categoria
HAVING AVG(Peso)>50;
```

3.4.- Ordenación de Resultados (ORDER BY)

La claúsula “ORDER BY” se utiliza para ordenar los resultados de una consulta, por la lista de columnas especificada, las cuales deberán estar separadas por comas. Por defecto, el operador “ORDER BY” ordena de forma ascendente, según los valores de la lista de columnas, lo cual equivale a usar la palabra clave “ASC”. Si se quiere ordenar por orden descendente se añadirá la palabra clave “DES”.

Se permite incluir más de un elemento en la cláusula “ORDER BY”. Si la primera columna contiene sólo valores únicos, no es necesario añadir campos adicionales para controlar la ordenación. Sin embargo, si los valores de la primera columna no son únicos, podría haber varias filas en la tabla de resultados con el mismo valor para la primera columna, y puede ser útil ordenar también dichas filas de acuerdo con una columna adicional.

El siguiente comando SQL obtiene una lista ordenada de proveedores (con su nombre, CIF y teléfono), ordenados por ciudad, y secundariamente, por orden descendiente de facturación :

```
SELECT P_Nombre, CIF, Telefono
FROM Proveedor
ORDER BY Ciudad, Facturacion DESC;
```

3.5.- Subconsultas

Una “subquery” o subconsulta se define como una consulta ubicada dentro de otra consulta, y que sirve para determinar el contenido del resultado final. Las subconsultas pueden a su vez anidarse dentro de otra subconsulta. Cada subconsulta produce una tabla temporal, que no se almacena y sólo puede ser accedida en el marco de la consulta.

Las subconsultas se pueden incorporar dentro de las consultas de tipo “SELECT”, “INSERT”, “UPDATE” y “DELETE”. Las subconsultas serán siempre sentencias “SELECT”, que deberán ir rodeadas de paréntesis.

Como norma, las subconsultas podrán estar ubicadas en las siguientes posiciones de la consulta principal : en la lista de selección del “SELECT”, en la cláusula “WHERE”, o en la cláusula “HAVING”.

En las clásulas “WHERE” y “HAVING”, se permite el uso de subconsultas en cualquier lugar donde se espere un valor o conjunto de valores. En este caso, el valor se derivará de la evaluación previa de la subconsulta.

De manera general, una subconsulta interna se ejecuta antes que su subconsulta padre, de tal manera que los resultados de una subconsulta interna se pasan a la subconsulta externa que le precede. Cabe destacar que las “SELECT” de las subconsultas presentan las siguientes restricciones o particularidades respecto a las “SELECT” generales :

- En las subconsultas no se permite la clausula “GROUP BY”, ya que sus resultados no son visibles al usuario.
- La lista del “SELECT” debe constar de un único nombre de columna, excepto en las subconsultas que emplean “EXISTS” (ver apartado 3.8), en las que se podrá usar el operador “*”.
- Por defecto, los nombres de columnas en una subconsulta se refieren a la tabla cuyo nombre aparece en la clausula “FROM” de la subconsulta. No obstante, también se permite dentro de una subconsulta, hacer referencia al valor de una columna que no se corresponda con ninguna columna de las tablas designadas en la “FROM” de la subconsulta, sino con una columna de las tablas indicadas en la clausula “FROM” de la consulta principal. Ese nombre de columna se denomina referencia externa.
- Cuando una subconsulta sea uno de los dos operadores que forman parte de una comparación, la subconsulta deberá aparecer en el lado derecho del operador de comparación.

El estándar SQL define tres tipos de subconsultas:

- Subconsultas de fila. Son aquellas que devuelven más de una columna pero una única fila.
- Subconsultas de tabla. Son aquellas que devuelve una o varias columnas y cero o varias filas.
- Subconsultas escalares. Son aquellas que devuelven una columna y una fila, es decir, un valor único.

Según el valor de retorno de la subconsulta sea simple o múltiple, el operador de selección que se utilice en la condición de selección del “WHERE” o “HAVING” que precede a la subconsulta deberá ser del tipo apropiado, según la tabla siguiente:

Retorno de la subconsulta	**Operador de selección**
Valor simple	De tipo aritmético (=,<>,<,>,<=,<=)
Más de un valor	De tipo lógico (IN , EXISTS, ANY, ALL)

A modo de ejemplo, el siguiente comando SQL, que se corresponde con una consulta escalar, permite obtener los artículos cuyo precio sea superior al del artículo llamado “TELEVISOR LED” :

```
SELECT *
FROM Articulo
WHERE A_Precio > (SELECT A_Precio
                  FROM Articulo
                  WHERE A_Nombre = ‘TELEVISOR LED MODELO ABC123’);
```

3.6.- Comparación con conjuntos (ANY/SOME Y ALL)

Los operadores “ANY” y “ALL” permiten realizar una comparación entre el valor de una columna y un conjunto de otros valores. Las palabras clave “ANY” y “ALL” se utilizan siempre dentro de las clausulas “WHERE” y “HAVING”, deben ir precedidas de un operador de comparación, y actúan sobre subconsultas que devuelven múltiples valores. Tanto “ANY” (que también se puede denominar “SOME” según el estándar ISO) como “ALL” devuelven un valor booleano como resultado. La diferencia entre ambos es la siguiente :

- “ANY” (o “SOME”) devuelve “True” si cualquiera de los valores de la subconsulta satisfacen la condición.
- “ALL” devuelve “True” si todos los valores de la subconsulta cumplen la condición.

Por ejemplo, la siguiente consulta devuelve el nombre y CIF de todos los proveedores cuya facturación sea mayor que la máxima de los proveedores situados en Madrid, mostrando los resultados ordenados por facturación :

```
SELECT P_Nombre, CIF
FROM Proveedor
WHERE Facturacion > ALL (SELECT Facturacion
                         FROM Proveedor
                         WHERE Ciudad = ‘Madrid’)
ORDER BY Facturacion;
```

3.7.- Consultas combinadas o consultas en múltiples tablas (JOIN)

Frecuentemente, cuando necesitamos recuperar la información de una base de datos nos encontramos con que dicha información se encuentra repartida en varias tablas. La instrucción “JOIN” se utiliza para enlazar los datos de dos o más tablas relacionadas a través de algún campo en común (típicamente una clave foránea), obteniéndose como resultado de esta operación una vista con filas que mezclan datos provenientes de dichas tablas, obtenidos de aquellos registros que cumplan la condición de comparación establecida sobre unos atributos comunes a las tablas. Estos atributos comunes aparecen una sola vez en el resultado.

El principio de las sentencias “JOIN” de SQL se basa en la operación homónima del álgebra relacional, la cual es una combinación de : producto cartesiano, más una selección (o restricción) basada en una condición de comparación sobre los atributos comunes, y por último una proyección eliminando los atributos comunes duplicados. La condición necesaria para que se pueda realizar un “JOIN” sobre dos tablas es que estas presenten, al menos, un campo en común sobre el que establecer la condición de comparación.

En SQL, para realizar una consulta combinada o “join”, basta con : incluir más de una tabla en la cláusula “FROM” de un “SELECT” -usando la coma como separador-, y aplicar una condición de combinación a través de una cláusula “WHERE”. El operador de comparación más usado en los “JOINs” suele ser el “=”, si bien se podrían también utilizar el resto de operadores de comparación (“>”, “<”, etc). En la siguiente tabla se presentan los operadores de comparación permitidos en las sentencias “JOIN” :

Operador de comparación	Significado
=	Igual
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Diferente a
¡=	Diferente a

También se permite emplear en las consultas combinadas un “alias” para las tablas que se incluyan en una clausula “FROM”, y ese “alias” iría separado del nombre de la tabla por un espacio. Los “alias” son utiles para identificar inequívocamente el nombre de una columna en los casos en los que coincida el mismo nombre de columna en dos o más tablas diferentes.

Veámoslo con un ejemplo. Si queremos obtener la relación de todos los proveedores y los artículos que venden, realizaríamos una consulta combinada sobre tres tablas (“Proveedor”, “Artículo” y “Vende”), enlazadas mediante sus atributos comunes, mediante la siguiente instrucción :

```
SELECT P.P_Nombre, A.A_Nombre
FROM Proveedor P, Vende V, Articulo A
WHERE P.P_No = V.P_No AND
        A.A_No = V.A_No;
```

Como se puede observar, en la clausula “FROM” se ha añadido un “alias” al nombre de cada tabla, porque existen atributos con nombre coincidente en tablas diferentes (“P_Nombre”, y “A_Nombre”). La forma de calcular el resultado de una consulta combinada es siempre el siguiente : primero se calcula el producto cartesiano de las tablas indicadas, a continuación se restringe dicho conjunto a las filas que satisfagan las condiciones expresadas en la clausula “WHERE”, posteriormente se eliminan las columnas comunes repetidas, y por último se escogen las columnas indicadas en la clasula “SELECT”.

Los “alias” presentan otra utilidad : para realizar un "JOIN" no es imprescindible utilizar 2 tablas distintas, podemos también combinar los registros de una misma tabla, y en ese caso deberemos utilizar en la consulta “SELECT” dos “alias” para la misma tabla.

La consulta combinada del ejemplo anterior fue realizada usando la sintaxis general de la instrucción SELECT que vimos al inicio del apartado 3.1, pero hay otra forma adicional - que se incorporó a partir de ANSI SQL-92-, la cual es más explícita e ilustrativa a la hora de realizar este tipo de combinaciones, y se basa en la inclusión de una cláusula llamada “JOIN” en la cláusula “FROM”. Existen 3 formas alternativas de especificar el JOIN correspondiente a la consulta combinada del ejemplo anterior (en todas ellas, se modificaría únicamente la parte de la consulta “SELECT” anterior que va desde “FROM” hasta el final) :

1.- **FROM** Proveedor P **JOIN** Vende V **ON** P.P_No = V.P_No **JOIN** Articulo A **ON** V.A_No = A.A_No;

2.- **FROM** Proveedor P **NATURAL JOIN** Vende V **NATURAL JOIN** Articulo A

3.- **FROM** Proveedor P **JOIN** Vende V **USING** P_No **JOIN** Articulo A **USING** A_No

La primera alternativa (basada en el uso de “ON”, que identifica las columnas comunes en las tablas) es la que se usa más habitualmente, y es la más versátil.

El “NATURAL JOIN” es un tipo de “JOIN” que combina dos o más tablas a partir de los nombres de columnas que tengan el mismo nombre y tipo en las tablas. En este tipo de consultas, no es

necesario especificar los nombres de columnas a combinar, y se escogerán automáticamente todos los que coincidan, para calcular el resultado de la consulta.

Como hemos visto, el "NATURAL JOIN" usa todas las columnas con nombre y tipo coincidentes a la hora de realizar el "JOIN". Es un tipo de consulta combinada que generalmente no se recomienda, ya que no proporciona control sobre las columnas, y presenta el riesgo de que cuando se realicen cambios en el esquema de alguna de las tablas, los cuales den lugar a la coincidencia inadvertida de nombres de columnas, eso causará que el NATURAL JOIN combine automáticamente las tablas a partir de dicha columna o columnas.

Si queremos restringir la combinación, buscando únicamente la coincidencia en una columna determinada, se puede emplear como alternativa al "ON" la clausula "USING", que permite abreviar el código cuando el nombre de la columna sea coincidente en las dos tablas, pero tiene la desventaja de que sólo permite realizar "JOINs" basados en la condición de igualdad, no soportando el resto de operadores de comparación.

Como puede advertirse en los ejemplos mostrados, también es posible anidar consultas combinadas ("JOINs"). Es decir, una de las tablas de un JOIN puede ser a su vez una tabla resultante de otro JOIN, lo cual permite la codificación de un "JOIN" de tres o más tablas, presentando como resultado al usuario una sola tabla combinada.

Las 3 formas alternativas descritas ("ON", "NATURAL JOIN", "USING") pueden emplearse en todos los tipos de consultas combinadas. Existen cinco tipos de "JOINs" en el estándar ANSI de SQL ("Cross Join", "Inner Join", "Left Outer Join", "Right Outer Join", "Full Outer Join"), y vamos a estudiarlos a continuación. Para facilitar su comprensión, vamos a limitar generalmente la explicación al caso de un "JOIN" realizado sobre dos tablas, y en el que la condición de comparación se trate de una igualdad entre dos columnas, pero el funcionamiento sería análogo para las consultas combinadas sobre varias tablas, y usando otras condiciones de comparación.

▪ CROSS JOIN

En el lenguaje SQL, un "CROSS JOIN" entre dos tablas es una operación que se corresponde con el producto cartesiano de dichas tablas.

En el algebra relacional, dadas dos relaciones o tablas "R" y "S", se define el producto cartesiano como : la relación resultante de combinar cada fila de "R" con todas las de "S", de tal manera que sus atributos corresponden a los de "R" seguidos por los de "S". El resultado de esta operación es una tabla formada por un conjunto de tuplas ordenadas, en la que cada tupla de la tabla "R" se combina con cada una de las tuplas de la tabla "S". Es decir, si hacemos el "CROSS JOIN" de dos tablas que tengan "M" y "N" filas respectivamente, la tabla resultante tendrá (M X N) filas.

La sintaxis para el producto cartesiano o "CROSS JOIN" en SQL es la siguiente (como vemos, para este tipo de join no se incluye una condición de comparación, o lo que es lo mismo, no tienen una cláusula "ON") :

SELECT [DISTINCT | ALL] {* | columnList}
FROM TableName1 **CROSS JOIN** TableName2
[**WHERE** condición]

Veamos un ejemplo de "CROSS JOIN". Si queremos obtener todas las combinaciones posibles de proveedores y artículos, sin tener en cuenta si el proveedor vende o no cada artículo, haríamos :

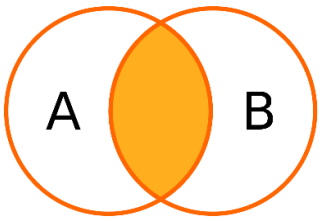
```
SELECT P.P_No AS 'NOMBRE PROVEEDOR', A.A_No AS 'NOMBRE ARTICULO'
FROM Proveedor AS P
CROSS JOIN Articulo AS A;
```

Analizaremos a continuación los "INNER JOINs", que son operaciones de combinación que preservan únicamente las tuplas que presentan una correspondencia en las dos tablas.

▪ INNER JOIN

Es el tipo de "JOIN" por defecto, y el más comunmente utilizado. Una "INNER JOIN" (que es intercambiable por el término "JOIN" en SQL) devuelve únicamente la combinación de aquellos registros de las tablas que cumplan la condición de selección en los dos campos que se comparan. Dicho de otro modo, esta cláusula busca coincidencias entre 2 tablas en función de una columna común, de tal manera que únicamente la **intersección** se mostrará en los resultados. La condición de selección o comparación del "JOIN" se establece en la clausula "ON" de la instrucción "SELECT".

El siguiente diagrama de Venn ilustra el concepto de "INNER JOIN" de dos tablas "A" y "B":



Por ejemplo, la siguiente consulta devolverá el nombre de todos los trabajadores y el departamento en el que trabajan, siempre que su salario sea mayor de 30.000 :

```
SELECT Per_Nombre, Per_Apellido1, Per_Apellido2, Dep_Nombre
FROM Personal
INNER JOIN Departamento
ON Personal.Dep_No = Departamento.Dep_No
WHERE Personal.Per_Salario>30000;
```

Esta última consulta también se podría haber realizado con una clausula "WHERE", sin hacer uso de una "INNER JOIN" :

```
SELECT Per_Nombre, Per_Apellido1, Per_Apellido2, Dep_Nombre
FROM Personal, Departamento
WHERE       Personal.Dep_No = Departamento.Dep_No
            AND Personal.Per_Salario>30000;
```

Sin embargo, el uso de la clausula "ON" en el código ayuda a hacerlo más claro, legible y fácil de comprender que si se incluyeran tanto las condiciones del "SELECT" como las condiciones de combinación en una sola clausula "WHERE", separadas por el operador "AND". La cláusula "ON" hace explícitas cuáles son las condiciones de combinación de la consulta.

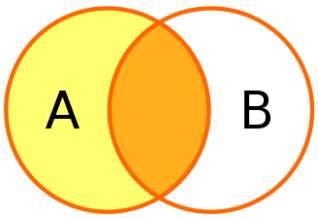
En muchos casos, podemos desear realizar consultas en las que no se pierdan los datos de los registros que aparezcan únicamente en una de las dos tablas, sin presentar una correspondencia con

la otra. En esas situaciones, se pueden emplear los “OUTER JOINs” (de los cuales hay tres tipos : “LEFT OUTER JOIN”, “RIGHT OUTER JOIN” y “FULL OUTER JOIN”), que analizaremos seguidamente.

▪ LEFT OUTER JOIN

En una “LEFT OUTER JOIN” (también denominada “LEFT JOIN”) entre dos tablas se obtiene : la combinación de aquellos registros de las tablas que cumplan la condición de selección en los dos campos que se comparan, y también todas aquellas filas de la primera tabla que no presentan una coincidencia con la segunda tabla. Si existen valores en la primera tabla que no se corresponden con ningún registro de la segunda tabla, se mostrará el valor “NULL” en la ubicación correspondiente a los campos de la segunda tabla de los registros resultantes de la combinación.

El siguiente diagrama de Venn refleja el concepto de LEFT OUTER JOIN de dos tablas “A” y “B”:



Para entender mejor el funcionamiento, la operación “LEFT OUTER JOIN” se procesa de acuerdo con el siguiente algoritmo :

- En primer lugar, se calcula el resultado del INNER JOIN.
- A continuación, por cada registro “R” de la tabla de la izquierda (o primera tabla) que no presente una coincidencia con ningún registro de la tabla de la derecha, añadir un nuevo registro “RJ” al resultado del JOIN, que se construiría como sigue :
    - Los campos del registro “RJ” que se han obtenido de la tabla de la izquierda se rellenan con los valores de los campos homónimos del registro “R”.
    - El resto de campos de “RJ” se rellenan con valores nulos.

Por ejemplo, la siguiente consulta devolverá un listado de los nombres de los clientes, con los datos de los pedidos que hayan realizado, en su caso. Para esta consulta, emplearemos el operador LEFT OUTER JOIN, dado que éste nos garantiza que todos los clientes aparecerán en el resultado, con independencia de que hayan cursado o no pedidos hasta el momento, y esto es lo que deseamos.

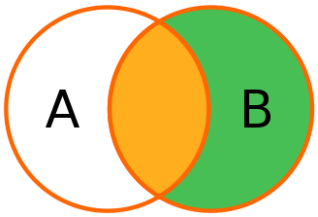
```
SELECT C.C_Nombre, PC.PC_No, PC.Fecha, PC.PC_Precio
FROM Cliente C
LEFT OUTER JOIN Pedido_Cliente PC
ON C.C_No = PC.C_No
ORDER BY C.C_Nombre, PC.PC_No;
```

▪ RIGHT OUTER JOIN

Se trata de una operación simétrica al “LEFT OUTER JOIN”. En una “RIGHT OUTER JOIN” (también denominada “RIGHT JOIN”) entre dos tablas se obtiene : la combinación de aquellos registros de las tablas que cumplan la condición de selección en los dos campos que se comparan, y también todas aquellas filas de la segunda tabla que no presentan una coincidencia con la primera tabla. Si existen valores en la segunda tabla que no se corresponden con ningún registro de la primera tabla, se

mostrará el valor “NULL” en el lugar correspondiente a los campos de la primera tabla de los registros resultantes de la combinación.

El siguiente diagrama de Venn refleja el concepto de “RIGHT OUTER JOIN” de dos tablas “A” y “B”:

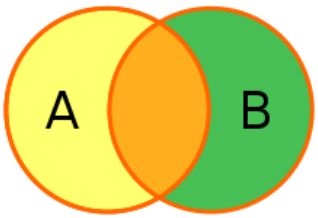


Por ejemplo, la siguiente consulta devolverá el nombre de todos los trabajadores y el departamento en el que trabajan, pero también incluirá en el listado todos los departamentos de la compañía que no tengan ningún trabajador (en caso de que no haya ningún empleado trabajando en un departamento determinado, se mostrará “NULL” en los campos correspondiente a la tabla Personal) :

```
SELECT  P.Per_Nombre AS 'Empleado',  D.Dep_Nombre as 'Departamento'
FROM Personal P
RIGHT JOIN Departamento D
ON P.Dep_No = D.Dep_No;
```

- **FULL OUTER JOIN**

El “FULL OUTER JOIN” (también denominado “FULL JOIN”) se corresponde con la suma de un “RIGHT OUTER JOIN” y un “LEFT OUTER JOIN”. Es decir, se trata de una operación que devuelve: los registros de las dos tablas que cumplen la condición de selección, y además también el resto de registros, tanto de la tabla derecha como de la izquierda (mostrando “NULL” en los campos de la tabla correspondiente cuando no haya una coincidencia). Un detalle a tener en cuenta es que los “FULL OUTER JOINS” presentan el problema potencial de que pueden llegar a devolver un conjunto de resultados muy extenso. El siguiente diagrama ayuda a la comprensión del concepto de “FULL OUTER JOIN” :



Veamos un ejemplo del uso de un FULL OUTER JOIN. Esta consulta devolverá todos los trabajadores de los proveedores (y el nombre del proveedor correspondiente), pero también mostrará los proveedores que no tienen ningún trabajador en plantilla, así como los trabajadores que no están asociados a ningún proveedor :

```
SELECT        Proveedor.P_Nombre, Personal.Per_Nombre, Personal.Per_Apellido1,
              Personal.Per_Apellido2
FROM Proveedor
FULL OUTER JOIN Personal
ON Proveedor.P_No=Personal.P_No;
```

Para concluir nuestro análisis de los “JOINS”, es importante destacar que las cláusulas “ON” y “WHERE” (que como vimos podían realizar una función equiparable en el caso de los “INNER JOINs”), tienen sin embargo un compartamiento diferente para los “OUTER JOINs”, en sus tres variaciones. Esto es así ya que los “OUTER JOINs” añaden registros rellenados con valores nulos, únicamente para aquellas tuplas que no contribuyen al resultado del “INNER JOIN” correspondiente. Por tanto, en el caso de los “OUTER JOINs” no existe una formulación alternativa de la consulta empleado una clasula “WHERE”, como sucedía con los “INNER JOINs”.

3.8.- Existencia o inexistencia de registros (EXISTS, NOT EXISTS)

La cláusula “EXISTS” permite verificar la existencia de algún registro en una subconsulta. Es decir, el operador “EXISTS” devuelve verdadero si la subconsulta devuelve uno o más registros. El operador “NOT EXISTS” es el opuesto de “EXISTS”. En consecuencia, “NOT EXISTS” devuelve VERDADERO si la subconsulta no devuelve ninguna fila.

A modo de ejemplo, la siguiente consulta obtiene la lista de proveedores que tienen en plantilla trabajadores cuyo salario es mayor de 100.000 :

```
SELECT P_No
FROM Proveedores
WHERE EXISTS (SELECT Per_No
                FROM Personal
                WHERE Personal.P_No = Proveedores.P_No AND Personal.Per_Salario > 100000);
```

Llegados a este punto, es importante señalar una diferencia entre el comportamiento de “IN” y “EXISTS”, en lo que respecta al tratamiento de los valores “NULL”. Semánticamente, “IN” solicita aquellos registros cuyo valor de un campo esté contenido dentro de un conjunto de valores (los cuales pueden venir indicados expresamente, separados por comas, o bien definirse como el resultado de un SELECT). Por su parte, la cláusula “EXISTS” incluye un registro si la subconsulta que le sigue devuelve un valor, sea cual sea, incluído “NULL”. En definitiva, “IN” no puede evaluar “NULL”s, por lo que las filas que los contengan se consideran siempre como falsas, y no se devuelven en el resultado del “SELECT”. Sin embargo, “EXISTS” puede evaluar “NULL”s, y considera que se cumple la condición de existencia en los registros resultantes de la subconsulta que contengan valores nulos.

3.9.- Combinando tablas de resultados (UNION, INTERSECT, EXCEPT)

SQL dispone de instruciones que permiten calcular la unión, la intersección y la diferencia (operaciones de la teoría de conjuntos y el algebra relacional), sobre las tuplas derivadas de dos subconsultas :

- El operador “UNION” sirve para combinar los resultados de dos subconsultas independientes, devolviendo como resultado conjunto la totalidad de registros obtenidos por ambas. Este operador elimina automáticamente los registros duplicados del resultado final.
- El operador “INTERSECT” devuelve los registros distintos devueltos por las consultas y que sean comunes a ambas.
- “EXCEPT” devuelve los registros de la consulta “SELECT” de la izquierda que no están presentes en los resultados devueltos por la consulta “SELECT” situada al lado derecho del operador “EXCEPT”.

Se pueden combinar dos o más sentencias “SELECT” en una consulta compuesta si cumplen las siguientes condiciones de “compatibilidad de unión”:

- Los conjuntos de resultados de ambas consultas deben tener el **mismo número de columnas**.
- Las columnas correspondientes de las dos consultas deben tener el **mismo tipo de datos y longitud**, o deben ser convertibles implícitamente al mismo tipo de datos. Es responsabilidad del usuario asegurar que los valores de los datos, en las columnas correspondientes de dos tablas que se desan combinar, provengan del mismo dominio.

Los nombres de las columnas en el conjunto de resultados se corresponderán con los de la primera consulta. En caso de que se desee cambiar el nombre a una columna del resultado, se utilizará una cláusula AS (esto es, un "alias") en la primera consulta.

En el estándar ISO, los tres operadores de conjuntos se denominan "UNION", "INTERSECT" y "EXCEPT". El formato de los tres operadores es :

operator [ALL] [CORRESPONDING [BY {column1 [, . . .]}]]

Si se incluye "CORRESPONDING BY" en la operación de conjuntos, ésta se realizará sólo sobre las columnas especificadas a continuación. Si se indica "CORRESPONDING", pero no "BY", la operación de conjuntos se efectuará sobre todas las columnas comunes a las dos tablas ("CORRESPONDING" se creó en SQL para permitir la unión de dos tablas que tengan los mismos campos – número, tipo y longitud-, pero en un orden cambiado). Por otro lado, si se indica "ALL", el resultado podrá incluir filas duplicadas.

Para ilustrar el uso del operador "UNION", la siguiente consulta obtendría el identificador, nombre y ciudad de todos los proveedores que estén en Alemania o Francia :

```
(SELECT P_No, P_Nombre, Ciudad
FROM Proveedor
WHERE Pais='Alemania')
UNION
(SELECT P_No, P_Nombre, Ciudad
FROM Proveedor
WHERE Pais='Francia');
```

La consulta anterior sería equivalente a :

```
(SELECT *
FROM Proveedor
WHERE Pais='Alemania')
UNION CORRESPONDING BY P_No, P_Nombre, Ciudad
(SELECT *
FROM Proveedor
WHERE Pais='Francia');
```

La consulta que mostramos a continuación, que es un ejemplo de uso del operador "INTERSECT", devuelve la lista de ciudades en las que existe un proveedor y un cliente :

```
(SELECT Ciudad
FROM Proveedor)
INTERSECT
(SELECT Ciudad
FROM Cliente);
```

Por último, esta consulta –que hace uso del operador EXCEPT-, obtiene una lista de todas las ciudades en las que hay algún proveedor pero no hay clientes :

```
(SELECT Ciudad
FROM Proveedor)
EXCEPT
(SELECT Ciudad
FROM Cliente);
```

3.10.- Actualización de bases de datos (INSERT, UPDATE, DELETE)

SQL es un lenguaje completo de manipulación de datos que permite tanto modificar como consultar los datos de una base de datos. Los registros de una tabla pueden ser modificados de tres modos: Crear nuevos registros, Modificarlos o bien Eliminarlos. Existen tres comandos SQL para realizar actualizaciones o modificaciones del contenido de una base de datos :

- INSERT : agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.
- UPDATE : modificar los valores de un conjunto de registros existentes en una tabla.
- DELETE : borra uno o más registros existentes en una tabla.

▪ INSERT

La adición de datos a una tabla se realiza mediante la instrucción INSERT. Su sintaxis fundamental es:

INSERT INTO nombreTabla [(listaDeCampos)]
VALUES (valor1 [,valor2 ...])

Si no se especifica la lista de campos, que es opcional, la lista de valores deberá seguir el orden de las columnas tal y como fueron creadas con la instrucción “CREATE TABLE” de DML. Los campos no rellenados explícitamente con la orden “INSERT”, se rellenarán con su valor por defecto (DEFAULT) – en caso de que exista-, o bien con un valor nulo, si no se estableció ningún valor por defecto y el campo no fue definido como NOT NULL en la creación de la tabla. Es decir, si algún campo tiene restricción de obligatoriedad (“NOT NULL”) y no lleva aparejado un valor por defecto, ocurrirá un error si no especificamos algún valor para dicho campo en una sentencia “INSERT”.

Por ejemplo, podríamos incluir la primera tupla en la tabla “Articulo” mediante la siguiente instrucción :

```
INSERT INTO Articulo(A_No, A_Nombre, A_Precio, Peso)
VALUES(1, ‘Portatil Alta Gama’, 3500, 1000);
```

Existe también otro formato de la instrucción INSERT, que permite copiar múltiples filas (procedentes de una o más tablas) a otra tabla, y tiene la siguiente sintaxis :

INSERT INTO nombreTabla[(listaDeCampos)]
SELECT . . .

Donde la cláusula “SELECT” del “INSERT” puede ser cualquier instrucción SELECT válida de SQL.

A título ilustrativo, la siguiente instrucción copiaría los valores –de nombre, ciudad y país- de los proveedores británicos a la tabla “Cliente” :

```
INSERT INTO Cliente (C_Nombre, Ciudad, Pais)
SELECT P_Nombre, Ciudad, Pais
FROM Proveedor
WHERE Pais='Gran Bretaña';
```

En este ejemplo no hemos indicado el valor de la clave primaria en el “INSERT”, porque partimos del supuesto de que cuando se definió la clave primaria al crear la tabla “Cliente”, se estableció que esta clave se incrementará automáticamente, y por tanto se añadirá un nuevo valor para la misma –sin necesidad de especificarlo - cada vez que se inserte un nuevo registro.

▪ UPDATE

La modificación de los datos de los registros de una tabla se efectúa en SQL mediante la instrucción UPDATE, cuya sintaxis es la siguiente :

UPDATE nombreTabla
SET columna1=valor1 [,columna2=valor2...]
[**WHERE** condición]

El nombre indicado a continuación de “UPDATE” es el de la tabla cuyos registros se quieren modificar. La clausula “SET” establece los nuevos valores para la columna o columnas que se van a actualizar. La clausula “WHERE” es opcional, de modo que si se incluye, sólo se actualizarán los registros que cumplan la condición indicada, mientras que si se omite, aplicará a todos los registros de la tabla.

Por ejemplo, la siguiente instrucción corrige el puesto del trabajador con nombre “Enrique” y apellidos “Martínez Muñoz”, pasándolo a “Jefe de Proyecto”.

```
UPDATE Personal
SET Puesto = 'Jefe de Proyecto'
WHERE Per_Nombre = ‘Enrique’
AND Per_Apellido1= ‘Martinez’
AND Per_Apellido2=’Muñoz’;
```

Esta sentencia incrementa un 16% el precio de los artículos pertenecientes la categoría “Electrónica” :

```
UPDATE Articulo
SET A_Precio = A_Precio + (A_Precio * 16/100)
WHERE Categoria = “Electrónica”;
```

▪ DELETE

La sentencia “DELETE” sirve para borrar filas de una tabla. Su sintaxis es la siguiente :

DELETE FROM nombre_tabla
[WHERE criterio_búsqueda]

La cláusula “WHERE”, que es opcional, se utiliza para indicar qué parte de los datos de la tabla se van a eliminar, los cuales serán aquellos que cumplan la condición “criterio_búsqueda”. Si no se establece una clásula “WHERE”, todos los datos de la tabla serán suprimidos. Sin embargo, esto no borraría la tabla propiamente dicha, para lo cual se requeriría una sentencia “DROP TABLE”.

Si solicitamos el borrado de un registro que no exista, es decir, si ningún registro cumple con la condición especificada, no se borrarán registros, al no haberse encontrado registros que cumplan la condición.

Por ejemplo, la siguiente instrucción borraría de la tabla “Articulo” todos los artículos pertenecientes a la categoría “Cosmética” :

```
DELETE FROM Articulo
WHERE Categoria=”Cosmética”;
```

Para concluir, esta sentencia borraría todos los registros en la tabla “Proveedor”, pero sin borrar la tabla :

```
DELETE FROM Proveedor;
```

4 RESUMEN ESQUEMÁTICO

- SQL es un lenguaje declarativo de 4G, que se ha convertido en el estándar para trabajar con bases de datos relacionales.

- Toda consulta formulada utilizando el cálculo relacional de tuplas o el álgebra relacional se puede realizar también mediante SQL.

- Las bases de datos relacionales permiten la ejecución de instrucciones SQL de dos formas : directa o interactiva, y también como parte de un programa de aplicación.

- SQL consta de los siguientes componentes : DDL, DML, DCL, TCL.

- El lenguaje DML tiene las siguientes sentencias : SELECT, UPDATE, INSERT, DELETE

- La instrucción “SELECT” se emplea en SQL para realizar consultas. Tiene la capacidad de combinar, en un solo comando, las operaciones del algebra relacional de Selección, Proyección y “Join”. Esta sentencia devuelve una tabla lógica.

- La cláusula “WHERE” de “SELECT” permite 5 tipos de condiciones de búsqueda : comparación, rango, pertenencia a conjuntos, coincidencia de patrones, condición Null.

- El estándar ISO define 5 funciones agregadas, que se encuentran en todos los gestores de bases de datos relacionales : COUNT, MIX, MAX, SUM, AVG.

- La cláusula “GROUPBY…HAVING” permite agrupar en un solo registro todos aquellos registros cuyo valor del campo o campos indicados como parámetro sean idénticos. Opcionalmente, concede la posibilidad de filtrar únicamente los registros que cumplan una condición dada, expresada mediante la cláusula “HAVING”.
    - A diferencia de la clausula WHERE, HAVING puede incluir funciones agregadas.

- Una “subquery” o subconsulta se define como una consulta ubicada dentro de otra consulta, y que sirve para determinar el contenido del resultado final. Las subconsultas se pueden incorporar dentro de las consultas de tipo “SELECT”, “INSERT”, “UPDATE” y “DELETE”.
    - Las subconsultas se pueden anidar. En las clásulas “WHERE” y “HAVING”, se permite el uso de subconsultas en cualquier lugar donde se espere un valor o conjunto de valores

- Los operadores “ANY” y “ALL” permiten realizar una comparación entre el valor de una columna y un conjunto de valores. Las palabras clave “ANY” y “ALL” se utilizan siempre dentro de las clausulas “WHERE” o “HAVING”, deben ir precedidas de un operador de comparación, y actúan sobre subconsultas que devuelven múltiples valores.

- La instrucción “JOIN” se utiliza para enlazar los datos de dos o más tablas relacionadas a través de algún campo en común, obteniéndose como resultado de esta operación una vista con filas que mezclan datos provenientes de dichas tablas, obtenidos de aquellos registros que cumplan la condición de comparación establecida sobre unos atributos comunes a las tablas.

- Existen cinco tipos de “JOINs” en el estándar ANSI de SQL (“Cross Join”, “Inner Join”, “Left Outer Join”, “Right Outer Join”, “Full Outer Join”)

    - CROSS JOIN se corresponde con el producto cartesiano de dos tablas.
    - INNER JOIN busca coincidencias entre 2 tablas en función de una columna común, de tal manera que únicamente se muestra la intersección.

  - LEFT OUTER JOIN deuelve la combinación de aquellos registros de las tablas que cumplan la condición de selección, y también todas aquellas filas de la primera tabla que no presentan una coincidencia con la segunda tabla.
  - RIGHT OUTER JOIN devuelve la combinación de aquellos registros de las tablas que cumplan la condición de selección, y también todas aquellas filas de la segunda tabla que no presentan una coincidencia con la primera tabla.
  - FULL OUTER JOIN se corresponde con la suma de un “RIGHT OUTER JOIN” y un “LEFT OUTER JOIN”

- El operador “EXISTS” devuelve verdadero si la subconsulta devuelve uno o más registros. El operador “NOT EXISTS” es el opuesto de “EXISTS”.

- SQL dispone de instruciones que permiten calcular la unión, la intersección y la diferencia, sobre las tuplas derivadas de dos subconsultas :
  - “UNION” sirve para combinar los resultados de dos subconsultas independientes, devolviendo como resultado conjunto la totalidad de registros obtenidos por ambas.
  - “INTERSECT” devuelve los registros distintos devueltos por las consultas y que sean comunes a ambas.
  - “EXCEPT” devuelve los registros de la consulta “SELECT” de la izquierda que no están presentes en los resultados devueltos por la consulta “SELECT” situada a la derecha del “EXCEPT”.

- Existen tres comandos SQL para realizar actualizaciones o modificaciones del contenido de una base de datos :
  - INSERT : agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.
  - UPDATE : modificar los valores de un conjunto de registros existentes en una tabla.
  - DELETE : borra uno o más registros existentes en una tabla.

4 GLOSARIO

Algebra relacional : el álgebra relacional es un lenguaje de consulta procedimiental constituido por un conjunto de operaciones que describen paso a paso cómo computar una respuesta sobre las relaciones o tablas, tal y como éstas son definidas en el modelo relacional. Las operaciones fundamentales, tanto básicas como derivadas, del álgebra relacional son : Selección, Proyección, Producto cartesiano, Unión, Diferencia, Intersección, Unión Natural (Natural Join), Divisón o Cociente, Agrupación.

Alias : nombres alternativos temporales que se dan a una tabla o una columna de una tabla, en el marco de una consulta, con la finalidad de hacer más legibles los nombres de la columnas o tablas, o para simplificar las sentencias SQL cuando los nombres de tablas o columnas son largos o complicados. El tiempo de vida de un "alias" se limita a la ejecución de la consulta en la que aparece.

Base de Datos : sistema formado por un conjunto de datos organizados y relacionados entre sí, que son almacenados sistemáticamente para su posterior uso, así como un conjunto de programas que permiten manipular ese conjunto de datos.

Base de Datos Relacional : tipo de base de datos que cumple con el modelo relacional, almacenando los datos en tablas. Una de las principales características de las bases de datos relacionales es que evitan la duplicidad de registros, y a su vez garantizan la integridad referencial, es decir: si se elimina uno de los registros, la integridad de los registros restantes no será afectada.

Cálculo Relacional : es un lenguaje de consulta que describe la respuesta deseada sobre una Base de datos sin especificar cómo obtenerla. A diferencia del Álgebra relacional - que es de tipo procedimental-, el cálculo relacional es de tipo declarativo, pero ambos métodos logran siempre los mismos resultados.

Clave Principal o **Clave Primaria** : conjunto de atributos que constituyen un identificador único para cada fila dentro de una tabla, permitiendo el acceso rápido a los datos.

Clave Foránea : conjunto de atributos en una tabla que son clave primaria en otra.

Columna, Campo o Atributo : cada uno de los valores únicos (datos) que proporcionan la estructura según la cual se descomponen las filas (registros o tuplas) de una tabla.

Consulta ("Query") : operación para acceder a la información en las bases de datos, que se realiza a través de un lenguaje de manipulación de datos. Mediante las consultas se puede modificar, borrar, mostrar y agregar datos de una base de datos.

Consulta combinada o "Join" : consulta realizada sobre dos o más tablas, y que devuelve como resultado una vista con filas que mezclan datos provenientes de dichas tablas, obtenidos de aquellos registros que cumplan la condición de comparación establecida sobre unos atributos comunes a las tablas. Estos atributos comunes aparecen una sola vez en el resultado.

Dialecto : cada una de las implementaciones de SQL. No hay dos implementaciones idénticas, y ninguna se corresponde exactamente con el estándar ISO.

Esquema conceptual : Descripción de alto nivel del contenido de información de la base de datos, independiente del SGBD que se vaya a utilizar.

Inner Join : tipo de consulta combinada que devuelve únicamente aquellos registros/filas que cumplen la condición de comparación en los dos campos comunes que se comparan para combinar ambas tablas.

Independencia lógica de datos: Capacidad de modificar el esquema conceptual sin provocar que se vuelvan a escribir los programas de aplicación.

Lenguaje declarativo : lenguaje de programación en el que siempre se describe el resultado final deseado, en lugar de detallar los pasos de la solución. Para alcanzar el objetivo, en la programación declarativa se determina automáticamente el mecanismo para obtener la solución. Los Lenguajes Declarativos se caracterizan por tener una sintaxis abreviada y abstracta.

Modelo Relacional : modelo de organización y gestión de bases de datos, basado en la lógica de predicados y en la teoría de conjuntos, y que utiliza un grupo de tablas para representar los datos y las relaciones entre ellos.

Nivel Conceptual de una base de datos : nivel de abstracción intermedio entre el nivel externo y el nivel interno según la arquitectura ANSI de bases de datos, que define el conjunto de entidades y relaciones que componen la base de datos y su información semántica asociada.

Nivel Externo (o esquema lógico) de una base de datos : nivel de abstracción que permite definir las informaciones a las que pueden acceder los usuarios o aplicaciones de una base de datos, y en el que las vistas que se definen vienen en función de las entidades y relaciones definidas en el nivel conceptual de la base de datos.

Outer Join : tipo de consulta combinada que, además de devolver aquellos registros/filas que cumplen la condición de comparación en los dos campos comunes que se comparan para combinar ambas tablas, también devuelve los registros restantes de una u otra tabla (según sea un “Left Outer Join” o un “Right Outer Join”), o bien los de ambas tablas (si se trata de un “Ful Outer Join”).

Producto cartesiano : dadas dos relaciones o tablas “R” y “S”, se define el producto cartesiano como la relación resultante de combinar cada fila de “R” con todas las de “S”, de tal manera que sus atributos corresponden a los de “R” seguidos por los de “S”.

Proyección : operador del algebra relacional que permite extraer columnas (atributos) de una relación, dando como resultado un subconjunto vertical de atributos de la relación.

Selección o Restricción : operador del álgebra relacional, que actúa sobre una relación o tabla “R” y da como resultado otra relación cuyas tuplas son el subconjunto de tuplas de “R” que cumplan la condición especificada (“C”).

Sistemas de Gestión de Base de Datos (SGBD) : conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, y que están dedicados a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

SQL : lenguaje especializado de programación, de tipo declarativo, que se utiliza para recuperar y actualizar la información contenida en una base de datos, y que surgió a partir del modelo relacional. Fue desarrollado en los años 70 por IBM. Se ha convertido en un estándar ISO y ANSI.

Subquery (subconsulta) : consulta ubicada dentro de otra consulta, y que sirve para determinar el contenido del resultado final.

Tabla (denominada “relacion” en el modelo relacional) : estructura de datos que organiza la información en filas o tuplas, y columnas o campos.

Tupla, Registro o Fila : una tupla se define como una función finita que asocia unívocamente los nombres de los atributos o columnas de una relación con los valores de una instanciación de la misma. En términos prácticos, una tupla se corresponde con una fila de una tabla relacional.

Transacción : unidades o secuencias de trabajo realizadas de forma ordenada y separada en una base de datos, y tratadas como una única unidad.

Vista : tabla virtual cuyo contenido está definido por una consulta (la cual puede operar sobre una o varias tablas o vistas).

5 BIBLIOGRAFIA BASICA

- T. Connolly y C. Begg. “Database systems: a practical approach to design, implementation, and management (6th ed.)”. Capítulo 6.
- Elmasri y Navathe. “Fundamentals of Database Systems” (Seventh Edition).
- C. J. Date. “An Introduction to Database Systems” (Eigth Edition).
- A. Silberschatz, H.F. Korth y S.Sudarshan. “Database System Concepts” (Seventh Edition)