TEMA:

SQL

UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS ECONOMICO ADMINISTRATIVAS

MAESTRIA EN TECNOLOGÍAS DE LA INFORMACION

MATERIA: BASE DE DATOS

EQUIPO:

ANTONIO C GUILLEN
BARBARA JUDITH BAENA ZUÑIGA

JUNIO, 2003

UNA BREVE HISTORIA DE SQL

A principios de los años setenta, en sus proyectos de investigación, IBM ideó un lenguaje denominado Structured English Query Language (SEQUEL) para construir un sistema de gestión de base de datos relacional. Este lenguaje evolucionó hasta convertirse en SEQUEL/2 y, finalmente, en Structured Query Language (SQL, Lenguaje de consulta estructurado). Además, hubieron otras empresas que se interesaron por el concepto de bases de datos relacionales y el interfaz SQL que había surgido. Relational Software, Inc. (actualmente Oracle Corporation) creó un producto denominado Oracle en el año 1979. En 1981, IBM lanzó su primer producto, denominado SQL Data System (SQL/DS).

En 1982, el American National Standards Institute (ANSI), dándose cuenta del significado potencial del modelo relaciona!, comenzó a trabajar en un estándar denominado Relational Database Language (RDL, Lenguaje de bases de datos relacionales). La aceptación surgida en 1984 en el mercado de productos tales como Oracle, SQL/DS y DB2 de IBM hizo que el comité ANSI pusiese su atención en SQL como la base para el nuevo estándar RDL. La primera versión de este estándar, SQL-86, fue adoptado tanto por ANSI como por el International Standards Organization (ISO) en octubre de 1986. En 1989, se adoptó una actualización de SQL-86 que abarcó las mejoras de la integridad. El estándar actual, al que en numerosas ocasiones se hace referencia como «SQL2» o «SQL-92», refleja el esfuerzo intenso llevado a cabo por las personas de los estándares internacionales para mejorar el lenguaje y corregir muchas características ambiguas, confusas y perdidas del estándar original de 1986.

El estándar existente actualmente representa tanto un subconjunto de las principales implementaciones comunes como un súper conjunto de casi todas las implementaciones. Es decir, el núcleo del estándar consta de características que podemos encontrar virtualmente en cada implementación comercial del lenguaje, aunque el estándar completo incluye características mejoradas que-muchos vendedores ya han implementado.

Como ya mencionamos en el capítulo anterior, el consorcio del SQL Access Group de vendedores de base de datos ha publicado lo que podría ser considerado como el «estándar comercial» para SQL; una variante del lenguaje que puede ser «hablado» por (o proyectado hacia) cada producto de base de datos relacional principal. Una versión extendida de este «Interfaz de Lenguaje Común» (CLI) forma parte del estándar SQL3 en 1996.

El lenguaje de base de Datos SQL :

En el presente documento nos concentraremos en la interfaz de consulta genérica del lenguaje. Es decir, lo veremos con un lenguaje autónomo de consultas, destacaremos las características que ofrecen casi todos los sistemas comerciales y también el estándar anterior de ANSI. Trataremos de dar al lector una idea general de la naturaleza de SQL, nos concentraremos en las características de mayoruso.

SELECT

La sentencia SELECT forma el núcleo del lenguaje de base de datos SQL. Utilizaremos la sentencia SELECT para seleccionar o recuperar las filas y columnas deseadas a partir de las tablas de nuestra base de datos. La sintaxis de la sentencia SELECT consta de cinco cláusulas principales, construidas normalmente de la siguiente manera:

SELECT *<lista de campos>*
FROM *<lista de tablas>*
[WHERE *<especificación de selección de fila>]*
[GROUP BY *<especificación de agrupación>]*
[HAVING *<especificación de selección de grupo>]*
[ORDER BY *<especificación de ordenación>*];

FROM, cláusula

Especifica las tablas o consultas que proporcionan los datos de origen para su consulta.

Sintaxis:

FROM *{{nombre-de-tabla* [[AS] *nombre-de-correlación*] |
nombre-de-consulta-de-selección [[AS] *nombre-de-correlación]* |
<tabla combinada>} ,...
(IN *<especificación-dei-origen>*]

Notas:

De manera opcional, se puede suministrar un nombre de correlación para cada nombre de tabla o consulta y utilizar este nombre de correlación como un alias para el nombre de tabla completo a la hora de especificar los nombres de columna en la lista de selección, en la *<lista-de-campo>*, en la *<especificación-de-combinación>* o en la cláusula WHERE y las subcláusulas. Si está combinando

una tabla o una consulta consigo misma, deberá utilizar los nombres de correlación para especificar a qué copia de la tabla o consulta está haciendo referencia en la lista de selección, criterio de combinación o criterio de selección. Si el nombre de la tabla o consulta es además una palabra reservada (por ejemplo, «Order»), deberá encerrar el nombre entre paréntesis.

Si incluye varias tablas en la cláusula FROM sin una especificación JOIN pero incluye un predicado que compara campos de varias tablas en la cláusula WHERE, en la mayoría de los casos, se optimiza esta situación llegando a la solución de que la consulta sea tratada como un JOIN. Por ejemplo:

```
SELECT *
FROM TablaA, TablaB
WHERE TablaA.ID = TablaB.ID
```

es tratado como si se hubiese especificado

```
SELECT *
FROM TablaA
    INNER JOIN TablaB
    ON TablaA.ID = TablaB.ID
```

No se pueden actualizar los campos de una tabla utilizando un conjunto de registros mostrado por la consulta, la hoja de datos de la consulta o un formulario basado en una consulta de varias tablas donde la combinación haya sido expresada utilizando una lista-de-tabla y una cláusula WHERE. En la mayoría de las ocasiones, podrá actualizar los campos de las tablas subyacentes cuando utilice la sintaxis JOIN.

Ejemplo:

Obtener la fecha de nacimiento y la direccion del empleado cuyo nombre es ‘José B. Silva’.

```
SELECT      FECHAAN, DIRECCION
FROM        EMPLEADO
WHERE       NOMBREP=’Silva’ AND INIC= ‘B’ AND APELLIDO = ‘Silva’
```

GROUP BY, cláusula

En una sentencia SELECT, esta cláusula especifica las columnas utilizadas para la formación de grupos a partir de las filas seleccionadas. Cada grupo contiene valores idénticos en la columna o columnas especificadas. La cláusula GROUP BY se utiliza para definir una consulta de totales. También, deberá incluir una cláusula GROUP BY en una consulta de tabla de referencias cruzadas.

```
SELECT tblCategorias.Categoría, Avg(tblLibros.PrecioSug) AS PromDePrecioSug,
    Max(tblLibros.PrecioSug) AS MaxDePrecioSug
FROM tblLibros
INNER JOIN (tblCategorías
        INNER JOIN tblCategoríasLibro
        ON (tblLibros.NúmeroISBN =
          tblCategoríasLibro.NúmeroISBN)
          AND (tblLibros.NúmeroISBN =
              tblCategoríasLibro.NúmeroISBN))
        ON tblCategorías.IDCategoría =
          tblCategoríasLibro.IDCategoría
```

```
GROUP BY tblCategorias .Categoría;
```

HAVING, cláusula

Especifica los grupos de filas que aparecen en la tabla lógica (un conjunto de registros de Access) definido por una sentencia SELECT. La condición de búsqueda se aplica a las columnas especificadas en una cláusula GROUP BY, a las columnas creadas por funciones de totales o a las expresiones que contienen funciones de totales. Si un grupo no cumple la condición de búsqueda, no será incluido en la tabla lógica.

Sintaxis:

HAVING *condición-de-búsqueda*

Notas:

Si no incluye una cláusula GROUP BY, la lista de selección deberá estar formada mediante el uso de una o más funciones de totales.

La diferencia entre la cláusula HAVING y la cláusula WHERE es que la condición de búsqueda de la cláusula WHERE se aplica a las filas individuales antes de que sean agrupadas, mientras que la condición de búsqueda de la cláusula HA VING se aplica a los grupos de filas.

```
SELECT Month([FechaPedido]) AS Mes,
        Avg(cEjempTotalesPedido.TotalPedido)
          AS PromDeTotalPedido,
        Max(cEjempTotalesPedido.TotalPedido)
          AS MaxDeTotalPedido
FROM cEjempTotalesPedido
WHERE (((cEjempTotalesPedido.EdoOProv)=«WA»))
GROUP BY Month([FechaPedido])
HAVING (( (Max(cEjempTotalesPedido.TotalPedido))<1900));
```

JOIN, operación

Gran parte de la potencia de SQL viene dada por su habilidad para combinar (unir) información procedente de varias tablas o consultas y mostrar el resultado como un único conjunto de registros. En la mayoría de los casos, SQL permite actualizar el conjunto de registros de una tabla combinada como si se tratase de una simple tabla base.

Utilice una operación JOIN en una cláusula FROM para especificar la forma en que desea que se enlacen dos tablas al formar un conjunto de registros lógico a partir del cual se seleccionará la información necesaria. Puede hacer que se combinen solamente aquellas filas que coinciden en ambas tablas (denominado *combinación interna)* o que devuelva todas las filas de una de las dos tablas incluso cuando una fila coincidente no exista en la segunda tabla (denominado *combinación externa).* Puede anidar varias operaciones de combinación para enlazar, por ejemplo, una tercera tabla con el resultado de la combinación de las otras dos tablas.

Notas:

De manera opcional, puede suministrar un nombre de correlación para cada nombre de tabla o consulta. Este nombre de correlación se puede utilizar como un alias para el nombre de tabla completo durante la especificación de los nombres de columna en la lista de selección o en la cláusula WHERE o subcláusulas. Si se encuentra combinando una tabla o una consulta consigo misma, deberá utilizar los nombres de correlación para especificar a qué copia de la tabla o consulta está haciendo referencia en la lista de selección, criterio de combinación o criterio de selección. Si el nombre de la tabla o consulta es además una palabra reservada (por ejemplo, «Orden», deberá encerrar el nombre entre paréntesis.

Utilice INNER JOIN para obtener como resultado todas las filas que coinciden con la especificación de combinación en ambas tablas. Utilice LEFT JOIN para obtener como resultado todas las filas procedentes de la primera tabla lógica (donde *tabla lógica* es cualquier expresión de tabla, consulta o tabla combinada) unida en la especificación de combinación con cualquiera de las filas que coincidan con la segunda tabla lógica.

Cuando no existe ninguna fila coincidente en la segunda tabla, Access da por resultado valores nulos en las columnas de esa tabla. En sentido inverso, al utilizar RIGHT JOIN obtendremos todas las filas procedentes de la segunda tabla lógica combinada con cualquiera de las filas que coincidan con la primera tabla lógica. Cuando solamente utilice predicados de comparación de *equivalencia* en la especificación de combinación, el resultado se denomina una *combinación equivalente.*

```
SELECT DISTINCTROW tblLibros.NúmeroISBN
FROM tblLibros
        INNER JOIN (tblPedidos
                INNER JOIN tblDetallesPedido
                ON tblPedidos.IDPedido =
                tblDetallesPedido.IDPedido)
        ON tblLibros.NúmeroISBN =
                tblDetallesPedido.NúmeroISBN
        WHERE (((tblPedidos.FechaPedido) Between Date ( ) And Date()-90));
```

ORDER BY, cláusula

Especifica la secuencia de filas a ser devuelta por una sentencia SELECT o una sentencia INSERT.

Sintaxis:

ORDER BY *{nombre-de-columna* [ASC | DESC] } , . . .

Notas:

Utilice los nombres de columna o los números de columna de salida relativos para especificar las columnas cuyos valores serán utilizados para ordenar las filas devueltas. (Si utiliza los números de columna de salida relativos, la primera columna de salida es la 1.) En una cláusula ORDER BY se pueden especificar varias columnas. La lista será ordenada primeramente por la columna cuyo

nombre se especifica en primer lugar. Si existen filas que poseen valores idénticos en esa columna, a continuación serán ordenados por el nombre de la siguiente columna de la lista ORDER BY. Se puede especificar el orden deseado, ascendente (ASC) o descendente (DESC), para cada columna. Si no indica ASC o DESC, se asume el orden ascendente, ASC. El uso de una cláusula ORDER BY en una sentencia SELECT es lo único que determina la definición de la secuencia de las filas a devolver.

Ejemplos:

Para seleccionar los clientes con los que negociamos por primera vez en Agosto de 1996 e incluso antes, y listados en orden ascendente por código postal, introduzca lo siguiente:

```
SELECT tblClientes.Nombre, tblClientes.SegundoNombre, tblClientes.Apellidos,
        tblClientes.Ciudad, tblClientes.CódigoPostal
FROM tblClientes
WHERE (((#1/9/96#)>=
        (Select Min([FechaPedido]) FrOID tblPedidos Where tblPedidos.IDCliente =
tblClientes.IDCliente)))
ORDER BY tblClientes. Códigopostal;
```

Predicado BETWEEN

Compara un valor con un rango de valores.

Sintaxis:

expresión [NOT] BETWEEN *expresión* AND *expresión*

Notas:

El tipo de datos de todas las expresiones debe ser compatible. La comparación de literales (cadenas) alfanuméricos en Access no hace distinción entre las mayúsculas y minúsculas.

Considere las expresiones *a, b,* y c. Podremos decir, en términos de otros predicados, que *a* BETWEEN *b* AND c es equivalente a la siguiente expresión:

a >= b AND a <= e

a NOT BETWEEN *b* AND c es equivalente a:

a < b OR a > e

El resultado es indefinido si cualquiera de las expresiones es Nula.

Ejemplo:

Para determinar si el promedio de Cantidad multiplicado por PrecioSug es mayor o igual a 500 dólares y menor o igual que 10000 dólares, introduzca lo siguiente:

AVG (Cantidad * precioSug) BETWEEN 500 AND 10000

Predicado IN

Determina si un valor es igual a alguno de los valores o es distinto a todos los valores de un conjunto devuelto por una subconsulta o proporcionado en una lista de valores.

Sintaxis:

expresión [NOT] IN { *(subconsulta)* | *({literal}* , . . .) | expresión}

Notas:

La comparación de cadenas en SQL no hace distinción entre mayúsculas y minúsculas. El tipo de datos de todas las expresiones, literales y la columna obtenida como resultado de la subconsulta deben ser compatibles. Si la expresión o cualquier valor devuelto por la subconsulta es Nulo, el resultado será indefinido. En términos de otros predicados, *expresión* IN *expresión* sería equivalente a:

expresión - expresión

expresión IN *(subconsulta)* sería equivalente a:

expresión = ANY *(subconsulta)*

expresión IN *(a, b,* c, ...), donde *a, b* y c son literales, sería equivalente a:

(expresión = a) OR *(expresión = b)* OR *(expresión* = e)

expresión NOT IN ... es equivalente a:

NOT *(expresión* IN . . .)

Ejemplos:

Para determinar si un Estado norteamericano pertenece a la costa oeste de Estados Unidos, introduzca lo siguiente:

```
[EdoOProv] IN ("CA", "OR", "WA")
```

Para determinar si el IDCliente coincide con cualquier IDLibrería de Washington, introduzca las siguientes sentencias:

```
IDCliente IN
   (SELECT IDLibrería
     FROM tblLibrerías
     WHERE tblLibrerías. [EdoOProv] = "WA")
```

Predicado LlKE

Localiza las cadenas que coinciden con un determinado modelo

Sintaxis:

nombre-columna [NOT] LIKE *cadena-coincidente*

Notas:

Las comparaciones de cadena en SQL no hacen distinción entre mayúsculas y minúsculas. Si la columna especificada por *nombre-columna* contiene un valor Nulo, el resultado es indefinido. La comparación de dos cadenas vacías o una cadena vacía con el carácter especial asterisco (*) se evalúa a Verdadero.

Proporcione una cadena de texto como el valor para *cadena-coincidente* que defina los caracteres que pueden existir y en qué posiciones deben encontrarse para que la comparación sea cierta. SQL comprende un cierto número de caracteres comodín que pueden ser utilizados para definir las posiciones que pueden contener cualquier carácter, ningún carácter o más de un carácter, o un único número cualquiera. Estos caracteres son los siguientes:

? Un único carácter cualquiera.
* Cero o más caracteres.
# Un único número cualquiera.

Ejemplos:

Para determinar si el campo Apellidos del cliente tiene al menos una longitud de cuatro caracteres y si además comienza por *Smi,* introduzca lo siguiente:

tblClientes.Apellidos LIKE "Smi?*"

Para comprobar si CódigoPostal es un código postal canadiense válido, introduzca

CódigoPostal LIKE "[A-Z] # [A-Z] # [A-Z] #"

Predicado NULL

Determina si la expresión se evalúa a Nulo. Este predicado se evalúa solamente a verdadero o falso y no será evaluado a indefinido.

Sintaxis:

expresión IS [NOT] NULL

Ejemplo:

Para determinar si la columna del número de teléfono del cliente nunca ha sido rellenada,

introduzca lo siguiente:

NumeroTelefono IS NULL

CONSULTAS ANIDADAS Y COMPARACIONES DE CONJUNTOS:

En algunas consultas es preciso obtener valores existentes en la base de datos para usarlo en una condición de comparación. Una forma cómoda de formular tales consultas es mediante consultas anidadas, que son consultas SELECT completas dentro de la cláusula WHERE de otra consulta, la cual se denomina consulta exterior. La siguiente consulta es un ejemplo:

```
SELECT    DISTINCT NÚMEROP
FROM      PROYECTO
WHERE     NÚMEROP IN  (SELECT NÚMEROP
          FROM PROYECTO, DEPARTAMENTO, EMPLEADO
          WHERE NÚMD=NÚMEROD AND NSSGTE=NSS AND APELLlDO = 'Silva')
               OR NÚMEROP IN     (SELECT NÚMP
                                  FROM TRABAJA_EN, EMPLEADO
                                  WHERE NSSE=NSS AND APELLlDO='Silva')
```

La primera consulta anidada selecciona los números de los proyectos en que un 'Silva' participa como gerente, y la segunda selecciona los números de los proyectos en que un 'Silva' participa como trabajador. En la consulta exterior, seleccionamos una tupla PROYECTO si el valor de NÚMEROP de esa tupla está en el resultado de cualquiera de las dos consultas anidadas.
El operador de comparación IN compara un valor v con un conjunto (o multiconjunto) de valores V y produce el valor TRUE si v es uno de los elementos de V.
El operador IN también puede comparar una tupla de valores entre paréntesis con un conjunto de tuplas compatibles con la unión. Por ejemplo, la consulta

```
SELECT DISTINCT NSSE
FROM TRABAJA_EN
WHERE (NÚMP, HORAS) IN (SELECT NÚMP, HORAS FROM TRABAJA_EN
                        WHERE NSSE='123456789');
```

seleccionará los números de seguro social de todos los empleados que trabajan para la misma combinación (horas, proyecto) en algún proyecto en el que trabaja el empleado 'José Silva' (cuyo NSS = '123456789'). Además del operador IN, podemos usar varios otros operadores de comparación para comparar un valor individual v (por lo regular un nombre de atributo) con un conjunto V (por lo regular una consulta anidada). El operador = ANY (o = SOME) devuelve TRUE si el valorv es igual a algún valor del conjunto V, y por tanto equivale a IN. Las palabras reservadas ANY y SOME tienen el mismo significado. Otros operadores que se pueden combinar con ANY (o SOME) incluyen>, > =, <, < = y < >. También es posible combinar la palabra reservada ALL (todas) con uno de estos operadores. Por ejemplo, la condición de comparación (v > ALL V) devuelve TRUE si el valor ves mayor que todos los valores del conjunto V. Un ejemplo es la siguiente consulta, que devuelve los nombres de los empleados cuyo salario es mayor que el de todos los empleados del departamento 5:

```
SELECT APELLIDO, NOMBREP
```

```
FROM EMPLEADO
WHERE SALARIO> ALL (SELECT SALARIO FROM EMPLEADO WHERE ND=5);
```

En general, podemos tener varios niveles de consultas anidadas. Una vez más, nos enfrentamos a posibles ambigüedades en los nombres de los atributos si hay atributos con el mismo nombre: uno en una relación listada en la cláusula FROM de la consulta exterior y el otro en una relación listada en la cláusula FROM de la consulta anidada. La regla es que una referencia a un atributo no calificado se refiere a la relación declarada en la consulta anidada más interior.
Por ejemplo, en la cláusula SELECT y en la cláusula WHERE de la primera consulta anidada, una referencia a cualquier atributo no calificado de la relación PROYECTO se refiere a la relación PROYECTO especificada en la cláusula FROM de la consulta anidada. Si queremos referimos a un atributo de la relación PROYECTO especificada en la consulta exterior, podemos especificar un seudónimo de esa relación y referimos a él. Estas reglas son similares a las reglas de alcance (validez) para las variables de programa en un lenguaje como PASCAL, que permite anidar procedimientos y funciones. Para ilustrar la ambigüedad potencial de los nombres de atributos en las consultas anidadas, consideremos la consulta siguiente.

Obtener el nombre de todos los empleados que tienen un dependiente con el mismo nombre de pila y sexo que el empleado.

```
SELECT E.NOMBREP, E.APELLlDO
FROM EMPLEADO AS E
WHERE E.NSS IN
(SELECT NSSE
FROM DEPENDIENTE
WHERE NSSE=E.NSS AND E.NOMBREP= NOMBRE_DEPENDIENTE AND SEXO=E.SEXO)
```

En la consulta anidada, debemos calificar E.SEXO porque se refiere al atributo SEXO del EMPLEADO de la consulta exterior, y DEPENDIENTE también tiene un atributo llamado SEXO. Todas las referencias no calificadas a SEXO en la consulta anidada se refieren a SEXO de DEPENDIENTE. Sin embargo, no tenemos que calificar NOMBREP ni NSS porque la relación DEPENDIENTE no tiene atributos llamados NOMBREP ni NSS, así que no hay ambigüedad. Obsérvese que es necesaria la condición NSS = NSSE en la cláusula WHERE de la consulta anidada; sin esta condición, seleccionaríamos los empleados cuyo nombre de pila y sexo coincidieran con los de cualquier dependiente, sea que dependiera de ese empleado en particular o no.
Siempre que una condición en la cláusula WHERE de una consulta anidada hace referencia a un atributo de una relación declarada en la consulta exterior, se dice que las dos consultas están correlacionadas. Podemos entender mejor qué es una consulta correlacionada si consideramos que la consulta anidada se evalúa una sola vez para cada tupla (o combinación de tuplas) en la co sulta exterior.

INSERT INTO

Crea tuplas en una relación.

Sintaxis:

INSERT INTO <nombre_realci(A_1,...An) VALUES (v_1,...vn)

Ejemplo:

INSERT INTO StarsIn (movieTitle, movieYear, starName)
VALUES (‘The Maltese Falcon’, 1942, ‘Sydney Greensreet’);

DELETE FROM

Esta proposición hace que todas las tuplas que cumplan con la condición (WHERE) sean eliminadas en la relación R.

Sintaxis:

DELETE FROM *R* WHERE <condicion>;

Ejemplo:

DELETE FROM MovieExec
WHERE netWorth < 10000000;

UPDATE

Esta proposición produce el efecto de encontrar todas las tuplas de *R* que cumplan la condición (WHERE). Se les modifica al hacer que las fórmulas sean evaluadas y asignadas a los componentes de la tupla para los atributos correspondientes de *R*.

Sintaxis:

UPDATE SET <nuevos-valores>WHERE <condicion>;

Ejemplo:

UPDATE MovieExec
SET name = ‘Pres.’ | | name
WHERE cert# IN (SELECT presC# FROM Studio);

CREATE TABLE

La forma más simple de una declaración del esquema relacional consta de las palabras clave CREATE TABLE seguidas del nombre de la relación y de una lista entre paréntesis de los nombres de los atributos y sus tipos.

Sintaxis:

CREATE TABLE <nombre_tabla> (
<Campo1> CHAR(n)

<Campo2> VARCHAR(n)
<Campo3> DATE) ;

Ejemplo:

CREATE TABLE MovieStar(
name CHAR (30),
address VARCHAR (255),
gender CHAR(1),
birthdate DATE);

DROP

Una relación podría ser temporal, quizá como resultado intermedio de alguna consulta que no puede expresarse como una proposición individual de SQL. De ser así, podemos eliminar una relación *R* mediante DROP.

Sintaxis:

DROP, <*nombre tabla*>

Ejemplo:

DROP, tablaTemp

Modificación de los esquemas relacionales

Las modificaciones se realizan mediante una proposición que comienza con las palabras clave ALTER TABLE y el nombrede la relación. Disponemos de varias opciones, siendo las más importantes:

1. ADD seguida de un nombre de columna y su tipo de datos

2. DROP seguida de un nombre de columna.

Ejemplos

ALTER TABLE MovieStar ADD pone CHAR(16);

ALTER TABLE MovieStar DROP birthdate;

NULL / DEFAULT

Cuando creamos o modificamos tuplas, a veces no tenemos valores para todos los components, entonces se utiliza el valor NULL. Este se convierte en el valor de cualquier componente al que no se le asigne uno específico. Pero hay ocasiones en que preferiremos recurrir a otra opción

por omisión, o sea el que se en un componente cuando no se conoce ningún otro valor entonces se utiliza DEFAULT.

Ejemplos:

1. campo CHAR(1) DEFAULT '?'
2. birthdate DATE DEFAULT DATE '0000-00-00'

CREATE DOMAIN

Hemos definido un tipo de datos para cada atributo. Una alternativa consiste en definir primero un *dominio*, que es un nuevo nombre para un tipo de datos. Varios atributos pueden usar el mismo dominio, integrándaonos de un modo útil.

Sintaxis:

CREATE DOMAIN <nombre> AS <descripción_del_tipo>;

Ejemplo:

CREATE DOMAIN MovieDomain AS VARCHAR (50) DEFAULT 'unknown';

CREATE INDEX

Un índice de un atributo A de una relación es una estructura de datos que permite encontrar rápidamente las tuplas que poseen un valor fijo en ese atributo. Los índices generalmente facilitan las consultas en las que el atributo A se compara con una constante, A= 3 por ejemplo, o hasta A<3.

Sintaxis:

CREATE INDEX <campo1_in> ON <nombre_relación> (campo1)

Ejemplo:

CREATE INDEX YearIndex ON Movie (year)

A menudo se cuenta con un índice multiatributos. Este adopta valores para varios atributos y localiza eficientemente las tuplas con esos valores para ellos.

Ejemplo:

CREATE INDEX KeyIndex ON Movie (title, year);

Si queremos eliminar el índice, usamos:

DROP INDEX YearIndex;

Temas avanzados de SQL sugeridos al lector:

Restricciones de llaves: Podemos delcarar un atributo o un conjunto de atributos como llave con una declaración UNIQUE o PRIMARY KEY en un esquema relacional.

Restricciones de Integridad Referencial: Podemos declarar que un valor que aparece en algún atributo o conjunto de atributos también ha de hacerlo en los atributos llave de alguna tupla de otra relación, usando para ello una declaración REFERENCES o FOREIGN KEY en un esquema relacional.

Restricciones de Verificación Basadas en Atributos: Podemos comprobar una restricción sobre el valor de un atributo incorporando la palabra clave CHECK y la condición a verificar después de la declaración del atributo en su esquema relacional. Otra opción consiste en servirnos de un dominio para el tipo de este atributo y especificar en la declaración del dominio la condición que debe verificarse.

Restricciones de Verificación Basadas en Tuplas: Podemos comprobar una condición concerniente a uno o todos los componentes de las tuplas de una relación, agregándole a la declaración de la relación la palabra clave CHECK y la condición a verificar.

Aserciones: Podemos declarar una aserción como elemento de un esquema de base de datos por medio de la palabra clave CHECK y la condición a verificar. Esta última puede incluir una o más relaciones del esquema de la base de datos y a la relación completa; por ejemplo, con agregación, así como también las condiciones referentes a las tuplas individuales.

Invocación de las verificaciones: Las aserciones se comprueban siempre que un cambio de una de las relaciones hace posible la violación de la restricción. Las verificaciones basadas en valores y en tuplas sólo se revisan cuando el atributo o la relación a que se aplican cambian por la inserción o por la actualización de tuplas. Así pues, estas restricciones pueden violarse sin tienen subconsultas que contienen otras relaciones o tuplas de la misma relación.

Disparadores de SQL3: El estándar propuesto de SQL3 incluye disparadores que especifican ciertos eventos (inserción, eliminación o actualización de una relación) que los ponen en acción. Una vez activados, una condición, puede ser comprobada y, de ser cierta, se ejecutará una secuencia específica de acciones (proposiciones en SQL como consultas y modificaciones de la base de datos).

Aserciones en SQL3: El estándar de este lenguaje incluye un concepto de aserción distinto al de la aserción en SQL2. A semejanza de los disparadores de SQL3, estas aserciones son activadas por uno o más eventos, como la inserción dentro de una relación. Al ser invocadas, las aserciones en SQL3 verifican una condición acerca de las relaciones o las tuplas, rechazando la modificación en caso de que no se cumpla la condición.

SQL incrustrado: En vez e utilizar una interfaz genérica de consultas para expresar las consultas y las modificaciones en este lenguaje, a menudo se logra una mayor eficiencia escribiendo programas que inserten las consultas en un lenguaje anfitrión ordinario.

Desigualdad de impedancia: El modelo de datos de SQL es muy distinto a los modelos de los lenguajes anfitriones ordinarios. Así, se intercambia información entre el SQL y el lenguaje anfitrión mediante variables compartidas capaces de representar los componentes de las tuplas en la parte del programa correspondiente al SQL.

Cursores: Un cursor es una variable de SQL que indica una de las tuplas de una relación. La conexión entre el lenguaje anfitrión y el SQL se facilita al hacer que el cursor recorra cada tupla de la relación, mientras que los componentes de la tupla actual se recuperan e introducen en las variables compartidas y se procesan usando el lenguaje anfitrión.

SQl dinámico: En vez de incrustrar determinadas proposiciones de SQL en el programa del lenguaje anfitrión, éste puede crear cadenas de caracteres que el sistema interpreta como proposiciones suyas y las ejecuta.

Control de concurrencia: El lenguaje SQL2 ofrece un par de mecanismos para evitar que las operaciones concurrentes interfieran entre sí: las transacciones y las restricciones impuestas a los cursores. Estas últimas incluyen la capacidad de declarar “insensible” un cursor; si así se declara, el cursor no verá cambio alguno en su relación.

Transacciones: El lenguaje SQL permite al programador agrupar las proposiciones en transacciones, pueden comprometerse o deshacerse (abortarse). En el segundo caso, se cancelan los cambios hechos por la transacción.

Niveles de aislamiento: El lenguaje SQL2 permite ejecutar las transacciones con cuatro niveles de aislamiento, desde los más rigurosos hasta los más laxos: el nivel ”serializable” (ha de parecer que la transacción se ejecuta completamente antes o después de cada una de las otras), la “lectura repetible” (las tuplas leídas en respuesta a una consulta aparecerán si se repite la consulta), “lectura de comprometiod permanente” (esta transacción no puede ver sino sólo las tuplas escritas por transacciones que ya fueron instaladas) y “lectura no instalada” (ninguna restricción de lo que puede ver una transacción).

Cursores y transacciones de lectura solamente: Podemos declarar que un cursor o una transacción son de lectura solamente. Con esta declaración garantizamos que ninguno de los dos cambiará la base de datos, con lo cual se indica al sistema SQL que no afectará a otras transacciones ni a los cursores en una forma que viole la insensibilidad, la seriabilidad u otros requeriemientos.

Organización de una base de datos: Una instalación con un sistema de administración de base de datos de SQL2 crea un ambiente de SQL. Dentro de él, los elementos de la base –las relaciones, por ejemplo- se agrupan en esquemas (de bases de datos), en catálogos y en conglomerados. Un catálogo es una colección de esquemas,y un conglomerado es la colección más grande de los elementos que el usuario puede ver.

Sistemas de cliente/servidor: Un cliente de SQL se conecta a un servidor, originando así una conexión (liga entre los dos procesos) y una sesión (serie de operaciones). El código ejecutado durante la sesión proviene de un módulo, y la ejecución de este último recibe el nombre de agente de SQL.

Privilegios: Por razones de seguridad, los sistemas SQL2 permiten obtener muchas clases de privilegios sobre los elementos de la base de datos. Entre ellos se encuentran: el derecho a seleccionar (leer), insertar, eliminar o actualizar relaciones y el derecho de referenciar relaciones (referirse a ellas en una restricción). Los privilegios de insertar, actualizar y referenciar también pueden conseguirse en determinadas columnas de una relación.

Diagramas de concesiones: Los propietarios pueden otrorgar privilegios a otros usuarios o al usuario general PUBLIC. Si los privilegios se otorgan con la opción de concesión, entonces pueden ser transmitidos a otros. Los privilegios también pueden revocarse. El diagrama de concesiones es un medio muy útil de recordar lo suficiente sobre la historia de las concesiones y revocaciones, a fin de llevar un control sobre los privilegios dados a los usuarios y sobre su origen.

BIBLIOGRAFIA

Título: Elmasri, R. /Navathe S (1997)
Autores: Sistemas de Bases de Datos
Editorial: Addison Wesley Iberoamericana
Pags:

Título: Introducción a los sistemas de Bases de Datos (1997)
Autores: Jeffrey D. Ullman/Jennifer Widow
Editorial: Prentice Hall
Pags: 279-410.