Módulo 4. SQL Avanzado y NoSQL

Introducción

En el módulo 4 veremos conceptos avanzados de SQL, como los procedimientos almacenados, funciones, SQL dinámico y múltiples ejemplos relacionados. Aprenderemos cómo crear un disparador, qué son los cursores y cómo manejar errores. Por último, aprenderemos qué es una base de datos NoSQL, los tipos existentes y algunos ejemplos con MongoDB. A modo de cierre, presentaremos una comparativa entre bases SQL y NoSQL.

Etapa 3: Mejorando nuestro proyecto

Video de inmersión

Unidad 1. SQL avanzado

Tema 1. Procedimiento almacenado y función

La tendencia es que las bases de datos asuman un papel cada vez más importante en la arquitectura general del procesamiento de datos. Así es que el DBMS asumió más responsabilidad, lo cual le proporcionó un control más centralizado y redujo la posibilidad de corrupción de datos originados a partir de errores de programación de las aplicaciones. Tres características han sido parte de esta tendencia: procedimientos almacenados, funciones y disparadores.

Los procedimientos almacenados pueden realizar tanto el procesamiento de datos como la lógica de aplicación dentro de la misma base de datos. Por ejemplo, un procedimiento almacenado podría implementar la lógica para aceptar una orden del cliente o para transferir dinero de una cuenta bancaria a otra.

Funciones

Las funciones son programas SQL almacenados que devuelven un solo valor para cada fila de datos. A diferencia de los procedimientos almacenados, las funciones se invocan a través de sentencias SQL en casi cualquier cláusula en la que se pueda utilizar un nombre de columna. Esto las hace ideales para realizar cálculos y transformaciones de datos en los datos que se muestran en los resultados de la consulta o utilizados en las condiciones de búsqueda. Casi todos los productos DBMS relacionales vienen con un conjunto de funciones suministradas por el proveedor para uso general y, por lo tanto, las funciones añadidas por los usuarios de la base de datos local a menudo se denominan “funciones definidas por el usuario”.

Los disparadores o triggers se utilizan para invocar automáticamente la capacidad de procesamiento de un procedimiento almacenado en función de las condiciones que surgen dentro de la base de datos. Por ejemplo, un disparador puede transferir fondos automáticamente de una cuenta de ahorros a una cuenta corriente si la cuenta corriente se convierte en sobregiro.

Procedimientos almacenados

Con procedimientos almacenados, el lenguaje SQL se amplía en varias capacidades normalmente asociadas con lenguajes de programación. Las secuencias de sentencias SQL se agrupan para formar programas o procedimientos SQL. Tanto para los procedimientos como para las funciones se utiliza una extensión de SQL denominada PL/SQL.

Se proporcionan las siguientes capacidades:

- **Ejecución condicional:** una estructura IF... THEN ... ELSE permite un uso de un procedimiento SQL para probar una condición y llevar a cabo diferentes operaciones dependiendo del resultado.
- **Bucle:** un ciclo WHILE o FOR o una estructura similar permiten realizar una secuencia de operaciones SQL repetidamente hasta que se cumpla alguna condición de

terminación. Algunas implementaciones proporcionan una estructura de bucle especial con base en cursor para procesar cada fila de resultados de consulta.

- **Estructura de bloques:** una secuencia de sentencias SQL puede agruparse en un solo bloque y utilizarse en otras construcciones de flujo de control como si el bloque de instrucciones fuera una sola sentencia.
- **Variables nombradas:** un procedimiento SQL puede almacenar un valor que ha calculado, recuperado de la base de datos o derivado de alguna otra forma en una variable de programa, y recuperar posteriormente el valor almacenado para su uso en cálculos subsiguientes.
- **Procedimientos con nombre:** una secuencia de sentencias SQL puede agruparse, asignar un nombre y asignar parámetros formales de entrada y salida, como una subrutina o una función en un lenguaje de programación convencional. Una vez definido de esta manera, el procedimiento puede ser llamado por nombre y pasar los valores apropiados para sus parámetros de entrada. Si el procedimiento es una función que devuelve un valor, puede utilizarse en expresiones de valor de SQL.

Colectivamente, las estructuras que implementan estas capacidades forman un lenguaje procedimental almacenado (SPL - *structure program language*).

Ejemplo de procedimiento almacenado

Consideremos el proceso de agregar un cliente a la base de datos. Los pasos que pueden estar involucrados son los siguientes:

- Obtener el número de cliente, nombre, límite de crédito y cantidad de ventas objetivo para el cliente, así como el vendedor y la oficina asignados.
- Agregar una fila a la tabla de cliente que contiene los datos del cliente.
- Actualizar la fila del vendedor asignado, aumentando el objetivo de cuota en la cantidad especificada.
- Actualizar la fila de la oficina, aumentando la meta de ventas en la cantidad especificada.

- Confirmar los cambios en la base de datos, si todas las declaraciones anteriores tuvieron éxito.

Sin una capacidad de procedimiento almacenado, hay una secuencia de instrucciones SQL que hace este trabajo para XYZ Corporation, el nuevo número de cliente 2137, con un límite de crédito de $ 30 000 y ventas de $ 50 000 para asignar a Paul Cruz (empleado N.° 103) de la oficina de Chicago:

```
INSERT INTO CUSTOMERS (CUST_NUM, COMPANY, CUST_REP, CREDIT_LIMIT) VALUES (2137, 'XYZ Corporation', 103, 30000.00);
UPDATE SALESREPS SET QUOTA = QUOTA + 50000.00 WHERE EMPL_NUM = 103;
UPDATE OFFICES SET TARGET = TARGET + 50000.00 WHERE CITY = 'Chicago';
COMMIT;
```

Con un procedimiento almacenado, todo este trabajo se puede integrar en una única rutina SQL definida.

Tabla 1: Procedimiento almacenado básico en PL/SQL

	Procedimiento almacenado básico en PL/SQL
Create Procedure Add_Cust	**Cabecera y nombre**
(c_Name **In Varchar2,** c_Num **In Integer,** Cred_Lim **In Number,** Tgt_Sls **In Number,** c_Rep **In Integer,** c_Offc **In Varchar2,** c_quota **Out Integer,** c_target **Out Varchar2)**	**Parámetros**
As	
new_cuota **number(16,2);** new_target **number(16,2);**	**Variables**
Begin	

Insert **Into** Customers **(**Cust_Num**,** Company**,** Cust_Rep**,** Credit_Limit**)** **Values** **(**c_Num**,** c_Name**,** c_Rep**,** Cred_Lim**);** New_quota := **Quota * 0.25**; **Update** Salesreps **Set Quota =** New_quota **+** Tgt_Sls **Where** Empl_Num **=** c_Rep**;**	**Cuerpo**

Fuente: elaboración propia

Tabla 2: Procedimiento almacenado básico en PL/SQL

Select Quota **Into** c_quota **From** Salesreps **Where** Empl_Num **=** c_Rep**;** New_target := Target *** 0.75**; **Update** Offices **Set** Target **=** New_Target **+** Tgt_Sls **Where** City **=** c_Offc**;** **Select** Target **Into** c_Target **From** Offices **Where** City **=** c_Offc**;** **Commit;**	
End;	**Fin**

Fuente: elaboración propia

Creación de un procedimiento almacenado

Nombre del procedimiento

La instrucción CREATE PROCEDURE le asigna un nombre al procedimiento recién definido, que se utiliza luego para invocarlo.

Parámetros

- Acepta cero o más parámetros como argumentos.
- Se debe especificar el nombre de los parámetros y sus tipos de datos, soportados por el DBMS.

Pueden ser los siguientes:

- Entrada [IN].
- Salida [OUT].
- Entrada-salida [IN-OUT].

Cuando se llama al procedimiento, se asignan a los parámetros los valores especificados en la llamada de procedimiento, y las instrucciones en el cuerpo del procedimiento comienzan a ejecutarse.

Los nombres de los parámetros pueden aparecer dentro del cuerpo del procedimiento, dondequiera que pueda aparecer una constante. Cuando aparece un nombre de parámetro, el DBMS utiliza su valor actual.

Además de los parámetros de entrada, algunos lenguajes SPL también admiten parámetros de salida. Esto permite que un procedimiento almacenado devuelva valores que calcula durante su ejecución. Los parámetros de salida proporcionan una capacidad importante para pasar información de un procedimiento almacenado a otro procedimiento almacenado que lo llama, y también puede ser útil para depurar procedimientos almacenados utilizando SQL interactivo.

Algunos lenguajes SPL admiten parámetros que funcionan como parámetros de entrada y salida. En este caso, el parámetro pasa un valor al procedimiento almacenado y cualquier cambio en el valor durante la ejecución del procedimiento se refleja en el procedimiento de

llamada.

Variables

Se declaran al principio del cuerpo del procedimiento, justo después del encabezado del procedimiento y antes de la lista de sentencias SQL. Los tipos de datos de las variables pueden ser cualquiera de los tipos de datos SQL soportados como tipos de datos de columna por el DBMS.

Las variables locales dentro de un procedimiento almacenado pueden utilizarse como fuente de datos dentro de las expresiones SQL en cualquier lugar en que pueda aparecer una constante. El valor actual de la variable se utiliza en la ejecución de la sentencia. Además, las variables locales pueden ser destinos para datos derivados de expresiones o consultas SQL.

Llamando a un procedimiento almacenado

Una vez definido por la instrucción CREATE PROCEDURE, el procedimiento se puede utilizar. Un programa de aplicación puede solicitar la ejecución del procedimiento almacenado si utiliza la instrucción SQL adecuada. Otro procedimiento almacenado puede llamar para realizar una función específica. El procedimiento almacenado también se puede invocar a través de una interfaz interactiva de SQL.

Tabla 3: Llamando y eliminando un procedimiento almacenado en PL/SQL

El programa llama al procedimiento almacenado y le pasa los seis valores especificados como sus parámetros. El DBMS ejecuta el procedimiento almacenado y lleva a cabo cada sentencia SQL en la definición del procedimiento, una por una. Si el procedimiento ADD_CUST completa su ejecución con éxito, se ha llevado a cabo una transacción confirmada dentro del DBMS. Si esto no ocurre, el código de error devuelto y el mensaje indican qué fue lo que falló.
Los valores que se deben utilizar para

EXECUTE ADD_CUST ('XYZ Corporation', 2137, 30000.00, 50000.00, 103,'Chicago');	los parámetros del procedimiento se especifican, por orden, en una lista que está encerrada entre paréntesis.
ADD_CUST ('XYZ Corporation', 2137, 30000.00, 50000.00, 103,'Chicago');	Cuando se llama desde dentro de otro procedimiento o un disparador, se puede omitir la instrucción EXECUTE.
EXECUTE ADD_CUST (c_name = 'XYZ Corporation', c_num = 2137, cred_lim = 30000.00, c_offc = 'Chicago', c_rep = 103, tgt_sales = 50000.00);	El procedimiento también se puede llamar mediante los parámetros nombrados, en cuyo caso los valores de parámetro se pueden especificar en cualquier secuencia.
Eliminando un procedimiento almacenado en PL/SQL	
DROP PROCEDURE ADD_CUST;	

Fuente: elaboración propia

Funciones

Además de los procedimientos almacenados, la mayoría de los lenguajes SPL soporta funciones almacenadas. La diferencia es que una función devuelve una sola cosa (como un valor de datos, un objeto o un documento XML) cada vez que se invoca, mientras que un procedimiento almacenado puede devolver muchas cosas o nada. El soporte para los valores devueltos varía según el lenguaje SPL. Las funciones se utilizan comúnmente como expresiones de columna en sentencias SELECT y, por lo tanto, se invocan una vez por fila en el conjunto de resultados. Esto permite que la función realice cálculos, conversión de datos y otros procesos para producir el valor devuelto para la columna. A continuación, se muestra un ejemplo sencillo de una función almacenada.

Tabla 4: Una función PL/SQL de Oracle

	Una función PL/SQL de Oracle
SELECT COMPANY, NAME FROM CUSTOMERS, SALESREPS	

```
WHERE CUST_REP = EMPL_NUM
AND GET_TOT_ORDS(CUST_NUM) > 10000.00;
```

```
/* Return total order amount for a customer */ create function get_tot_ords(c_num in number) return number
as
/* Declare one local variable to hold the total */ tot_ord number(16,2);
begin
/* Simple single-row query to get total */ select sum(amount) into tot_ord
from orders
where cust = c_num;
/* return the retrieved value as fcn value */ return tot_ord;
end;
```

Fuente: elaboración propia

Supongamos que queremos definir un procedimiento almacenado y que, dado un número de cliente, calculamos la cantidad total de pedido actual para ese cliente. Si definimos el procedimiento SQL como una función, la cantidad total se puede devolver como su valor. La tabla 4 muestra una función de Oracle que calcula la cantidad total de pedidos actuales para un cliente, dado el número de cliente. La cláusula RETURN, en la definición del procedimiento, le indica al DBMS el tipo de datos del valor que se devuelve. En la mayoría de los productos DBMS, si introduce una llamada de función a través de la capacidad interactiva de SQL, el valor de la función se muestra en respuesta. Dentro de un procedimiento almacenado, podemos llamar a una función almacenada y utilizar su valor devuelto en los cálculos o almacenarla en una variable. Muchos lenguajes SPL también permiten utilizar una función como una función definida por el usuario dentro de expresiones de valor de SQL.

A medida que el DBMS evalúa la condición de búsqueda para cada fila de resultados de la consulta, utiliza el número de cliente de la fila candidata actual (CUST_NUM) como un argumento a la función GET_TOT_ ORDS y comprueba si supera el umbral de $ 10 000. Esta misma consulta podría expresarse como una consulta agrupada, con la tabla ORDERS también incluida en la cláusula FROM y los resultados agrupados por el cliente y el vendedor. En muchas implementaciones, el DBMS lleva a cabo la consulta agrupada más

eficientemente que la precedente, lo que probablemente obliga al DBMS a procesar la tabla de pedidos una vez para cada cliente.

Tema 2. SQL dinámico

El concepto central de SQL dinámico es simple: no codifiquemos una sentencia SQL incrustada en el código fuente del programa. En su lugar, dejemos que el programa construya el texto de una instrucción SQL en una de sus áreas de datos en tiempo de ejecución. A continuación, el programa pasa el texto de la sentencia al DBMS para su ejecución sobre la marcha. Aunque los detalles se vuelven bastante complejos, todo el SQL dinámico se basa en este concepto simple, y es una buena idea tenerlo en cuenta.

En SQL dinámico, la situación es muy diferente. La instrucción SQL que se ejecutará no se conoce hasta el tiempo de ejecución, por lo que el DBMS no puede prepararse para la declaración de antemano. Cuando el programa se ejecuta realmente, el DBMS recibe el texto de la sentencia para ser ejecutado dinámicamente (esto es conocido como “la cadena de instrucción”). El SQL dinámico es menos eficiente que el SQL estático. Sin embargo, el SQL dinámico ha crecido en importancia.

Ejecución dinámica de instrucciones (EXECUTE IMMEDIATE)

La forma más simple de SQL dinámico es proporcionada por la instrucción EXECUTE IMMEDIATE. Esta sentencia pasa el texto de una instrucción SQL dinámica al DBMS y le pide al DBMS que ejecute la instrucción dinámica de inmediato. Para usar esta instrucción, el programa pasa por los siguientes pasos:

EXECUTE IMMEDIATE cadena de texto

1. El programa construye una sentencia SQL como una cadena de texto en una de sus áreas de datos y la almacena en la memoria como una variable con nombre. La sentencia puede ser casi cualquier sentencia SQL que no recupera datos.
2. El programa pasa la instrucción SQL al DBMS con la instrucción EXECUTE IMMEDIATE.

3. El DBMS ejecuta la sentencia y establece los valores SQLCODE / SQLSTATE para indicar el estado de finalización, exactamente como si la sentencia hubiera sido codificada mediante SQL estático.

La instrucción EXECUTE IMMEDIATE es la forma más simple de SQL dinámico, pero es muy versátil. Podemos utilizarla para ejecutar dinámicamente la mayoría de las sentencias DML, incluidas INSERT, DELETE, UPDATE, COMMIT y ROLLBACK. También podemos utilizar EXECUTE IMMEDIATE para ejecutar dinámicamente la mayoría de las sentencias DDL, incluidas las instrucciones CREATE, DROP, GRANT y REVOKE.

Sin embargo, la instrucción EXECUTE IMMEDIATE tiene una limitación significativa: no podemos utilizarla para ejecutar dinámicamente una sentencia SELECT, ya que no proporciona un mecanismo para procesar los resultados de la consulta. Así como SQL estático requiere cursores y declaraciones de propósito especial (DECLARE CURSOR, OPEN, FETCH y CLOSE) para consultas programáticas, SQL dinámico utiliza cursores y algunas nuevas instrucciones de propósito especial para manejar consultas dinámicas.

Es importante advertir que la entrada del usuario no debe colocarse directamente en sentencias SQL (como se muestra en los ejemplos simplificados anteriores) sin analizarlas primero. Hacerlo le permitiría a un hacker incluir caracteres en la entrada, que terminaría la sentencia SQL deseada y añadiría otra al final de ella, lo cual permitiría el acceso no autorizado a otros datos en la base de datos, una técnica conocida como “inyección de SQL”.

Tema 3. Disparadores (*triggers*)

Triggers

Cursores

Una necesidad común para la repetición de sentencias dentro de un procedimiento almacenado surge cuando el procedimiento ejecuta una consulta y necesita procesar los resultados de la consulta, fila por fila. Todos los lenguajes principales proporcionan una

estructura para este tipo de procesamiento. Conceptualmente, las estructuras son paralelas a las instrucciones DECLARE CURSOR, OPEN CURSOR, FETCH y CLOSE CURSOR en SQL incorporado o en las llamadas de la API SQL correspondientes. Sin embargo, en lugar de buscar los resultados de la consulta en el programa de aplicación, en este caso se están obteniendo en el procedimiento almacenado, que se ejecuta dentro del propio DBMS. En lugar de recuperar los resultados de la consulta en variables de programa de aplicación (variables de host), el procedimiento almacenado los recupera en variables de procedimiento almacenadas locales. Para ilustrar esta capacidad, asumamos que deseamos rellenar dos tablas con datos de la tabla ORDERS. Una tabla, llamada BIGORDERS, debe contener el nombre del cliente y el tamaño del pedido para cualquier pedido de más de $ 10 000. El otro, SMALLORDERS, debe contener el nombre del vendedor y el tamaño de la orden para cualquier orden inferior a $ 1000. La mejor y más eficiente manera de hacerlo sería utilizar dos sentencias SQL INSERT independientes con subconsultas, pero con fines de ilustración. Consideremos el siguiente método en su lugar:

- Ejecutemos una consulta para recuperar el importe de la orden, el nombre del cliente y el nombre del vendedor para cada pedido.
- Para cada fila de resultados de la consulta, comprobemos el monto de la orden para corroborar que caiga en el rango adecuado para incluir en las tablas BIGORDERS o SMALLORDERS.
- Dependiendo de la cantidad, debemos INSERTAR la fila apropiada en la tabla BIGORDERS o SMALLORDERS.
- Repitamos los pasos 2 y 3 hasta que se agoten todas las filas de los resultados de la consulta.
- Confirmemos las actualizaciones a la base de datos.

Tabla 5: Un bucle FOR basado en el cursor PL/SQL

Un bucle FOR basado en el cursor PL/SQL
create procedure sort_orders() /* Cursor for the query */ cursor o_cursor is

```
select amount, company, name from orders, customers, salesreps where cust
= cust_num
and rep = empl_num;
/* Row variable to receive query results values */ curs_row o_cursor%rowtype;
begin
/* Loop through each row of query results */ for curs_row in o_cursor
loop
/* Check for small orders and handle */ if (curs_row.amount < 1000.00)
then insert into smallorders
values (curs_row.name, curs_row.amount);
/* Check for big orders and handle */ elsif (curs_row.amount > 10000.00) then
insert into bigorders
values (curs_row.company, curs_row.amount); end if;
end loop; commit; end;
```

Fuente: elaboración propia

La tabla 5 muestra un procedimiento almacenado de Oracle que realiza este método. El cursor que define la consulta se define temprano en el procedimiento y se le asigna el nombre O_CURSOR. La variable CURS_ROW se define como un tipo de fila de Oracle. Es una variable de fila estructurada de Oracle con componentes individuales (como una estructura en lenguaje C). Al declarar que tiene el mismo tipo de fila que el cursor, los componentes individuales de CURS_ROW tienen los mismos tipos de datos y nombres que las columnas de resultados de la consulta del cursor.

La consulta descrita por el cursor se realiza, en realidad, mediante el bucle FOR con base en el cursor. Básicamente, le dice al DBMS que realice la consulta descrita por el cursor (equivalente a la instrucción OPEN en SQL incorporado) antes de iniciar el procesamiento de bucle. El DBMS, entonces, ejecuta el bucle FOR repetidamente, busca una fila de resultados de consulta en la parte superior del bucle, coloca los valores de columna en la variable CURS_ROW y luego ejecuta las instrucciones en el cuerpo del bucle. Cuando no se busquen más filas de resultados de consulta, el cursor se cierra y el procesamiento continúa después del bucle.

Disparadores

A diferencia de los procedimientos almacenados, un disparador no se activa mediante una instrucción CALL o EXECUTE, sino que se asocia con una tabla de base de datos. Cuando los datos de la tabla cambian por una instrucción INSERT, DELETE o UPDATE, se lanza el disparador, lo que significa que el DBMS ejecuta las instrucciones SQL que forman el cuerpo del disparador. Algunas marcas de DBMS permiten la definición de actualizaciones específicas que provocan un disparo, y algunas de estas marcas, especialmente Oracle, permiten que los disparadores tengan base en eventos del sistema, como los usuarios que se conectan a la base de datos o la ejecución de un comando de apagado de la base de datos.

Los disparadores se pueden utilizar para provocar actualizaciones automáticas de la información dentro de una base de datos. Por ejemplo, supongamos que deseamos configurar la base de datos para que cada vez que se inserte un nuevo vendedor en la tabla SALESREPS, el objetivo de ventas de la oficina donde trabaja el vendedor se actualice por la cuota del nuevo vendedor. A continuación, presentamos un disparador de Oracle PL/SQL que logra este objetivo:

```
CREATE OR REPLACE Disparador upd_tgt before insert on salesreps
for each row begin
if :new.quota is not null then
update offices
set target = target + new.quota; END IF;
END;
```

El cuerpo de este disparador le dice al DBMS que, para cada nueva fila insertada en la tabla, debería ejecutar la instrucción UPDATE especificada para la tabla OFFICES. El valor QUOTA de la fila SALESREPS recién insertada se denomina NEW.QUOTA dentro del cuerpo del disparador.

Actividad de repaso

¿Las funciones y los procedimientos almacenados difieren solo en la cantidad de parámetros?

Verdadero.

Falso.

Justificación

Before update

create Disparador bef_upd_ord before update on orders

begin

/* Calculate order total before changes */ old_total = add_orders();

end;

create Disparador aft_upd_ord after update on orders

begin

/* Calculate order total after changes */ new_total = add_orders();

end;

create Disparador dur_upd_ord before update of amount on orders referencing old as pre new as post

/* Capture order increases and decreases */ for each row

when (:post.amount != :pre.amount) begin

if post.amount != :pre.amount) then

if (:post.amount < :pre.amount) then

/* Write decrease data into table */ insert into ord_less

values (:pre.cust,

:pre.order_date,

:pre.amount,

```
:post.amount);
elsif (:post.amount > :pre.amount) then
/* Write increase data into table */ insert into ord_more
values (:pre.cust,
:pre.order_date,
:pre.amount,
:post.amount); end if;
end if; end;
```

Tema 4. Manejo de errores

Cuando escribimos una instrucción SQL interactiva que causa un error, el programa de SQL interactivo mostrará un mensaje de error, anulará la instrucción y nos pedirá que escribamos una nueva instrucción.

Tipos de errores

- **Errores en tiempo de compilación:** las comillas mal colocadas, las palabras clave de SQL mal escritas y los errores similares en sentencias de SQL embebido son detectados por el precompilador de SQL e informados al programador.
- **Errores en tiempo de ejecución:** el intento de insertar un valor de datos no válido o la falta de permisos para actualizar una tabla solo se puede detectar en tiempo de ejecución. Estos errores deben ser detectados y manejados por el programa de aplicación. En los programas de SQL embebido, el DBMS informa al programa de aplicación de errores de ejecución a través de un código de error devuelto. Si se detecta un error, están disponibles una descripción más detallada del error y otra información sobre la sentencia que acaba de ejecutarse a través de información de diagnóstico adicional.

Tratamiento de errores con SQLCODE

A medida que el DBMS ejecuta cada sentencia de SQL embebido, establece el valor de la

variable SQLCODE en el SQLCA (área de comunicaciones de SQL) para indicar el estado de la sentencia:

- Un SQLCODE de cero indica la finalización satisfactoria de la sentencia, sin errores ni advertencias.
- Un valor SQLCODE negativo indica un error grave que impidió que la instrucción se ejecutara correctamente. Por ejemplo, un intento de actualizar una vista de solo lectura produciría un valor SQLCODE negativo.
- Un valor SQLCODE positivo indica una condición de advertencia. Por ejemplo, el truncamiento o redondeo de un elemento de datos recuperado por el programa produciría una advertencia.

La advertencia más común, con un valor de +100 en la mayoría de las implementaciones y en el estándar SQL, es el aviso de ausencia de datos, que es devuelto cuando un programa intenta recuperar la siguiente fila de resultados de consulta y no quedan más filas para recuperar. Debido a que cada sentencia SQL embebido puede, potencialmente, generar un error, un programa bien escrito comprobará el valor de SQLCODE después de cada sentencia de SQL embebido ejecutable.

Unidad 2. Bases NoSQL

Base de datos NOSQL

Son muchas las aplicaciones web que utilizan algún tipo de bases de datos para funcionar.

> Hasta ahora estábamos acostumbrados a utilizar bases de datos SQL, pero desde hace ya algún tiempo han aparecido otras que reciben el nombre de NoSQL (*not only SQL* – no solo SQL) y que han llegado con la intención de hacer frente a las bases relacionales utilizadas por la mayoría de los usuarios.
>
> Se puede decir que la aparición del término NoSQL aparece con la llegada de la web 2.0, ya que hasta ese momento solo subían contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter o YouTube, cualquier usuario podía subir contenido, provocando así un crecimiento exponencial de los datos. (Acens, 2014, https://bit.ly/2OTiVgi)

Es en este momento cuando empiezan a aparecer los primeros problemas de la gestión de toda esa información almacenada en bases de datos relacionales. En un principio, para solucionar estos problemas de accesibilidad, las empresas optaban por utilizar un mayor número de máquinas, pero pronto se dieron cuenta de que esto no solucionaba el problema, además de que resultaba ser una solución muy cara.

> La otra solución era la creación de sistemas pensados para un uso específico que con el paso del tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL. Por lo tanto, hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas debido principalmente a problemas de escalabilidad y rendimiento donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias. Además de lo comentado anteriormente, las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad–relación. Tampoco utilizan una estructura de datos en forma de tabla donde se van almacenando los datos, sino que para el almacenamiento hacen uso de otros formatos como clave–valor, mapeo de columnas o grafos. (Acens, 2014, https://bit.ly/2OTiVgi)

Tema 1. Tipos de bases NoSQL

Dependiendo de la forma en la que se almacene la información, podemos encontrar varios tipos distintos de bases de datos NoSQL. Veamos los tipos más utilizados.

Bases de datos clave–valor

> Son el modelo de base de datos NoSQL más popular, además de ser el más sencillo en cuanto a funcionalidad. En este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB). Se caracterizan por ser muy eficientes tanto para las lecturas como para las escrituras. (Acens, 2014, https://bit.ly/2OTiVgi)

Este concepto es el mismo que vimos en el módulo 1 de tablas hash. Algunos ejemplos de este tipo de bases son Cassandra, BigTable o HBase.

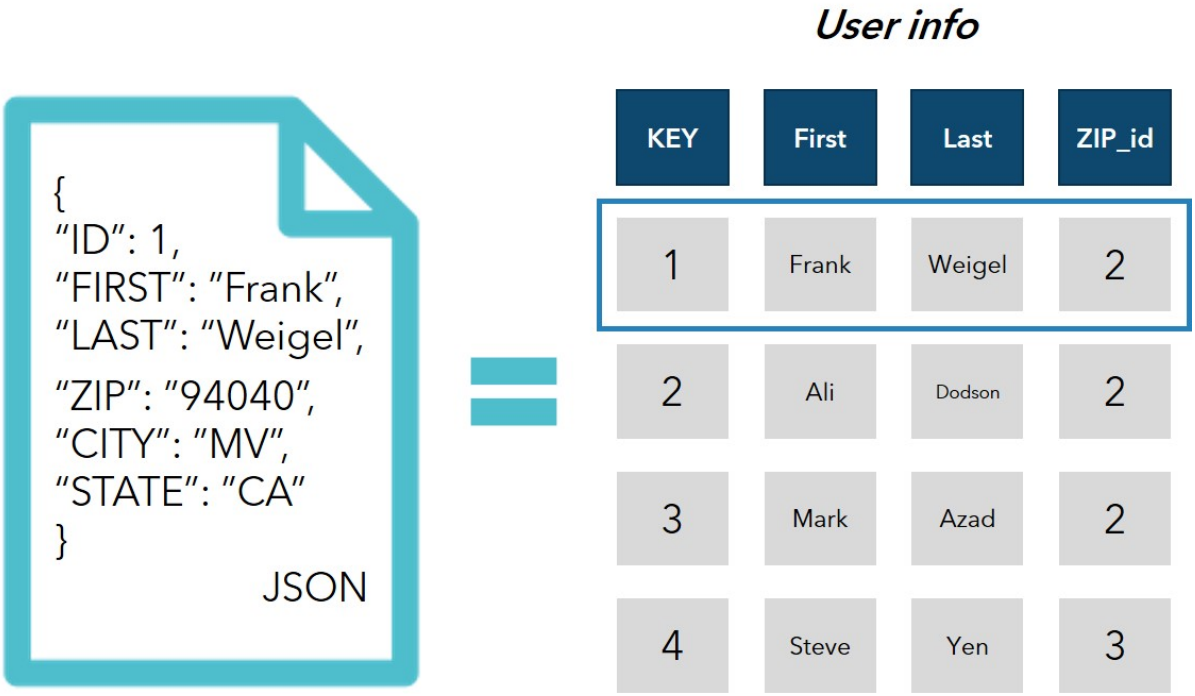
Bases de datos documentales

Base de datos no relacionales

Se almacena la información como un documento, generalmente utilizando para ello una estructura simple como JSON o XML y donde se utiliza una clave única para cada registro. Este tipo de implementación permite, además de realizar búsquedas por clave–valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de este tipo son MongoDB o CouchDB. (Acens, 2014, https://bit.ly/2OTiVgi)

En el siguiente tema veremos algunos ejemplos con MongoDB, pero para adelantarnos podemos ver en la siguiente figura cómo es que se almacena la información en este tipo de bases.

Figura 1: Bases de datos documentales



Fuente: Acens, 2014, https://bit.ly/2OTiVgi

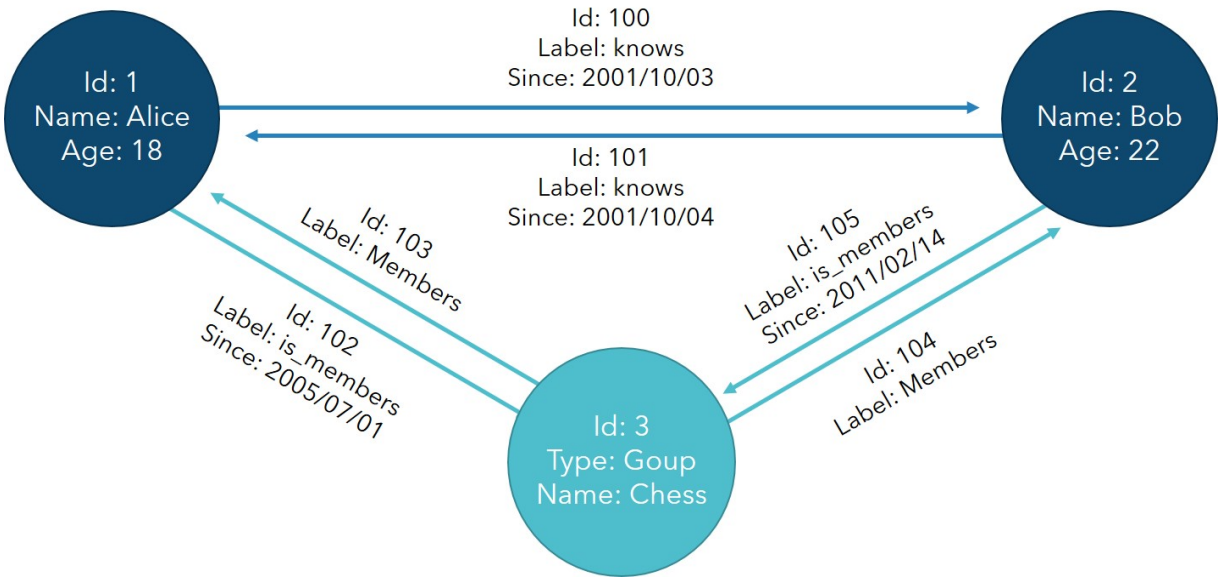
Bases de datos en grafo

En este tipo de bases de datos, la información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos. Este tipo de bases de datos ofrece

una navegación más eficiente entre relaciones que en un modelo relacional. (Acens, 2014, https://bit.ly/2OTiVgi)

Algunos ejemplos de este tipo son Neo4j, InfoGrid o Virtuoso. Veamos un ejemplo gráfico de cómo sería este tipo de bases.

Figura 2: Bases de datos en grafo



Fuente: Acens, 2014, https://bit.ly/2OTiVgi

Bases de datos orientadas a objetos

En este tipo de bases, la información se representa mediante objetos, de la misma forma que son representados en los lenguajes de programación orientada a objetos (POO), como ocurre en JAVA, C# o Visual Basic .NET. Algunos ejemplos de este tipo de bases de datos son Zope, Gemstone o Db4o.

Tema 2. Creación y consultas en BD No SQL

En el siguiente tema veremos algunos ejemplos de interacción con una base de datos NoSQL. Todos estos ejemplos corresponden a los casos en los que se usa el motor **MongoDB.**

En MongoDB **no existe ningún comando estilo *create database*** o algo parecido. Lo que hace Mongodb es crear una colección (base de datos) en el momento que se le inserta un objeto o documento (registro de una tabla, por llamarlo de alguna forma) a dicha colección.

Para crear una base de datos en MongoDB hay que usar, con la sentencia use, una base de datos o colección que todavía no existe.

use peliculas;

Ahora, vamos a insertar una película en la colección y en ese momento esta se creará: ***db.peliculas.save({titulo:'Batman el caballero oscuro'}).***

Ahora, hemos creado un registro y la base de datos a la vez, aunque en Mongodb realmente lo que creamos son colecciones y guardamos documentos json.

Podemos hacer una consulta para ver que hay dentro de la colección **db.peliculas.find().**

Podemos ver el resto de base de datos creadas con el comando: **show dbs**. (Robles, 2016, https://bit.ly/3238uMp)

Tema 3. Comparativa SQL vs. NoSQL

Tabla 6: Ventajas y desventajas según tipo de BD

	SQL (relacionales)	NoSQL
Ventajas	• Está más adaptado su uso y los perfiles que las conocen son mayoritarios y más baratos. • Debido al largo tiempo que llevan en el mercado, estas herramientas tienen un mayor soporte y mejores *suites* de	• Su escalabilidad y su carácter descentralizado son una de sus ventajas. Soportan estructuras distribuidas. • Suelen ser bases de datos mucho más abiertas y flexibles. Permiten adaptarse a necesidades de proyectos mucho más

	productos y *add-ons* para gestionar. • Hay atomicidad en las operaciones en la base de datos. Esto quiere decir que en estas bases de datos o se hace la operación entera o no se hace, utilizando la famosa técnica del *rollback.* • Los datos deben cumplir requisitos de integridad tanto en tipo de dato como en compatibilidad.	fácilmente que los modelos de entidad relación. • Se puede hacer cambios de los esquemas sin tener que parar bases de datos. • Tienen escalabilidad horizontal: son capaces de crecer en número de máquinas, en lugar de tener que residir en grandes máquinas. • Se pueden ejecutar en máquinas con pocos recursos. • Permiten una optimización de consultas en base de datos para grandes cantidades de datos.
Desventajas	• La atomicidad de las operaciones juega un papel crucial en el rendimiento de las bases de datos. • Tiene una escalabilidad, que aunque probada en muchos entornos productivos, suele, por norma, ser inferior a las	• No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos. Soportan lo que se llama “consistencia eventual”. • Existen problemas de compatibilidad entre instrucciones SQL. Las nuevas bases de datos utilizan sus propias

	bases de datos NoSQL.	características en el lenguaje de consulta y no son 100 % compatibles con el SQL de las bases de datos relacionales. El soporte a problemas con las *queries* de trabajo en una base de datos NoSQL es más complicado. • Falta de estandarización. Hay muchas bases de datos NoSQL y aún no hay un estándar como sí lo hay en las bases de datos relacionales. Se presume un futuro incierto en estas bases de datos. • Soporte multiplataforma. Aún quedan muchas mejoras en algunos sistemas para que soporten sistemas operativos que no sean Linux. • Suelen tener herramientas de administración no muy usables o se accede por consola.

Fuente: elaboración propia en base a Pandorafms, 2015

NoSQL vs SQL; cuándo utilizar qué tipo de base de datos

- Cuando los datos deben ser consistentes sin dar posibilidad al error de utilizar una base de datos relacional, SQL.
- Cuando nuestro presupuesto no se puede permitir grandes máquinas y debe destinarse a máquinas de menor rendimiento, NoSQL.
- Cuando las estructuras de datos que manejamos son variables, NoSQL.
- Cuando hay que analizar grandes cantidades de datos en modo lectura, NoSQL.
- Cuando hay que capturar y procesar eventos, NoSQL.
- Cuando se trata de tiendas *online* con motores de inteligencia complejos, NoSQL.

Actividad de repaso

¿Cuáles de las siguientes opciones corresponden a tipos de bases de datos NoSQL?

Clave-valor.

Documentales.

Dibujadas.

Grafo.

Pictóricas.

Justificación

Microactividades

La plataforma que utilizaremos será: https://extendsclass.com/postgresql/e176003

Si utilizamos pgAdmin, todas las tablas, datos, funciones, procedimientos almacenados y *triggers* serán guardados en nuestra computadora de forma tal que podremos acceder a ellos indiferentemente si cerramos el programa o reiniciamos nuestros sistemas.

En el caso de utilizar la herramienta *online*, deberán guardar las consultas que vayan realizando, ya que, al cerrar el navegador se perderán todos los datos y se deberá copiar, pegar y ejecutar las consultas que ya habíamos ejecutado (y que guardado en un documento de texto cualquiera de los disponibles). De esta forma podrán avanzar en las próximas microactividades sin necesidad de volver a escribirlas una a una nuevamente.

A lo largo de todas las microactividades se debe crear y trabajar sobre un sistema para una universidad.

Desde la universidad Puls-ar nos solicitan llevar el registro de los alumnos y las materias que ellos cursan. Cuando los alumnos se inscriben se les solicita su DNI, nombre, apellido y dirección de correo electrónico.

Dado de alta en la universidad ellos ya están en condiciones de anotarse a las materias disponibles, teniendo en cuenta que las materias tienen un nombre y un año de cursado.

1. Crear la función cantidad_inscriptos donde pasaremos el identificador de materia y nos retornará la cantidad de inscriptos (el valor de retorno debe ser del tipo de datos *bigint* ya que en algún momento podemos tener muchos alumnos inscriptos a una materia).
2. Probar la función anterior para encontrar la cantidad de alumnos inscriptos a la materia 1 (escribir la consulta realizada).

Tabla 7. Ejercicio

cantidad_inscriptos
3

Fuente: elaboración propia.

3. Crear una tabla llamada alumnos_logs idem de alumnos, pero agregando un campo llamado fecha para registrar un tipo de datos fecha y hora. Esta tabla, a su vez, no debe tener

un identificador único. La utilidad de esta tabla la definiremos más adelante para poder registrar los cambios realizados sobre los alumnos.

4. Crear un procedimiento almacenado que luego utilizaremos para llamar desde un *trigger*. La idea de este procedimiento almacenado es registrar las modificaciones ocurridas sobre la tabla alumnos cuando sus registros se actualicen. Estas modificaciones las guardaremos dentro de nuestra nueva tabla alumnos_logs. Este procedimiento almacenado debe llamarse alumnos_update_trigger_fnc().

5. Crear el *trigger* que monitoree nuestras tablas alumnos. En caso de actualización de algún registro, deberá llamar al procedimiento almacenado anteriormente para ir agregando los registros en la tabla alumnos_logs.

6. Modificar la tabla de alumnos de forma que se completen mediante el *trigger* (de forma automática) los registros en la tabla alumnos_logs para obtener el siguiente esquema de la misma. Recordar que la columna fecha tendrá otros valores ya que se registrará la fecha hora del momento en que ejecuten las actualizaciones.

Tabla 8. Ejercicio

id	dni	nombre	apellido	email	fecha
1	1000000	Pedro	Picapiedras	pedro@picapiedras.com	2021-07-03T14:47:18.340Z
6	54354353	Ban Ban	Marmol	ban_ban@marmol.com	2021-07-03T14:49:21.194Z
9	12344325	Alicia	Pascuale	aliciapasc@hotmail.com	2021-07-03T14:50:27.971Z
9	12344325	Adriana	Pascuale	aliciapasc@hotmail.com	2021-07-03T14:51:32.002Z

Fuente: elaboración propia.

7. Realizar un *select* de la tabla log que muestre los resultados de la actualización realizada por el disparador o *trigger* ordenado por fecha.

8. Modificar la tabla de alumnos de forma que se completen mediante el *trigger* (de forma automática) los registros en la tabla alumnos_logs para obtener el siguiente esquema de la misma. Recordar que la columna fecha tendrá otros valores ya que se registrará la fecha hora

del momento en que ejecuten las actualizaciones.

Tabla 9. Ejercicio

id	dni	nombre	apellido	email	fecha
1	1000000	Pedro	Picapiedras	pedro@picapiedras.com	2021-07-03T14:47:18.340Z
6	54354353	Ban Ban	Marmol	ban_ban@marmol.com	2021-07-03T14:49:21.194Z
9	12344325	Alicia	Pascuale	aliciapasc@hotmail.com	2021-07-03T14:50:27.971Z
9	12344325	Adriana	Pascuale	aliciapasc@hotmail.com	2021-07-03T14:51:32.002Z

Fuente: elaboración propia.

9. Realizar un *select* de la tabla log que muestre los resultados de la actualización realizada por el disparador o *trigger* ordenado por fecha.

10. Crear un *trigger* que actualice en la tabla ALUMNOS_MATERIAS el campo final con el promedio de las notas del parcial 1 y parcial 2 cuando las notas del primer y segundo parcial sean mayor o igual a 6.

11. Actualizar los segundos parciales rendidos por los siguientes alumnos:

Tabla 10. Ejercicio

Alumno	Materia	Parcial1	Parcial2
Pedro Picapiedras	Fisica1	6	6
Pedro Picapiedras	Algebra 1	7	2
Vilma Picapiedras	Fisica1	4,5	8
Vilma Picapiedras	Base de datos	9	6
Stephen King	Matemática 1	7	5

Fuente: elaboración propia.

12. Realice una consulta de los alumnos que rindieron el segundo parcial, es decir que el campo parcial2 no es nulo.

13. Realice una consulta que contenga la siguiente información:

- Apellido y nombre del alumno.
- Nombre de la materia.
- Nota del primer parcial.
- Nota del segundo parcial.
- Nota del final.

Ordenado alfabéticamente por apellido y nombre ascendente con los estudiantes que hayan rendido, aunque sea un parcial.

Podrás descargar las resoluciones a las microactividades en el siguiente PDF:

Resolución microactividad

Fuente: elaboración propia.

Video de habilidades

Fuente: HolaMundo. (2019). ¿Qué es sql y nosql? ¿Cuáles son sus diferencias y cuándo deberías utilizarlos? [Video de YouTube]. Recuperado de https://www.youtube.com/watch?v=zmXl2dOGWL8

En el video podemos ampliar y conocer más sobre las diferentes características que poseen las bases SQL y las NoSQL: rendimiento, capacidad, seguridad y disponibilidad. Tener claridad sobre estas diferencias es muy importante ya que nos permitirá elegir en qué situaciones utilizar cada tipo de base.

Preguntas de habilidades

Dos de las siguientes son características de las bases NoSQL:

Escalabilidad.

Flexibilidad (no hay esquema).

Atomicidad.

Inseguridad.

Justificación

Dos de las siguientes son características de las bases SQL:

Confiabilidad.

Integridad/Consistencia.

Escalabilidad.

Velocidad.

Justificación

Las bases de datos documentales son un tipo de bases relacional (SQL):

Verdadero.

Falso.

Justificación

Las bases de datos NoSQL, a diferencia de las ……. no poseen una estructura definida.

Relacionales.

Documentales.

Grafo.

SiSQL.

Justificación

Las bases de datos NoSQL son más eficientes en las ……. de grandes cantidades de datos, respecto de las relacionales.

Escrituras.

Actualización.

Lecturas.

Eliminación.

Justificación

Cierre

En este módulo estudiamos SQL avanzado y NoSQL: sus diferencias, similitudes y para que podemos utilizar a cada uno de ellos. Con el cierre de este módulo tendrás nuevas herramientas necesarias para poder desarrollarte en el campo profesional ¡felicitaciones!

Glosario

Referencias

Acens, (2014). *Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar.* Recuperado de https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

Pandorafms, (2015). *NoSQL vs SQL; principales diferencias y cuándo elegir cada una de ellas*. Recuperado de https://pandorafms.com/blog/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/

Robles, V. (2016). *Crear una base de datos en MongoDB.* Recuperado de https://victorroblesweb.es/2016/12/24/crear-una-base-datos-mongodb/