Bases de Datos / Elementos de Bases de Datos 2015

Stored Procedures, Triggers y Transacciones en MySQL

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Stored Procedures

- Un stored procedure (S.P.) es un procedimiento (como una subrutina en cualquier lenguaje de programación) almacenado en una B.D. que puede ser invocado externamente.
- Esta formado por un encabezado y un cuerpo.
- El encabezado contiene el nombre del procedimiento y una lista de parámetros.
- El cuerpo puede contener sentencias SQL y sentencias propias de un lenguaje de programación imperativo: declaración y asignación de variables, condicionales, repetición, etc.

Stored Procedures: Ventajas

- Mejora la performance: Reduce la comunicación por red (se ejecuta en el servidor).
- Las operaciones comunes y frecuentes pueden almacenarse en la B.D. como un S.P. y ser invocadas desde cualquier aplicación.
  - Reduce la complejidad de las aplicaciones que usan la B.D.
  - Las aplicaciones pueden compartir código.
  - Separamos la interfase de la lógica del sistema.
- Mejora la portabilidad del sistema. Los S.P. son portables, se pueden ejecutar en cualquier plataforma donde se ejecuta MySQL.
- Permite asociar los datos con las operaciones para manejar los datos.

Creando un Stored Procedure en MySQL

- Se utiliza la sentencia *create procedure.*

create procedure <*nombre*> ([*in* | *out* | *inout*] <*nombre_param*> <*tipo*>, …)
begin
  <*sentencia*>;
  ⋮
  <*sentencia*>;
end;

encabezado

cuerpo

- *In, out* e *inout* definen parámetros de entrada, salida y entrada-salida respectivamente.
- <*tipo*>: cualquier tipo válido de MySQL: int,char,date,etc.
- <*sentencia*>: sentencia SQL, condicionales, repetición, …
- El S.P. es validado sintácticamente y optimizado en el momento de la creación.

Stored Procedures: ejemplos

- Consideremos una B.D. de un banco que mantiene el saldo de cada cuenta.

```
CREATE DATABASE bank;
USE bank;
CREATE TABLE cuentas(
    numero INT UNSIGNED NOT NULL,
    saldo DECIMAL(7,2) NOT NULL,
    PRIMARY KEY(numero)
) ENGINE=InnoDB;

INSERT INTO cuentas VALUES (1, 1000);
INSERT INTO cuentas VALUES (2, 2000);
INSERT INTO cuentas VALUES (3, 3000);
```

Creando un Stored Procedure: ejemplo

```
use bank;
delimiter !   # define "!" como delimitador
create procedure p()
  begin
    SELECT * FROM cuentas;
  end; !
delimiter ; # reestablece el ";" como delimitador
```

- Antes de crear un S.P. debemos seleccionar la B.D. asociada. En este ejemplo: **`use bank;`**
- Dado que ";" se usa para separar las sentencias dentro del S.P. y también se usa para separar las sentencias sql, debemos cambiar el delimitador a "!" para que el cliente sepa donde termina la sentencia *create procedure.*
  Atención: no dejar espacios después de "!" o ";".

Invocando un Stored Procedure: ejemplo

- Una vez creado en S.P. puede invocarse a través de la sentencia *call.*

```
mysql> call p();
+--------+----------+
| numero | saldo    |
+--------+----------+
|      1 |  1000.00 |
|      2 |  2000.00 |
|      3 |  3000.00 |
+--------+----------+
```

- Los usuarios deben tener un privilegio especial para poder ejecutar un S.P.

```
grant execute on procedure bank.p to <usuario>;
```

- Para eliminar un S.P. utilizamos la sentencia *drop procedure.*

```
mysql> drop procedure p;
```

Pasaje de Parámetros: ejemplo (1)

```
delimiter !
create procedure q(IN c CHAR(10), IN d DATE)
 begin
   SELECT c as dia, d as fecha;
 end; !
delimiter ;

mysql> call q('Viernes', '2014-10-10');
+---------+-------------+
| dia     | fecha       |
+---------+-------------+
| viernes| 2014-10-10  |
+---------+-------------+
```

Pasaje de Parámetros: ejemplo (2)

```
delimiter !
create procedure r(IN i INT, OUT o INT)
 begin
   if i < 0 then
     set o=i; # asignación: SET <Variable>=<expresion>;
   else
     set o=i+10;
   end if;
 end; !
delimiter ;
```

```
mysql> call r(10, @A); #@A es una variable local a la conexión
Query OK, 0 rows affected (0.00 sec)

mysql> select @A as resultado; #acceder a la variable
+-----------+
| resultado |
+-----------+
| 20        |
+-----------+
```

Declaración y uso de variables: ejemplo

```
delimiter !
create procedure porcentaje(IN C INT)
# Calcula el 10% del saldo asociado a la cuenta C
 begin
  # declaración de variables: declare <lista_variables> <tipo>;
  declare Saldo_Cuenta, P DECIMAL(7,2);
  # recupero el saldo de la cuenta C en la variable Saldo_Cuenta
  SELECT saldo INTO Saldo_Cuenta FROM cuentas WHERE numero = C;
  #OJO! SELECT ... INTO ... debe devolver una sola fila
  set P = Saldo_Cuenta * 0.1;
  SELECT C as cuenta, Saldo_Cuenta as saldo, P as diez_porciento;
 end; !
delimiter ;

mysql> call porcentaje(1);
+--------+---------+----------------+
| cuenta | saldo   | diez_porciento |
+--------+---------+----------------+
|      1 | 1000.00 |         100.00 |
+--------+---------+----------------+
```

Cursores

- Los *cursores* permiten recuperar las tuplas resultantes de una consulta *select* dentro de un S.P.
- Los cursores se declaran de la sig. forma:

```
declare <nombre_cursor> cursor for <consulta_select>
```

- Para acceder al resultado de la consulta debemos primero abrir el cursor:

```
open <nombre_cursor>
```

- Una vez abierto se puede recuperar cada tupla mediante la siguiente sentencia:

```
fetch <nombre_cursor> into <variable1>[...,<variableN>]
```

recupera la próxima tupla (si existe) utilizando el cursor especificado y avanza el puntero a la siguiente tupla.

- Una vez utilizado podemos cerrar el cursor mediante:

```
close <nombre_cursor>
```

Mas información: sección 17.2.9 del manual refman-5.0-es.a4.pdf o sección 20.2 de refman-5.6-en.a4.pdf

Cursores: ejemplo

```
create procedure inc_saldo(IN monto DECIMAL(7,2))
# incrementa el saldo en el valor de monto, para aquellas cuentas cuyo saldo <=  $2000
 begin
   # declaracion de variables
   declare fin boolean default false;
   declare nro_cuenta int;
   # declaro un cursor para la consulta que devuelve las cuentas con saldo <= $2000
   declare C cursor for select numero from cuentas where saldo <= 2000;
   # defino operacion a realizar cuando el fetch no encuentre mas filas:  set fin=true;
   declare continue handler for not found set fin = true;
   open C; # abro el cursor (ejecuta la consulta asociada)
   fetch C into nro_cuenta; # recupero la primera fila en la variable nro_cuenta
   while not fin do
         update cuentas    # actualizo el saldo de la cuenta
         set saldo = saldo + monto
         where numero = nro_cuenta;
         fetch C into nro_cuenta; # recupero la próxima fila en la variable nro_cuenta
   end while;
   close C; # cierro el cursor
 end; !
```

Cursores: ejemplo

```
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 | 1000.00 |
|      2 | 2000.00 |
|      3 | 3000.00 |
+--------+---------+
3 rows in set (0.00 sec)

mysql> call inc_saldo(500);
Query OK, 0 rows affected (0.00 sec)

mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 | 1500.00 |
|      2 | 2500.00 |
|      3 | 3000.00 |
+--------+---------+
3 rows in set (0.00 sec)
```

Triggers

- Un trigger es un objeto que se almacena en la B.D. asociado a una tabla y se activa cuando ocurre un evento en particular para esa tabla.
- Los trigger contienen sentencias que se ejecutan cuando el trigger se activa.
- Los eventos que pueden activar un trigger son:
  - Se inserta una nueva fila en la tabla (INSERT).
  - Se borra una o mas filas de la tabla (DELETE).
  - Se actualiza una o mas filas de la tabla (UPDATE).
- También es posible definir en que momento se activará un trigger, esto es, si se activará antes (BEFORE) o después (AFTER) que se produzca el evento.
- No puede haber dos triggers asociados a una misma tabla que correspondan al mismo momento y evento.

Para que usar triggers?

- Implementar reglas de consistencia de datos no provistas por el modelo relacional.
- Verificar y prevenir que se inserten datos inválidos.
- Replicación de datos.
- Auditoria: monitorear la actividad de los cambios sobre los datos.
- Acciones en cascadas.
- Autorización y seguridad.

Creación de triggers

CREATE TRIGGER <nombre del trigger>

{ BEFORE | AFTER } → *cuando se ejecuta: antes o después*

{ INSERT | UPDATE | DELETE } → *que evento lo activa*

ON <nombre de tabla> → *tabla donde se produce el evento*

FOR EACH ROW

<sentencia> → *Qué se ejecuta. Cualquier sentencia válida para un S.P., incluyendo: sentencias SQL, sentencias compuestas, llamadas a S.P., condicionales, etc. Existen* ***restricciones****, por ejemplo: dentro de un trigger no es posible modificar la tabla que activó el trigger (Ver sección Apendice H.1 de refman-5.0-es.a4.pdf sección Appendix D.1 de refman-5.6-en.a4.pdf).*

Creación de triggers: ejemplo

- Crearemos un trigger que registre la fecha y hora de todos los cambios de saldo que se producen sobre las cuentas.
- Para almacenar estos datos creamos la siguiente tabla:

```
USE bank;
CREATE TABLE movimientos(
   numero INT UNSIGNED AUTO_INCREMENT NOT NULL,
   cuenta INT UNSIGNED NOT NULL,
   fecha DATE NOT NULL,
   hora TIME NOT NULL,
   saldo_anterior DECIMAL(7,2) NOT NULL,
   saldo_posterior DECIMAL(7,2) NOT NULL,

   CONSTRAINT pk_movimentos
   PRIMARY KEY(numero),
   CONSTRAINT fk_movimentos_cuentas
   FOREIGN KEY(cuenta) REFERENCES cuentas(numero)
) ENGINE=InnoDB;
```

Creación de triggers: ejemplo

```
delimiter !
CREATE TRIGGER cuentas_update
AFTER UPDATE ON cuentas
FOR EACH ROW
BEGIN
   INSERT INTO movimientos(cuenta, fecha,
        hora,saldo_anterior, saldo_posterior)
   VALUES(OLD.numero, curdate(), curtime(),
          OLD.saldo, NEW.saldo);
END; !
delimiter ;
```

- Las **variables especiales de transición OLD y NEW** hacen referencia a los **valores de la fila** afectada **antes y después de la modificación** respectivamente.

Creación de triggers: ejemplo

- **OLD contiene los valores de la fila afectada antes de ser modificada.**
- **NEW contiene los valores de la fila afectada después de ser modificada.**
- **Si el evento es INSERT solo se puede utilizar NEW y si es DELETE solo se puede usar OLD. Con UPDATE se pueden usar ambas.**

```
mysql> select * from movimientos;
Empty set (0.00 sec)

mysql> update cuentas set saldo=5000 where numero<=2;
Query OK, 2 row affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from movimientos;
+--------+--------+------------+----------+-----------------+-----------------+
| numero | cuenta | fecha      | hora     | saldo_anterior | saldo_posterior |
+--------+--------+------------+----------+-----------------+-----------------+
|      1 |      1 |  <actual>  | <actual> |         1000.00 |         5000.00 |
|      2 |      2 |  <actual>  | <actual> |         2000.00 |         5000.00 |
+--------+--------+------------+----------+-----------------+-----------------+
2 rows in set (0.00 sec)
```

Transacciones

- Transacción: secuencia de instrucciones (sentencias SQL) relacionadas, que deben ser tratadas como una unidad indivisible.
- El uso de las B.D. Es mas seguro y eficiente:
  - Reglas ACAD (ACID): atomicidad, consistencia, aislamiento (isolation), durabilidad.
  - muchos programas y usuarios pueden acceder concurrentemente a las B.D.
  - Control de concurrencia.
  - Recuperación de fallos.

Transacciones: ejemplo 1

- Consideremos nuevamente la B.D. *bank* que almacena el saldo de cada cuenta.
- Operación habitual: transferir un monto M de una cuenta A a una cuenta B.

```
Transferir(M,A,B)
   Read(SaldoA);
   if SaldoA > M then
      SaldoA:=SaldoA-M;
       Write(SaldoA);
       Read(SaldoB);
      SaldoB:=SaldoB+M;
      write(SaldoB);.
```

Transacciones: ejemplo 1

- En una transferencia se modifican los saldos de dos cuentas. Ambos saldos deben modificarse o no debe modificarse ninguno (atomicidad).

```
Transferir(500,A,B)
 Read(SaldoA);
 if SaldoA > 500 then
    SaldoA:=SaldoA-500;
    Write(SaldoA);
   FALLA
```

Valores iniciales:

SaldoA = 1000

SaldoB = 2000

SaldoC = 3000

Valores finales:

SaldoA = 500

SaldoB = 2000

SaldoC = 3000

Perdimos $500 !

Transacciones

- Si se realizan dos transferencias concurrentemente y estas acceden a la misma cuenta, puede violarse la serializabilidad y dejar la B.D en un estado inconsistente.
- Se necesita un protocolo de control de concurrencia

Transacciones: ejemplo 2

Transferir(500,A,B)	Transferir(500,A,C)
Read(SaldoA);	
	Read(SaldoA);
if SaldoA > 500 then	
SaldoA:=SaldoA-500;	
Write(SaldoA);	
Read(SaldoB);	
SaldoB:=SaldoB+500;	
Write(SaldoB);.	
	if SaldoA > 500 then
	SaldoA:=SaldoA-500;
	Write(SaldoA);
	Read(SaldoC);
	SaldoC:=SaldoC+500;
	Write(SaldoC);.

Valores iniciales:

SaldoA = 1000

SaldoB = 2000

SaldoC = 3000

Valores finales:

SaldoA = 500

SaldoB = 2500

SaldoC = 3500

Inconsistencia !

Transacciones en MySQL

- Las transacciones están disponibles solo para las tablas de tipo InnoDB.
- Respeta las reglas ACAD (ACID).
- Protocolo de bloqueo 2F riguroso para control de concurrencia (mantiene los bloqueos exclusivos y compartidos hasta que la transacción comete).
- Bloqueo a nivel de fila de una tabla.
- Recuperación de fallos
- Prevención de Deadlocks por timeout.

Ver *variables innodb_lock_wait_timeout, slave_transaction_retries*, secciones 5.2.1 a 5.2.3 (variables de sistema) y 4.3.2 (archivos de configuración) de refman5.0-es-a4.pdf o secciones 5.1.5 a 5.1.6 (system variables) y. 4.2.6 (using options files) de refman5.6-en-a4.pdf.

Transacciones en MySQL

- Todas las sentencias SQL (select, update, insert, etc.) se ejecutan de manera atómica.
- Para ejecutar que una secuencia de sentencias SQL de manera atómica, hay que agruparlos dentro de una transacción.
- Para esto se utilizan dos sentencias SQL especiales: start transaction y commit.

```
start transaction;
sentencia_SQL_1;
      :
      :
sentencia_SQL_N;
commit;
```

Transacciones en MySQL

- La sentencia *start transaction* comienza una transacción.
- La sentencia *commit* termina la transacción y almacena todos los cambios.
- Las sentencias SQL ejecutadas entre *start transaction* y *commit* se ejecutan atómicamente.
- La sentencia *rollback* puede utilizarse para retroceder la transacción y volver al estado previo al comienzo de la misma.

Transacciones en MySQL: bloqueos

- protocolo de bloqueo riguroso: mantiene todos los bloqueos hasta que la transacción comete.
- Dentro de una transacción, las sentencias *insert, delete, update* bloquean automáticamente las filas involucradas en modo exclusivo.
- Para la sentencia *select* debemos indicar el modo en el que vamos a acceder a los datos.
- *select ... lock in share mode* bloquea las filas involucradas en la consulta en modo compartido.
- *select ... for update* bloquea las filas involucradas en la consulta en modo exclusivo.

Transacciones en MySQL: ejemplos

- Consideremos nuevamente la B.D. de un banco que mantiene el saldo de cada cuenta.

```
CREATE DATABASE bank;
USE bank;
CREATE TABLE cuentas(
    numero INT UNSIGNED NOT NULL,
    saldo DECIMAL(7,2) NOT NULL,
    PRIMARY KEY(numero)
) ENGINE=InnoDB;

INSERT INTO cuentas VALUES (1, 1000);
INSERT INTO cuentas VALUES (2, 2000);
INSERT INTO cuentas VALUES (3, 3000);
```

Transacciones en MySQL: ejemplo 1

- Transferir $500 de la cuenta 1 a la cuenta 2:

```
mysql> use bank;
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 | 1000.00 |
|      2 | 2000.00 |
|      3 | 3000.00 |
+--------+---------+
mysql> start transaction;
mysql> select saldo from cuentas where numero=1 for update;
+---------+
| saldo   |
+---------+
| 1000.00 |
+---------+
mysql> update cuentas set saldo= saldo-500 where numero=1;
```

Transacciones en MySQL: ejemplo 1 (cont.)

```
mysql> update cuentas set saldo= saldo+500 where numero=2;
mysql> commit;
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 |  500.00 |
|      2 | 2500.00 |
|      3 | 3000.00 |
+--------+---------+
```

Transacciones en MySQL: ejemplo 2

- Al Transferir $500 de la cuenta 1 a la cuenta 2 simulamos una falla cortando la conexión con *exit*:

```
mysql> use bank;
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 |  500.00 |
|      2 | 2500.00 |
|      3 | 3000.00 |
+--------+---------+
mysql> start transaction;
mysql> select saldo from cuentas where numero=1 for update;
+---------+
| saldo   |
+---------+
|  500.00 |
+---------+
```

Transacciones en MySQL: ejemplo 2 (cont.)

```
mysql> update cuentas set saldo= saldo-500 where numero=1;
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 |    0.00 |
|      2 | 3000.00 |
|      3 | 3000.00 |
+--------+---------+
mysql> exit # simulamos una falla cortando la conexión

c:\Program Files\MySQL\MySQL Server 5.0\bin> mysql –u root
...  # restablecemos la conexión
mysql> use bank;
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 |  500.00 |
|      2 | 2500.00 |
|      3 | 3000.00 |
+--------+---------+
# la transacción se deshizo automáticamente dejando la B.D.
# en el estado previo a su ejecución
```

Transacciones en MySQL: ejemplo 3

- Dos transacciones concurrentes:
  - Conexión 1: T1= transferir $500 de la cuenta 2 a la 1
  - Conexión 2: T2= transferir $500 de la cuenta 2 a la 3

Conexión 1: T1= $500 de 2 a 1	Conexión 2: T2= $500 de 2 a 3
mysql> **select** * **from** cuentas; +--------+---------+ | numero | saldo | +--------+---------+ | 1 | 500.00 | | 2 | 2500.00 | | 3 | 3000.00 | +--------+---------+	
mysql> **start transaction;**	
	mysql> **start transaction;**

Transacciones en MySQL: ejemplo 3 (cont.)

Conexión 1: T1= $500 de 2 a 1	Conexión 2: T2= $500 de 2 a 3

`mysql> select saldo from cuentas where numero=2 for update;`
`+---------+`
`| saldo |`
`+---------+`
`| 2500.00 |`
`+---------+`	

	`mysql> select saldo from cuentas where numero=2 for update;`
`mysql> update cuentas set saldo= saldo-500 where numero=2;`	 **T2 debe esperar por que T1 seleccionó la cuenta 2 para actualizar. T2 queda bloqueada hasta que T1 comete y libera el dato.**
`mysql> update cuentas set saldo= saldo+500 where numero=1;`	
`mysql> commit;`	

	`+---------+`
	`| saldo |`
	`+---------+`
	`| 2000.00 |`
	`+---------+`

Transacciones en MySQL: ejemplo 3 (cont.)

Conexión 1: T1= $500 de 2 a 1	Conexión 2: T2= $500 de 2 a 3
	`mysql> update cuentas set saldo= saldo-500 where numero=2;`
	`mysql> update cuentas set saldo= saldo+500 where numero=3;`
	`mysql> commit;`

	`mysql> select * from cuentas;`
	`+--------+---------+`
	`| numero | saldo   |`
	`+--------+---------+`
	`|      1 | 1000.00 |`
	`|      2 | 1500.00 |`
	`|      3 | 3500.00 |`
	`+--------+---------+`

Transacciones en MySQL: ejemplo 4

- Dos transacciones concurrentes:
  - Conexión 1: T1= transferir $500 de la cuenta 1 a la 3
  - Conexión 2: T2= transferir $500 de la cuenta 2 a la 3

Conexión 1: T1= $500 de 1 a 3	Conexión 2: T2= $500 de 2 a 3
mysql> **select** * **from** cuentas; +--------+---------+ | numero | saldo | +--------+---------+ | 1 | 1000.00 | | 2 | 1500.00 | | 3 | 3500.00 | +--------+---------+ 3 rows in set (0.00 sec)	
mysql> **start transaction;** Query OK, 0 rows affected ...	
	mysql> **start transaction;** Query OK, 0 rows affected ...

Transacciones en MySQL: ejemplo 4 (cont.)

Conexión 1: T1= $500 de 1 a 3	Conexión 2: T2= $500 de 2 a 3
mysql> **select** saldo **from** cuentas **where** numero=1 **for update**; +---------+ | saldo | +---------+ | 1000.00 | +---------+ 3 rows in set (0.00 sec)	
	mysql> **select** saldo **from** cuentas **where** numero=2 **for update**; +---------+ | saldo | +---------+ | 1500.00 | +---------+ 3 rows in set (0.00 sec)
mysql> **update** cuentas **set** saldo= saldo-500 **where** numero=1; Query OK,1 rows affected ...	
	mysql> **update** cuentas **set** saldo= saldo-500 **where** numero=2; Query OK,1 rows affected ...

Transacciones en MySQL: ejemplo 4 (cont.)

Conexión 1: T1= $500 de 1 a 3	Conexión 2: T2= $500 de 2 a 3
mysql> **update** cuentas **set** saldo= saldo+500 **where** numero=3; Query OK,1 rows affected ... *# bloquea implicitamente en* *# modo exclusivo la cuenta 3*	
	mysql> **update** cuentas **set** saldo= saldo+500 **where** numero=3;
mysql>commit Query OK, 0 rows affected ...	 **T2 debe esperar por que T1 actualizo la cuenta 2. T2 queda bloqueada hasta que T1 comete y libera el dato.**
	Query OK,1 rows affected ...
	mysql>commit Query OK, 0 rows affected ...
	select * **from** cuentas; +--------+---------+ | numero | saldo | +--------+---------+ | 1 | 500.00 | | 2 | 1000.00 | | 3 | 4500.00 | +--------+---------+

Transacciones y Aplicaciones

- Una aplicación puede conectarse con el servidor ejecutar la sentencia *start transaction,* luego ejecutar una secuencia de sentencias SQL que componen la transacción y por último ejecutar *commit*.
- Desventaja: cada aplicación que accede a la B.D. Puede implementar la misma transacción de diferentes formas y esto podría producir inconsistencias. Ejemplo: una aplicación podría realizar una transferencia sin controlar que el saldo de la cuenta sea suficiente.

Transacciones y Stored Procedures

- La semántica de una transacción depende de la B.D., no de las aplicaciones que acceden la B.D.
- Una transacción debe comportarse de la misma forma para todas las aplicaciones que acceden a la B.D. *Ejemplo: Una transferencia de una cuenta a otra debe realizar las mismas modificaciones si es realizada desde un cajero automático o desde una página web.*
- Una transacción se puede definir y almacenar como un procedimiento (stored procedure) en el servidor para ser invocada por las aplicaciones.

Transacciones y S.P.: ejemplo

```
CREATE PROCEDURE transferir(IN monto DECIMAL(7,2), IN cuentaA INT, IN cuentaB INT)
#  Transacción  para transferir un monto de la cuentaA  a la cuentaB
BEGIN
   DECLARE saldo_actual_cuentaA DECIMAL(7,2);
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
       BEGIN #  Si se produce una SQLEXCEPTION,  se retrocede la transacción con ROLLBACK
         SELECT 'SQLEXCEPTION!,  transacción abortada' AS resultado
         ROLLBACK;
       END;
   START TRANSACTION;
      IF EXISTS  (SELECT * FROM Cuentas WHERE numero=cuentaA) AND
         EXISTS (SELECT * FROM Cuentas WHERE numero=cuentaB)  THEN
          SELECT saldo INTO saldo_actual_cuentaA
          FROM cuentas WHERE numero = cuentaA FOR UPDATE;
           IF saldo_actual_cuentaA >= monto THEN
               UPDATE cuentas SET saldo = saldo - monto  WHERE numero=cuentaA;
               UPDATE cuentas SET saldo = saldo + monto  WHERE numero=cuentaB;
               SELECT 'La transferencia se realizo con exito' AS resultado;
           ELSE
               SELECT 'Saldo insuficiente para realizar la transferencia' AS resultado;
           END IF;
      ELSE
          SELECT 'Error: cuenta inexistente;' AS resultado;

     END IF;
     COMMIT;
END; !
```

Transacciones y S.P.: ejemplo

```
mysql> use bank;
mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 | 1000.00 |
|      2 | 2000.00 |
|      3 | 3000.00 |
+--------+---------+

mysql> call transferir(500,1,2);
+--------------------------------------+
| resultado                            |
+--------------------------------------+
| La transferencia se realizo con exito |
+--------------------------------------+

mysql> select * from cuentas;
+--------+---------+
| numero | saldo   |
+--------+---------+
|      1 |  500.00 |
|      2 | 2500.00 |
|      3 | 3000.00 |
+--------+---------+
```

Transacciones: manejo de excepciones

- Si se produce un error interno en la alguna sentencia de una transacción, esta no es retrocedida automáticamente. Hay que capturar la excepción con un *handler* y ejecutar la instrucción rollback en caso de ser necesario.
- Por ejemplo:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
 BEGIN
     SELECT 'SQLEXCEPTION!,  transacción abortada' AS resultado
     ROLLBACK;
 END;
```

- MySQL provee dos clases de códigos de error:
  - SQLSTATE: 5 caracteres, standart de SQL para los errores. SQLEXCEPTION representa cualquier valor SQLSTATE que comience con “00” “01” o “02”
  - Error code: número de 4 dígitos específico de MySQL
- Mas info:
  - http://dev.mysql.com/doc/refman/5.6/en/declare-handler.html
  - http://dev.mysql.com/doc/refman/5.6/en/error-messages-server.html

Transacciones: manejo de excepciones

- Para recuperar el SQLSTATE o error code desde un S.P. hay que utilizar la sentencia GET DIAGNOSTICS (solo disponible a partir de la versión 5.6).
- Por ejemplo:

```
DECLARE codigo_SQL CHAR(5) DEFAULT '00000';
DECLARE codigo_MYSQL INT DEFAULT 0;
DECLARE mensaje_error TEXT;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
 BEGIN
  GET DIAGNOSTICS CONDITION 1  codigo_MYSQL= MYSQL_ERRNO,
                               codigo_SQL= RETURNED_SQLSTATE,
                               mensaje_error= MESSAGE_TEXT;
   SELECT 'SQLEXCEPTION!, transacción abortada' AS resultado,
                codigo_MySQL, codigo_SQL,  mensaje_error;
   ROLLBACK;
 END;
```

- Mas info: http://dev.mysql.com/doc/refman/5.6/en/get-diagnostics.html