Técnico en

DESARROLLO DE SOFTWARE

Bases de Datos II

Bases de Datos II

Unidad III

Índices / Funciones

1. Índices

Key/Index

La definición de índices puede hacerse también en el momento de creación de la tabla, mediante la palabra clave key (o index), a la que deberemos proporcionar el nombre que vamos a asignar a esta clave y las columnas que la forman, entre paréntesis. Existen modificadores opcionales sobre el índice que nos permiten especificar si se trata de un índice único o múltiple (según puedan existir o no varios valores iguales del índice en la tabla). En versiones recientes de MySQL existen otros tipos de índices (espaciales, de texto completo, etc.) para tipos de datos concretos y que ofrecen prestaciones adicionales.

Indexación

La indexación es la principal herramienta para optimizar el rendimiento general de cualquier base de datos. Es también la más conocida por los usuarios de servidores MySQL y, paradójicamente, su no utilización es una de las principales causas de bajo rendimiento en servidores de bases de datos.

Muchos administradores y diseñadores simplemente parecen olvidar usar índices para optimizar los accesos a las bases de datos. Por otro lado, algunas personas tienden a

indexar todo, esperando que de esta manera el servidor acelere cualquier tipo de consulta que se le solicite. En realidad, esta práctica puede causar una disminución en el rendimiento, sobre todo en lo que respecta a inserciones y modificaciones.

Para ver las ventajas de utilizar índices, analizaremos en primer término una simple búsqueda en una tabla sin índice alguno:

- El constante acceso de escritura de una tabla la mantiene desordenada.
- La ordenación de una tabla es una operación costosa: el servidor tendría que detenerse un tiempo considerable para ordenar sus tablas.
- Muchas tablas tienen más de un criterio de ordenación: ordenar según una columna implica desordenar otra.
- La inserción y eliminación de datos sin alterar el orden en una tabla es costosa: la inserción de un registro en una tabla grande implicaría una larga espera en la actualización de la misma.
- Si se opta por mantener la tabla desordenada (que es la opción más viable), una búsqueda implicaría forzosamente un recorrido secuencial (también denominado full scan), registro por registro.

El uso de índices en la ordenación de las bases de datos ofrece las ventajas siguientes:

- Permite ordenar las tablas por varios criterios simultáneamente.
- Es menos costoso ordenar un archivo índice, porque incluye sólo referencias a la información y no la información en sí.
- El coste de inserción y eliminación es menor.

- Con los registros siempre ordenados se utilizarán algoritmos mucho más eficientes que el simple recorrido secuencial en las consultas.

El uso de índices también comporta alguna desventaja:

- Los índices ocupan espacio en disco.
- Aun teniendo registros pequeños, el mantener en orden un índice disminuye la velocidad de las operaciones de escritura sobre la tabla.
- A pesar de estos inconvenientes, la utilización de índices ofrece mayores ventajas que desventajas, sobre todo en la consulta de múltiples tablas, y el aumento de rendimiento es mayor cuanto mayor es la tabla.

Consideremos por ejemplo una consulta sobre las tablas A, B, y C, independientemente del contenido de la cláusula where, las tres tablas se deben de combinar para hacer posible posteriormente el filtrado según las condiciones dadas:

```
select *
from A,B,C
where A.a = B.b
and B.b = C.c;
```

Consideremos que no son tablas grandes, que no sobrepasan los 1.000 registros. Si A tiene 500 registros, B tiene 600 y C 700, la tabla resultante de la consulta anterior tendrá 210 millones de registros. MySQL haría el producto cartesiano de las tres tablas y, posteriormente, se recorrería la relación resultante para buscar los registros que

satisfacen las condiciones dadas, aunque al final el resultado incluya solamente 1.000 registros.

Si utilizamos índices MySQL los utilizaría de una forma parecida a la siguiente:

Tomaría cada uno de los registros de A.

Por cada registro de A, buscaría los registros en B que cumpliesen con la condición A.a = B.b. Como B está indexado por el atributo 'b', no necesitaría hacer el recorrido de todos los registros, simplemente accedería directamente al registro que cumpliese la condición.

Por cada registro de A y B encontrado en el paso anterior, buscaría los registros de C que cumpliesen la condición B.b = C.c. Es el mismo caso que en el paso anterior.

Comparando las dos alternativas de búsqueda, la segunda ocuparía cerca del 0,000005% del tiempo original. Por supuesto que sólo se trata de una aproximación teórica, pero adecuada para comprender el efecto de los índices en las consultas sobre bases de datos.

2. Funciones

Una función en MySQL es una rutina creada para tomar unos parámetros, procesarlos y retornar en una salida.

Se diferencian de los procedimientos en las siguientes características:

- Solamente pueden tener parámetros de entrada IN y no parámetros de salida OUT o INOUT
- Deben retornar en un valor con algún tipo de dato definido
- Pueden usarse en el contexto de una sentencia SQL
- Solo retornan un valor individual, no un conjunto de registros.

Como crear una función

```
CREATE FUNCTION nombre_función (parametro1,parametro2,...)
RETURNS tipoDato
[atributos de la rutina]
<bloque de instruccciones>
```

La única diferencia entre la creación de un procedimiento y una función es que la sintaxis de una función contiene la palabra reservada RETURNS para indicar que tipo de dato se retornará.

Ejemplo:

```
DELIMITER //

CREATE FUNCTION factorial(x INT) RETURNS INT(11)
BEGIN
        DECLARE factorial INT;

        -- Guardamos el valor de x
        SET factorial = x ;

        -- Caso en que x sea menor o igual a 0
        IF x <= 0 THEN
        RETURN 1;
        END IF;

        -- Iteramos para obtener multiplicaciones
        consecutivas

        bucle: LOOP

        -- Cada iteracion reducimos en 1 a x
        SET x = x - 1 ;

        -- Condición de parada del bucle
        IF x<1 THEN
        LEAVE bucle;
        END IF;

        -- Factorial parcial
        SET factorial = factorial * x ;

        END LOOP bucle;

        -- Retornamos en el factorial
        RETURN factorial;

END//
DELIMITER ;
```

No confundir RETURNS con RETURN. La primera es para indicar el tipo de dato de retorno de la función y la segunda es para retornar el valor en el cuerpo de la función.

Como mostrar una función dentro de una sentencia SELECT

A continuación, crearemos una función que retorne en el nombre completo de la prioridad de un cliente, introduciendo como parámetro el campo prioridad.

Creación de la función

```
DELIMITER //
CREATE FUNCTION EXT_PRIORIDAD (cliente_prioridad VARCHAR(5)) RETURNS VARCHAR(20)
BEGIN
        CASE cliente_prioridad

        WHEN 'A' THEN
            RETURN 'Alto';
        WHEN 'M' THEN
            RETURN 'Medio';
        WHEN 'B' THEN
            RETURN 'Bajo';
        ELSE
            RETURN 'NN';
        END CASE;
END//
DELIMITER ;
```

Con ella podremos consultar de la siguiente forma a los clientes de la base de datos:

```
SELECT NOMBRE, APELLIDO, EXT_PRIORIDAD(PRIORIDAD)
FROM CLIENTE;
```

Como actualizar una función

Para actualizar una función usamos el comando ALTER FUNCTION. Con esta sentencia podemos cambiar los atributos de la función, pero no podremos cambiar el cuerpo. Veamos la sintaxis:

```
ALTER FUNCTION nombre_funcion
[SQL SECURITY {DEFINER|INVOKER}]
[COMMENT descripción ]
```

Si quisiéramos añadir una descripción a una función que calcula el promedio de huéspedes diario con respecto a una fecha llamada promedio_huespedes, hacemos lo siguiente:

```
ALTER FUNCTION promedio_huespedes
COMMENT 'Calculo del promedio diario de huéspedes entre una fecha inicial y una
```

Como borrar una función

Usando el comando DROP FUNCTION. Simplemente especificamos el nombre de la función y esta se borrará de la base de datos. Su sintaxis está definida de la siguiente forma:

```
DROP FUNCTION nombre_funcion
```

Por ejemplo, para borrar una función que retorna en el ingreso neto con respecto a todos los tickets aéreos comprados en una sucursal de una aerolínea, llamada ingreso_neto_sucursal

```
DROP FUNCTION ingreso_neto_sucursal;
```