



Stored Procedures in MySQL & Transactions in C#

Peter Demeester

Woord vooraf

Elk databanksysteem heeft zijn eigen manier om stored procedures te maken

Wij beperken ons hier tot MySQL

Stored procedures maken in SQL Server is bijv. helemaal anders

User Defined Variables

- een *user variable* (voorafgegegaan door een @) kan eender welke scalaire waarde bevatten en persisteert gedurende de levensduur van één connectie
- je kan een variabele instellen met SET of binnen een query een waarde geven:

```
mysql> set @var = 3;  
mysql> select @diff := datediff(curdate(),  
    '2015-11-24');
```
- user variables zijn vooral handig om tussentijdse resultaten te bewaren wanneer je meerdere queries wilt uitvoeren binnen eenzelfde pagina of stored procedure.

Stored procedures

- een *stored procedure* (SPROC) is een verzameling van SQL commando's die als één geheel (een procedure) bewaard en uitgevoerd worden op de MySQL server
- met SPROC's heb je een volledige SQL-based programmeertaal ter beschikking
- merk op: SPROC's werden pas in MySQL 5.0 geïntroduceerd en staan dus nog in de kinderschoenen in vergelijking met Oracle en SQL Server waar ze al veel langer gebruikt worden
- meer info kan je vinden op <http://dev.mysql.com/doc/refman/5.7/en/stored-routines.html>

SPROC Administratie

- een overzicht van de bestaande SPROC's kan je opvragen met **SHOW PROCEDURE STATUS**
`mysql> show procedure status;`
- de code van een SPROC kan je opvragen met **SHOW CREATE PROCEDURE**
`mysql> show create procedure helloworld;`
- een SPROC kan je verwijderen met **DROP PROCEDURE**
`mysql> drop procedure if exists helloworld;`
- merk op: SPROC's worden intern in de `mysql.proc` systeemtabel bewaard

Hello World SQL style

- een SPROC wordt aangemaakt met CREATE PROCEDURE en opgeroepen met CALL
- de delimiter is het karakter of de string die je gaat gebruiken om de MySql client te vertellen dat de input werd beëindigd
- door de delimiter tijdelijk aan te passen kan een ; gebruikt worden binnen de SPROC om verschillende statements te gebruiken

```
DELIMITER $$  
DROP PROCEDURE IF EXISTS hello $$  
CREATE PROCEDURE hello()  
begin  
    SELECT 'Hello, it's me. I was wondering if after  
           all these years you'd like to meet.';  
end $$  
DELIMITER ;  
CALL hello();
```

Parameters

er bestaat verschillende mogelijkheden om parameters aan SPROCS mee te geven:

- `CREATE PROCEDURE sproc_name () ...`
- `CREATE PROCEDURE sproc_name([IN] name data-type) ...`
- `CREATE PROCEDURE sproc_name(OUT name data-type) ...`
- `CREATE PROCEDURE sproc_name(INOUT name data-type) ...`

met IN parameters kunnen waarden aan een SPROC meegegeven worden (default), via OUT parameters kunnen resultaten teruggegeven worden en INOUT parameters werken bidirectioneel

Parameters

voorbeeld van IN parameter:

```
mysql> CREATE PROCEDURE p1(p INT) SET @x = p $$  
mysql> CALL p1(12345) $$  
mysql> SELECT @x $$  
+-----+  
| @x    |  
+-----+  
| 12345 |  
+-----+
```

voorbeeld van OUT parameter:

```
mysql> CREATE PROCEDURE p2(OUT p INT) SET p = -5 $$  
mysql> CALL p2(@y) $$  
mysql> SELECT @y $$  
+-----+  
| @y    |  
+-----+  
|    -5 |  
+-----+
```


BEGIN/END blocks

meerdere statements worden gegroepeerd tussen BEGIN en END (te vergelijken met { en } in Java of C#)

```
CREATE PROCEDURE p3(a INT, b INT)
BEGIN
    SET @product = a * b;
    INSERT INTO t VALUES (@product);
END $$
```

binnen BEGIN/END blocks kan je lokale variabelen of flow control gebruiken (zie verder)

blocks kunnen genest worden en geven aanleiding tot een andere scope

BEGIN/END blocks

optioneel kan je een label meegeven, dit is vooral handig als je vroegtijdig dit block wilt verlaten met **LEAVE**

```
blockname: BEGIN
  commands;
  IF condition THEN LEAVE blockname; END IF;
  further commands;
END blockname;
```

Lokale variabelen

met DECLARE kunnen binnen een SPROC variabelen aangemaakt worden

```
CREATE PROCEDURE p4 ()  
BEGIN  
    DECLARE a, b INT;  
    DECLARE name VARCHAR(25) ;  
END $$
```

merk op dat lokale variabelen (in tegenstelling tot user defined variabels) niet met een @ beginnen
DECLARE statements moeten aan het begin van het BEGIN/END block komen

je kan variabelen een waarde geven met SET of met SELECT INTO (zie later voor een voorbeeld)

IF-THEN-ELSE

binnen een BEGIN/END block kunnen conditionals gebruikt worden:

```
CREATE PROCEDURE p5(IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;
  IF variable1 = 0 THEN
    INSERT INTO t VALUES(17);
  END IF;
  IF parameter1 = 0 THEN
    UPDATE t SET s1 = s1 + 1;
  ELSE
    UPDATE t SET s1 = s1 + 2;
  END IF;
END $$
```

CASE

alternatieve conditional is een (switch) CASE structuur:

```
CREATE PROCEDURE p6(IN parameter1 INT)
BEGIN
  DECLARE variabele1 INT;
  SET variabele1 = parameter1 + 1;
  CASE variabele1
    WHEN 0 THEN INSERT INTO t VALUES (17);
    WHEN 1 THEN INSERT INTO t VALUES (18);
    ELSE INSERT INTO t VALUES (19);
  END CASE;
END $$
```

welk resultaat levert CALL p6(NULL) op?

Loops

naast *conditionals* kunnen ook *loops* gebruikt worden

loops komen in 3 smaken:

```
[loopname:] REPEAT
  commands;
UNTIL condition END REPEAT [loopname];

[loopname:] WHILE condition DO
  commands;
END WHILE [loopname];

loopname: LOOP
  commands;
END LOOP loopname;
```

m.b.v. **LEAVE** kan een lus onderbroken worden (noodzakelijk bij **LOOP**), met **ITERATE** kan je een lusdoorgang overslaan

WHILE

```
CREATE PROCEDURE p7()  
BEGIN  
    DECLARE v INT;  
    SET v = 0;  
    WHILE v < 5 DO      -- klassieke while  
        INSERT INTO t VALUES (v);  
        SET v = v + 1;  
    END WHILE;  
END $$
```

```
CREATE PROCEDURE p8()  
BEGIN  
    DECLARE v INT;  
    SET v = 0;  
    REPEAT              -- do ... while  
        INSERT INTO t VALUES (v);  
        SET v = v + 1;  
    UNTIL v >= 5 END REPEAT;  
END $$
```

LOOP

```
CREATE PROCEDURE p9()  
BEGIN  
    DECLARE v INT;  
    SET v = 0;  
    loop_label: LOOP -- leave is te vergelijken met break  
        INSERT INTO t VALUES (v);  
        SET v = v + 1;  
        IF v >= 5 THEN  
            LEAVE loop_label;  
        END IF;  
    END LOOP;  
END $$
```


LOOP

```
CREATE PROCEDURE p10()  
BEGIN  
    DECLARE v INT;  
    SET v = 0;  
    loop_label: LOOP -- iterate is te vergelijken met continue  
        IF v < 3 THEN  
            SET v = v + 1;  
            ITERATE loop_label;  
        END IF;  
        INSERT INTO t VALUES (v);  
        SET v = v + 1;  
        IF v >= 5 THEN  
            LEAVE loop_label;  
        END IF;  
    END LOOP;  
END $$
```

Error handling

om fouten binnen de SPROC zelf op te vangen kan je handlers voorzien: een HANDLER is een stuk code die getriggered wordt naar aanleiding van een bepaalde fout

er bestaan EXIT en CONTINUE handlers:

- een exit handler zal het omsluitende BEGIN/END block verlaten,
- na een continue handler wordt de uitvoer vervolgd

```
DECLARE
{EXIT | CONTINUE}
HANDLER FOR
{error-number | SQLSTATE error-string | condition}
SQL statement
```

MySQL errorcodes zijn te vinden op

<http://dev.mysql.com/doc/refman/5.7/en/error-handling.html>

Error handling

aangezien handlers met DECLARE worden aangemaakt moeten ze in het begin van het BEGIN/END block komen

Voorbeeld: Wat is de waarde van @x na een SQLSTATE 23000 = PK of FK violation?

```
CREATE TABLE t1 (s1) $$
CREATE TABLE t2 (s1) $$
BEGIN
  DECLARE CONTINUE HANDLER
  FOR SQLSTATE '23000' SET @x = 2;
  SET @x = 1;
  INSERT INTO t2 VALUES (1);
  SET @x = 2;
  -- dubbele waarde voor PK: continue handler wordt uitgevoerd
  INSERT INTO t2 VALUES (1);
  -- na handler gaat uitvoer hier verder
  SET @x = 3;
END $$
```

Warning vs error

A guy is standing on the corner of the street smoking one cigarette after another. A lady walking by notices him and says

"Hey, don't you know that those things can kill you? I mean, didn't you see the giant warning on the box?!"

"That's OK" says the guy, puffing casually "I'm a computer programmer"

"So? What's that got to do with anything?"

"We don't care about warnings. We only care about errors."

Cursors

met een cursor kan je record per record door een resultset stappen

cursors in MySQL zijn read-only en forward-only

```
DECLARE cursor-name CURSOR FOR SELECT ...;  
OPEN cursor-name;  
FETCH cursor-name INTO variable [, variable];  
CLOSE cursor-name;
```

Cursors

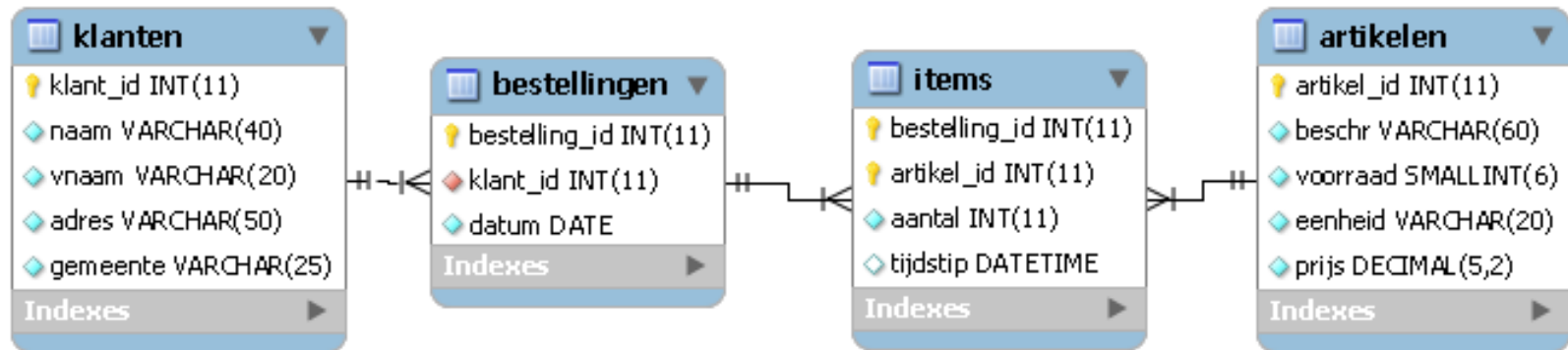
```
CREATE PROCEDURE p12()  
BEGIN  
    DECLARE done INT DEFAULT 0;  
    DECLARE a CHAR(16);  
    DECLARE b, c INT;  
    DECLARE cur1 CURSOR FOR SELECT id, data FROM test.t1;  
    DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;  
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;  
  
    OPEN cur1; OPEN cur2;  
  
    REPEAT  
        FETCH cur1 INTO a, b;  
        FETCH cur2 INTO c;  
        IF NOT done THEN  
            IF b < c THEN INSERT INTO test.t3 VALUES (a,b);  
            ELSE INSERT INTO test.t3 VALUES (a,c);  
            END IF;  
        END IF;  
    UNTIL done END REPEAT;  
  
    CLOSE cur1; CLOSE cur2;  
END
```

Tijdelijke tabellen

- om tijdelijk resultaten te bewaren in een tabel kan je een *temporary table* aanmaken m.b.v. het **TEMPORARY** sleutelwoord
- een temporary table wordt automatisch verwijderd van zodra de connectie wordt gesloten
- voor tijdelijke tabellen wordt meestal het tabel-type **HEAP** gebruikt, dergelijke tabellen worden enkel in RAM geheugen bewaard (sneller), niet op de harde schijf
- kunnen met **DROP TABLE** manueel op dezelfde manier verwijderd worden als 'gewone' tabellen van het type **MyISAM** of **InnoDB**

VOORBEELDEN

De *postorder* database

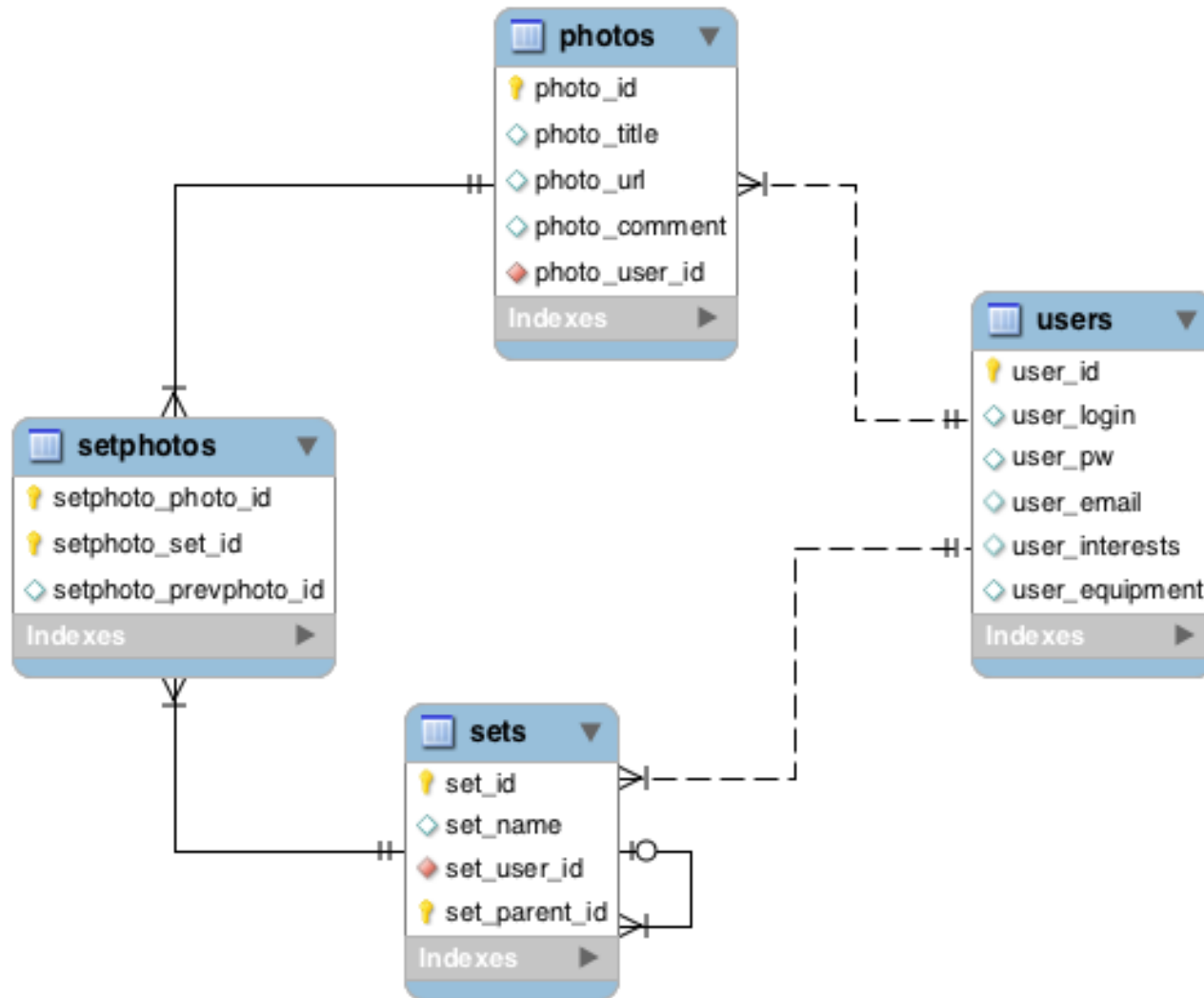


Voorbeeld 1

- schrijf een SPROC waarmee de waarden voor datum en tijdstip in de postorder geactualiseerd worden naar vandaag

```
CREATE PROCEDURE perform_updates()  
BEGIN  
    DECLARE maximum DATE;  
    DECLARE diff int;  -- aantal dagen verschil  
    SELECT MAX(datum) FROM bestellingen INTO maximum;  
    SELECT TO_DAYS(CURDATE()) - TO_DAYS(maximum) INTO diff;  
    -- tel zowel bij datum als tijd dat aantal dagen bij  
    UPDATE bestellingen SET datum = ADDDATE(datum, INTERVAL diff DAY);  
    UPDATE items SET tijdstip = ADDDATE(tijdstip, INTERVAL diff DAY);  
END $$
```

De *fotofactory* database



Voorbeeld 2

schrijf een SPROC die een nieuwe set toevoegt aan de sets tabel, gegeven de naam van de nieuwe set en de id van de parent, het resultaat is de id van de nieuw aangemaakte set

```
mysql> select * from sets;
```

set_id	set_name	set_user_id	set_parent_id
1	Asia	36	NULL
2	Africa	36	NULL
3	Europe	36	NULL
4	Bosnia	36	3
5	Botswana	36	2
6	Morocco	36	2
7	Russia	36	1
8	Cambodia	36	1
9	Phnom Penh	36	8

Voorbeeld 2

```
CREATE PROCEDURE sets_insert(IN newsetname VARCHAR(60),
                             IN parent INT, OUT newid INT)
proc: BEGIN
  DECLARE cnt, userid INT;
  SET newid = -1;

  -- basic validation: check if parent exists and
  -- newsetname is valid
  SELECT COUNT(*) FROM sets WHERE set_id = parent INTO cnt;
  IF ISNULL(newsetname) OR TRIM(newsetname)="" OR cnt=0 THEN
    LEAVE proc;
  END IF;

  -- test if set already exists
  SELECT COUNT(*) FROM sets
  WHERE set_parent_id=parent AND set_name=newsetname
  INTO cnt;
```

Voorbeeld 2

```
IF cnt=1 THEN
    SELECT set_id FROM sets
    WHERE set_parent_id=parent AND set_name=newsetname
    INTO newid;
    LEAVE proc;
END IF;

-- select user_id from parent set
SELECT set_user_id FROM sets WHERE set_id=parent INTO
userid;

-- insert new set
INSERT INTO sets (set_name, set_parent_id, set_user_id)
VALUES (newsetname, parent, userid);
SET newid = LAST_INSERT_ID();
END proc $$
```

LAST_INSERT_ID() (with no argument) returns a BIGINT (64-bit) value representing the first automatically generated value successfully inserted for an AUTO_INCREMENT column as a result of the most recently executed INSERT statement. The value of LAST_INSERT_ID() remains unchanged if no rows are successfully inserted.

Voorbeeld 3

als derde voorbeeld bekijken we een sproc die van een gegeven set al z'n *parent sets* teruggeeft
bv. Als set_id 9 meegegeven wordt; genereert de sproc de volgende lijst:

set_id	set_name
1	Asia
8	Cambodia
9	Phnom Penh

Voorbeeld 3

```
DELIMITER $$
DROP PROCEDURE IF EXISTS get_parent_sets $$
CREATE PROCEDURE get_parent_sets(startID INT)
BEGIN
    DECLARE i, id, pid, cnt INT DEFAULT 0;
    DECLARE sname VARCHAR(60);

    DROP TABLE IF EXISTS __parent_sets;
    CREATE TEMPORARY TABLE __parent_sets
        (level INT, set_id INT, set_name VARCHAR(60)) ENGINE = HEAP;

    main: BEGIN
        -- test if startid is OK
        SELECT COUNT(*) FROM sets WHERE set_id=startID INTO cnt;
        IF cnt=0 THEN LEAVE main; END IF;

        -- insert start set into new table
        SELECT set_id, set_parent_id, set_name
        FROM sets WHERE set_id=startID
        INTO id, pid, sname;
        INSERT INTO __parent_sets VALUES(i, id, sname);
```


Voorbeeld 3

```
-- loop until root of sets is reached
parentloop: WHILE NOT ISNULL(pid) DO
    SET i=i+1;
    SELECT set_id, set_parent_id, set_name
    FROM sets WHERE set_id=pid
    INTO id, pid, sname;
    INSERT INTO __parent_sets VALUES(i, id, sname);
END WHILE parentloop;
END main;

SELECT set_id, set_name FROM __parent_sets ORDER BY level DESC;
DROP TABLE __parent_sets;
END $$
DELIMITER ;

-- example use
CALL get_parent_sets(9);
```

Voorbeeld 4

in MySQL kan je ook *stored functions* schrijven

gelijkaardig aan SP's, behalve dat

- ze steeds een resultaat teruggeven en
- het gebruik van IN en OUT parameters is niet mogelijk

bij wijze van illustratie schrijven we een functie *shorten* die strings afkort tot een gegeven lengte

de string *A Programmer's Introduction to PHP*

zou er bv. op lengte 20 als *A Programm ... o PHP* uitkomen

Voorbeeld 4

```
DELIMITER $$
CREATE FUNCTION shorten(s VARCHAR(255), n INT)
    RETURNS VARCHAR(255)
BEGIN
    IF ISNULL(s) THEN
        RETURN '';
    ELSEIF n < 15 THEN
        RETURN LEFT(s, n);
    ELSE
        IF CHAR_LENGTH(s) <= n THEN
            RETURN s;
        ELSE
            RETURN CONCAT(LEFT(s, n-10), ' ... ', RIGHT(s, 5));
        END IF;
    END IF;
END $$
DELIMITER ;

-- example use
SELECT shorten(title, 20) FROM titles LIMIT 10;
```

SP's gebruiken in jouw applicaties?

- met deze presentatie hebben we een staalkaart gegeven van de mogelijkheden die MySQL biedt om *program logic* te vervangen door *SQL logic*, met als orgelpunt SP's
- er bestaat echter geen *one-size-fits-all* antwoord op de vraag of en wanneer je SP's moet gebruiken in je eigen applicaties
- er bestaat een heel levendige discussie over dit onderwerp, maar uiteindelijk komt het neer op een keuze maken op basis van de context en de objectieve voor- en nadelen van SP's

Pro: database security

- SP's vormen een extra abstractielaag tussen de onderliggende data-laag en de *business logic* in de *middle tier*
- door de applicatie geen directe toegang te verlenen tot de tabellen en enkel via SP's te werken, kan je sterk reguleren hoe de data gebruikt wordt, en bijgevolg de veiligheid van de data aanzienlijk verhogen
- applicaties waarin de veiligheid van de data zeer belangrijk is (bv. financiële applicaties) maken hierom zeer vaak gebruik van SP's

Pro: scheiding van data en business logic

- bij *large scale* applicaties wordt typisch gewerkt met afzonderlijke ontwikkelaars voor de database enerzijds, en voor de applicatie anderzijds
- door met SP's te werken kunnen deze twee werelden beter gescheiden worden en kan elke ontwikkelaar zich concentreren op z'n eigen specialiteit: een C# wizard is niet noodzakelijk een databasespecialist, en omgekeerd
- doordat SP's een API vormen tussen de database en de applicatie, zijn bv. wijzigingen in het onderliggende databaseschema gemakkelijker door te voeren zonder al te grote gevolgen voor de applicatiecode

Pro: minder network traffic

- met SP's kunnen meerdere SQL opdrachten in één database *round trip* uitgevoerd worden, wat kan resulteren in aanzienlijk minder network traffic
- let wel: in het geval van zeer reken-intensieve operaties (bv. complexe string manipulaties) vervalt dit voordeel snel, aangezien MySQL hiervoor (nog) niet geoptimaliseerd is (terwijl bv. C# wel)
- SP's zijn vooral interessant wanneer een relatief kleine resultaatset gegenereerd wordt op basis van zeer grote tabellen, MySQL is immers wel geoptimaliseerd voor *set processing*

Con: meer abstractie, meer complexiteit

- SP's kunnen leiden tot een grotere fragmentatie van de applicatiecode wanneer de logica onzorgvuldig verdeeld wordt over de database server en de applicatie
- sowieso: deze extra abstractie is moeilijker te ontwerpen en te debuggen!
- efficiënte SP's schrijven is een extra skill die niet altijd aanwezig is bij de ontwikkelaars van een applicatie
- in het geval van kleine tot middelgrote applicaties is het wellicht beter om je aandacht volledig te richten op goede C# code schrijven

TRANSACTIONAL IN C#

Transacties?

Eenheid van werk die in zijn geheel moet uitgevoerd worden

Denk maar terug aan overschrijvingsvoorbeeld uit vak Relational Databases

Moet voldoen aan ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

Transacties

- Transactie begint als je aan databank vertelt dat je er één nodig hebt en
- Eindigt ofwel met een
 - Commit
 - Rollback

Lokale transacties

- Klasse `System.Data.Common.DBTransaction`
- Voor `SqlServer` wordt dit `System.Data.Common.SqlTransaction`
- Volgende voorbeelden zijn `SqlServer` voorbeelden
 - Voorbeelden voor andere databanken zijn analoog

Lokale transacties

Gebruik

- Connectie openen (met SqlConnection)
- SqlTransaction instantie creëren op die connectie
- Schrijf queries (in de context van een transactie)
- Commit of doe een rollback
- Sluit de connectie met de databank

Transactie beginnen

```
using (SqlConnection conn = new
        SqlConnection(connectionString)){
    conn.Open();
    SqlTransaction envelope = conn.BeginTransaction();
    ...
}
```

Voeg queries toe aan transactie

```
// Include the transaction in the SqlCommand constructor.  
SqlCommand updateCommand = new  
    SqlCommand(sqlText, conn, envelope);
```

```
// Or add it to an existing SqlCommand object.  
SqlCommand updateCommand = new  
    SqlCommand(sqlText, conn);  
updateCommand.Transaction = envelope;
```

Commit of rollback

```
// Commit the transaction.
```

```
envelope.Commit();
```

```
// Rollback the transaction.
```

```
envelope.Rollback();
```

Steeds commit of rollback aanroepen

Indien niet

- Kans dat garbagecollector automatisch achter de schermen een rollback doet!

Exception handling

Commit & Rollback & BeginTransaction

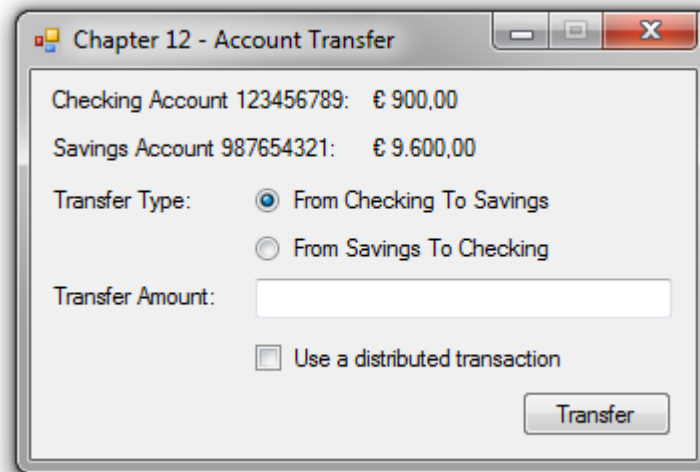
- Kunnen exceptions genereren
- Opvangen!

Exception handling

```
try{
    envelope.Commit();
}
catch (Exception ex){
    MessageBox.Show("Error saving data: " + ex.Message);
    try{
        envelope.Rollback();
    }
    catch (Exception ex2){
        // Although the rollback generated an error, the
        // transaction will still be rolled back by the
        // database because it did not get a commit order.
        MessageBox.Show("Error undoing the changes: " + ex2.Message);
    }
}
```

Demo

Banktransfer



Chapter 12 - Account Transfer

Checking Account 123456789: € 900,00

Savings Account 987654321: € 9.600,00

Transfer Type: ☒ From Checking To Savings
☐ From Savings To Checking

Transfer Amount:

☐ Use a distributed transaction

Transfer

Bijhorende code

```
public string GetConnectionString() {  
    // Build a connection string for the active database.  
    SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();  
    builder.DataSource = @"(localdb)\mssqllocaldb";  
    builder.InitialCatalog = "StepSample";  
    builder.IntegratedSecurity = true;  
    return builder.ConnectionString;  
}
```

Bijhorende code

```
private void AccountTransfer_Load(System.Object sender,  
                                System.EventArgs e) {  
    //Load in the account balances.  
    RefreshBalances();  
}  
  
private void RefreshBalances() {  
    //Prepare the form.  
    decimal result;  
    string sqlText;  
    SqlCommand accountCommand;  
    using (SqlConnection linkToDB = new  
        SqlConnection(GetConnectionString())) {  
        // Open the database.  
        try {  
            linkToDB.Open();  
        }  
    }
```

Bijhorende code

```
catch (Exception ex) {
    MessageBox.Show("Error accessing the database: " +
        ex.Message);
    return;
}
// Build a statement to get the balances.
sqlText = "SELECT Balance FROM BankAccount
           WHERE AccountNumber = @ID";
accountCommand = new SqlCommand(sqlText, linkToDB);
// Get the checking account value.
try {
    accountCommand.Parameters.AddWithValue("@ID",
                                         CheckingAccountID);
    result = (decimal)accountCommand.ExecuteScalar();
    CheckingBalance.Text = string.Format("{0:c}", result);
}
```

Bijhorende code

```
catch (Exception ex) {
    MessageBox.Show("Error retrieving checking account
        balance: " + ex.Message);
    return;
}
// Get the savings account value.
try {
    accountCommand.Parameters["@ID"].Value = SavingsAccountID;
    result = (decimal)accountCommand.ExecuteScalar();
    SavingsBalance.Text = string.Format("{0:c}", result);
}
catch (Exception ex) {
    MessageBox.Show("Error retrieving savings account
        balance: " + ex.Message);
    return;
}
}
```

Bijhorende code

```
private bool TransferLocal() {  
    // Transfer money using a local transaction.  
    string sqlText;  
    decimal toTransfer;  
    SqlCommand withdrawal;  
    SqlCommand deposit;  
    SqlTransaction envelope;  
    // Retrieve the transfer amount.  
    toTransfer = decimal.Parse(TransferAmount.Text);  
    using (SqlConnection linkToDB = new  
        SqlConnection(GetConnectionString())) {  
        try {  
            // The database must be opened to create the transaction.  
            linkToDB.Open();
```


Bijhorende code

```
// Prepare a transaction to surround the transfer.
envelope = linkToDB.BeginTransaction();
}
catch (Exception ex) {
    MessageBox.Show("Error accessing the database: " +
                    ex.Message);
    return false;
}
// Prepare and perform the withdrawal.
sqlText = @"UPDATE BankAccount SET Balance = Balance -
           @ToTransfer WHERE AccountNumber = @FromAccount";
withdrawal = new SqlCommand(sqlText, linkToDB, envelope);
withdrawal.Parameters.AddWithValue("@ToTransfer", toTransfer);
```

Bijhorende code

```
if (OptFromChecking.Checked)
    withdrawal.Parameters.AddWithValue("@FromAccount", CheckingAccountID);
else
    withdrawal.Parameters.AddWithValue("@FromAccount", SavingsAccountID);
// Prepare and perform the deposit.
sqlText = @"UPDATE BankAccount SET Balance = Balance + @ToTransfer
            WHERE AccountNumber = @ToAccount";
deposit = new SqlCommand(sqlText, linkToDB, envelope);
deposit.Parameters.AddWithValue("@ToTransfer", toTransfer);
if (OptFromChecking.Checked)
    deposit.Parameters.AddWithValue("@ToAccount", SavingsAccountID);
else
    deposit.Parameters.AddWithValue("@ToAccount", CheckingAccountID);
```

Bijhorende code

```
// Perform the transfer.  
try {  
    withdrawal.ExecuteNonQuery();  
    deposit.ExecuteNonQuery();  
    envelope.Commit();  
}  
catch (Exception ex){  
    MessageBox.Show("Error transferring funds: " +  
                    ex.Message);  
    // Do a rollback instead.
```

Bijhorende code

```
        try {
            envelope.Rollback();
        }
        catch (Exception ex2) {
            MessageBox.Show("Error cancelling the transaction: " +
                            ex2.Message);
        }
        return false;
    }
}

// ----- Successful transaction.
return true;
}
```

