**2.3 Lenguaje de Consultas**

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos. Pero como sucede con cualquier sistema de normalización hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no nos servirá para resolver todos los problemas, aunque si se puede asegurar que cualquier sentencia escrita en ANSI será interpretable por cualquier motor de datos.

**2.3.1 Historia del Lenguaje de Consultas**

La historia de SQL (que se pronuncia deletreando en inglés las letras que lo componen, es decir "ese-cu-ele" y no "siquel" como se oye a menudo) empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transformó en estándar ISO. Esta versión del estándar va con el nombre de SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89 y, posteriormente, a la actual SQL/92.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa en la propia base de datos sólo el corazón del lenguaje SQL (el así llamado Entry level o al máximo el Intermediate level), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva encarnación de SQL deberían ser su transformación en un lenguaje stand-alone (mientras ahora se usa como lenguaje hospedado en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales.

**2.3.2 Componentes del SQL**

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

**2.3.2.1 Comandos**

Existen dos tipos de comandos SQL:

* Los DLL que permiten crear y definir nuevas bases de datos, campos e índices.
* Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Entre los comandos DLL tenemos a los siguientes:

**CREATE:** Utilizado para crear nuevas tablas, campos e índices.

**DROP:** Empleado para eliminar tablas e índices.

**ALTER:** Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

A continuación se presentan los comandos DML utilizados para el manejo de consultas:

**SELECT:** Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.

**INSERT:** Utilizado para cargar lotes de datos en la base de datos en una única operación.

**UPDATE:** Utilizado para modificar los valores de los campos y registros especificados.

**DELETE:** Utilizado para eliminar registros de una tabla de una base de datos.

**2.3.2.2 Cláusulas**

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular. Entre las cláusulas utilizadas para el manejo de consultas tenemos:

**FROM:** Utilizada para especificar la tabla de la cual se van a seleccionar los registros.

**WHERE:** Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.

**GROUP BY:** Utilizada para separar los registros seleccionados en grupos específicos.

**HAVING:** Utilizada para expresar la condición que debe satisfacer cada grupo.

**ORDER BY:** Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

**2.3.2.3 Operadores Lógicos**

Los operadores lógicos son utilizados para enlazar condiciones que se expresan en la cláusula **WHERE** o para negar una condición específica. A continuación se presentan a los operadores lógicos para manejo de consultas:

**AND:** Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.

**OR:** Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.

**NOT:** Negación lógica. Devuelve el valor contrario de la expresión.

**2.3.2.4 Operadores de Comparación**

Estos operadores son utilizados para realizar comparaciones entre valores o variables en las condiciones que se expresan en la cláusula **WHERE**. A continuación se presentan los operadores de comparación utilizados en el manejo de consultas:

**>** Mayor que

**<** Menor que

**>=** Mayor ó igual que

**<=** Menor ó igual que

**=** Igual que

**<>** Distinto que

**BETWEEN** Utilizado para especificar un rango de valores

**LIKE** Utilizado en la comparación de un modelo

**IN** Utilizado para especificar registros de una base de datos.

**2.3.2.5 Funciones de Agregado**

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros. A continuación se menciona las funciones de agregado utilizadas para el manejo de consultas:

**AVG:** Utilizada para calcular el promedio de los valores de un campo determinado.

**COUNT:** Utilizada para devolver el número de registros de la selección.

**SUM:** Utilizada para devolver la suma de todos los valores de un campo determinado.

**MAX:** Utilizada para devolver el valor más alto de un campo especificado.

**MIN:** Utilizada para devolver el valor más bajo de un campo especificado.

**2.3.3 Consultas de selección**

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

**2.3.3.1 Estructura básica de las consultas**

Para el manejo de consultas de bases de datos se debe seguir una estructura básica, que es la que especifica a continuación:

SELECT A1,A2,...,An

FROM r1,r2,...,rn

WHERE P

Donde Ai = atributo ( Campo de la tabla )

ri = relación ( Tabla )

P = predicado ( condición )

Por ejemplo, para seleccionar todos los nombres de las personas que tengan el apellido CASTRO de la tabla persona se utiliza una consulta como la siguiente:

SELECT nombre

FROM persona

WHERE apellido = " CASTRO"

Es posible renombrar los atributos y las relaciones, a veces por conveniencia y otras veces por ser necesario, para esto usamos la cláusula **AS**. A continuación se presenta el ejemplo anterior con el uso de esta cláusula.

SELECT P.nombre AS [PRIMER NOMBRE]

FROM persona P

WHERE apellido = "CASTRO"

La complejidad de una consulta puede aumentar cada vez más. Esto depende de los requerimientos que se haga y del uso de los comandos y cláusulas que se necesiten. Por ejemplo, puede haber **subconsultas**, que no son nada más que una consulta dentro de otra; es decir, un SELECT dentro de otro. Puede utilizar tres formas de sintaxis para crear una subconsulta:

comparación [ANY | ALL | SOME] (instrucción sql)

expresión [NOT] IN (instrucción sql)

[NOT] EXISTS (instrucción sql)

En donde:

**comparación**

Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

**expresión**

Es una expresión por la que se busca el conjunto resultante de la subconsulta.

**instrucción sql**

Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING.

Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento:

SELECT \* FROM Productos WHERE PrecioUnidad > ANY

(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento:

SELECT \* FROM Productos WHERE IDProducto IN

(SELECT IDProducto FROM DetallePedido WHERE Descuento >= 0.25);

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:

SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE EXISTS

(SELECT FROM Pedidos WHERE Pedidos.IdPedido = Clientes.IdCliente)

Esta consulta es equivalente a esta otra:

SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE IdClientes IN

(SELECT Pedidos.IdCliente FROM Pedidos)

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título. A la tabla Empleados se le ha dado el alias T1:

SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1

WHERE Salario >= (SELECT Avg(Salario) FROM Empleados

WHERE T1.Titulo = Empleados.Titulo) ORDER BY Titulo;

**2.3.3.2 Consultas de combinación entre tablas**

Las vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es la siguiente:

SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2

En donde:

**tb1, tb2**: Son los nombres de las tablas desde las que se combinan los registros.

**campo1, campo2**: Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

**Comp**: Es cualquier operador de comparación relacional: =, <, >, <=, >=, o <>.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

SELECT Nombre\_Categoría, NombreProducto

FROM Categorias INNER JOIN Productos

ON Categorias.IDCategoria = Productos.IDCategoria;

**2.3.3.3 Consultas de autocombinación**

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis es la siguiente:

SELECT alias1.columna, alias2.columna, ...

FROM tabla1 as alias1, tabla2 as alias2

WHERE alias1.columna = alias2.columna

AND otras condiciones

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

SELECT t.num\_emp, t.nombre, t.puesto, t.num\_sup,s.nombre, s.puesto

FROM empleados AS t, empleados AS s WHERE t.num\_sup = s.num\_emp

**2.3.3.4 Consultas de combinaciones no comunes**

La mayoría de las combinaciones están basadas en la igualdad de valores de las columnas que son el criterio de la combinación. Las no comunes se basan en otros operadores de combinación, tales como NOT, BETWEEN, <>, etc.

Por ejemplo, para listar el grado salarial, nombre, salario y puesto de cada empleado ordenando el resultado por grado y salario habría que ejecutar la siguiente sentencia:

SELECT grados.grado,empleados.nombre, empleados.salario, empleados.puesto

FROM empleados, grados

WHERE empleados.salario

BETWEEN grados.salarioinferior AND grados.salariosuperior

ORDER BY grados.grado, empleados.salario