MySQL 5.0
Stored Procedures, Views and Triggers

Jan Kneschke
Consulting, MySQL AB
jan@mysql.com

MySQL UC 2005
Santa Clara

Overview

- Stored Procedures
  - Purpose
  - Syntax
  - Building Trees
  - Recursion
  - Locking
- Trigger
  - Pre/Post Trigger
  - Automatic Maintenance
- Views
  - Comparison to SELECT
  - Restricting Permissions
  - Updatability
  - Algorithms

Who is this guy ?

- Jan Kneschke
  - Lives in Kiel, Germany at the Baltic Sea
- Consultant and Trainer in the Germany and Europe
- In the future he will be part of the Merlin team
- Develops a fast, low-resource and flexible webserver called `lighttpd` (called `lighty`) in his spare time

Questionnaire

- Who is using MySQL ... ok, scratch that. :-)
- Who is using a RDBMS which supports Stored Procedures, Triggers and so on ?
  - Oracle, Sybase, DB/2, MS SQL, ...
- Who is using MySQL 5.0 or newer ?
- Who is using a older version of MySQL ?
  - 3.23.x
  - 4.0.x
  - 4.1.x
- Which programming language are you using to access MySQL ?
  - Java, PHP, Perl, C#, C/C++, Ruby

New Features in MySQL 5.0

- **Stored Procedures**
- **Cursors**
- **Trigger**
- **Views**
- Greedy Optimizer
- High Precision Math
- Improved Cluster Support
- Traditional Mode (true VARCHAR, no truncation, error on division by zero, ...)
- Compact storage for InnoDB

Stored Procedures

- Handling Business Logic in at database level
  - Transactional
  - Handling complex queries directly in the database
- Reduced network traffic
  - Transfer only the result back to the client
  - Faster response time
- Application Code in the database
  - Check access to the data more strictly

Usage Scenarios

- Recursion
  - Recursion is not possible with standard SQL-DML-Statements
  - Classic Example: Tree
  - Requires several queries to walk the tree once
  - Goal: reducing network traffic
- Business Logic
  - Handling the full access to the storage via strict checking of input
  - Goal: abstracting the access to the data

Managing Stored Procedures

- Managing Stored Procedures
  - `CREATE PROCEDURE <proc_name>`
  - `DROP PROCEDURE <proc_name>`
  - `SHOW PROCEDURE STATUS`
  - `SHOW CREATE PROCEDURE <proc_name>`
- Using Stored Procedures
  - `CALL <proc_name>`

User Defined Functions

- Either external written in C
  - Since MySQL 4.1
  - Can access everything at system
  - Can't access anything in the database
- Or internal using the syntax of the Stored Procedures
  - Can be part of a SQL-Query
  - Can access table-data
  - Can altered without using a compiler

```
CREATE FUNCTION <func_name> ( ... )
 RETURNS ...
```

Syntax

- Follows the SQL:2003 standard, same as the one used by DB/2 from IBM
  - Every other DB-vendor has invented its own language
- Prototypes
- Variables
- Flow-Control
- Loops
- Error-Handling

Prototype

- Functions only have INPUT parameters and return a single values
- Procedures don't return any data directly, but can have INPUT, OUTPUT and IN/OUTPUT parameter

```
DELIMITER $$
CREATE PROCEDURE sp_create_user
   (IN uname VARCHAR(64))
BEGIN
   INSERT INTO user (username)
     VALUES (uname);
END$$
DELIMITER ;
```

MySQL-cli and Stored Procedures

- Stored Procedures use the semicolon as terminator for a statement
- The mysql command line client is using the semicolon as delimiter of the full SQL statement
- Switching the delimiter in the mysql-cli to something that is not used the Stored Procedure syntax like `$$` solves this

Variables

- SPs can access system variables, parameters, field names and local variables
- All are in the same namespace
- Local variables can take any SQL-datatype

```
CREATE PROCEDURE sp_ins_area(IN r DOUBLE)
BEGIN
  DECLARE d_area DOUBLE;
  SET d_area = 2 * PI() * r;
  INSERT INTO circle (area)
    VALUES (d_area);
END$$
```

Flow-Control

- Changing the flow of the execution of statements
- Conditional execution
  - IF ... THEN ... ELSE ... END IF
  - CASE
- Breaking out of loops
  - LEAVE, ITERATE

```
IF id < 10 THEN
  SELECT * FROM mysql.user;
END IF
```

Cursors

- LOOPs often involve a cursor
- Cursors are pointers to a result-set
- Cursors are declared at the start of the SP and used afterwards

```
DECLARE username VARCHAR(64);
DECLARE cur CURSOR FOR
    SELECT user FROM mysql.user;

OPEN cur;
FETCH cur INTO username;
CLOSE cur;
```

Loops

- LOOP ... END LOOP, REPEAT ... UNTIL ... END REPEAT, WHILE ... END WHILE
- No FOR loop yet

```
DECLARE n INT DEFAULT 0;
loop1: LOOP
  IF n < 10 THEN
    LEAVE loop1;
  END IF;
  SET n = n + 1;
END LOOP loop1;
```

Exception Handler

- Defining what should happen if a statement fails
- Handles warnings, errors and exceptions
- Possible actions: CONTINUE and EXIT

```
DECLARE CONTINUE HANDLER
   FOR SQLSTATE '23000' ...;
DECLARE CONTINUE HANDLER
   FOR NOT FOUND ...;
DECLARE CONTINUE HANDLER
   FOR SQLWARNING ...;
```

Functions

- Functions are used to return a value in SQL statements
- They can include SQL statements

```
CREATE FUNCTION sf_count_users()
  RETURNS INT
BEGIN
  DECLARE c INT;
  SELECT count(*) FROM mysql.user INTO c;
  RETURN c;
END$$
SELECT sf_count_users();
```

Trigger

- Is executed before or after a writing SQL statement has been executed
- Triggers are bound to a existing table
- They can modify the data that will be written to the table
- Used to generate sequences, maintaining 'hidden' data fields,
- Triggering external source via UDF

Example

- Reversing email-addresses for better indexing
- NEW and OLD reference the new or old row

```
CREATE TRIGGER tbi_email_reverse_address
 BEFORE INSERT ON email FOR EACH ROW
BEGIN
   IF NEW.address IS NOT NULL THEN
     SET NEW.reverse_address = REVERSE
   (NEW.address);
   END IF;
END$$
```

Misusing Triggers

- Generating the content directly in the database
- Putting the full business logic into the database
- Using a trigger to call a long running SQL statement

Invalidating external Caches

- On Demand generation of content needs trigger source to invalidate the cached content
- Using UDFs enables MySQL to execute external functions
- A trigger bound to a UPDATE calls the UDFs to tell the cache that a entry is dirty
- When the cached entry is requested the dirty entry is re-generated and put back into the cache
- Generating the content is On-Demand and asynchronous
- The content-generator might be located on a remote system

Problems with Stored Procedures

- They don't scale
- Errors are hard to find
- No debugger available yet
- They are executed in the same context as the database
- They might cause dead-locks
- They are not very portable

Right Tool for the Right Job

- Use the power of Stored Procedures and Triggers
- Don't misuse them
- They help to reduce network traffic
- Use Application Servers
- Handle the complex jobs in a Application Server in the local network
- Handle them asynchronously

Locking in SP and Triggers

- The necessary locks on referenced are calculated at creation time
- Before the SP or the Trigger is executed the LOCKs are taken first
- This was added in 5.0.3, sometimes the calculations are incomplete
- The corresponding bugs will be fixing until 5.0.x gets stable

LOCK problems

```
CREATE PROCEDURE read_only ()
BEGIN
  CALL insert_only();
  SELECT * FROM tbl1;
END$$
CREATE PROCEDURE insert_only()
BEGIN
  INSERT INTO tbl1 (field) VALUES (1);
END$$
CALL read_only()$$
```

Current State of SP + Triggers

+ Syntax is compatible to DB/2
+ They work
+ They can reference tables, VIEWs and system variables
– Can't throw errors to emulate CHECK constraints or cancel a transaction
– LOCKing problems
– Debugging is hard

VIEWs

- Limiting visibility of columns in JOINs
- Limiting visibility of rows
- Updatability of Views
- Permissions on Views compared to simple GRANTs on columns
- Hiding changed table structure for legacy applications

Managing VIEWs

- CREATE VIEW view_name AS SELECT ...
- DROP VIEW view_name
- SELECT can select base-tables and other VIEWs

```
CREATE VIEW user_email AS
   SELECT username, email
     FROM user JOIN email
          USING (user_id)
    WHERE email LIKE "%@%"
     WITH CHECK OPTION;
SELECT * FROM user_email LIMIT 1;
```

Restrictions

- Not every SELECT statement can be specified in a VIEW definition
  - No sub-queries in FROM clause
  - No system or user variables or parameters from stored routines or prepared statements
  - Referred table must exist
  - No triggers on VIEWs
- ORDER BY is ignored if the calling SELECT supplies its own

Algorithm

- At creation-time a algorithm for the VIEWs can be specified
  - MERGE
  - TEMPTABLE
  - UNDEFINED
- MERGE is more efficient, allows updatability
- TEMPTABLE copies result to a temporary table and releases to LOCKs as soon as possible
- UNDEFINED lets MySQL choose the best algorithm

MERGE Algorithm

- Merges SELECT in the view and the outer SELECT and executes the merged SELECT

```
CREATE VIEW v AS
   SELECT name FROM user
   WHERE user_id < 10;
SELECT * FROM v WHERE user_id > 5;
# becomes
SELECT name FROM user
 WHERE user_id > 5 AND user_id < 10;
```

TEMPTABLE algorithm

- MERGE doesn't work for some SELECTs
  - SUM(), MIN(), MAX(), ...
  - DISTINCT
  - GROUP BY
  - HAVING
  - UNION and UNION ALL
  - Without underlying table (`SELECT 1`)
- In those cases a TEMPTABLE is used

Updatability of VIEWs

- Some VIEWs are updatable (DELETE, INSERT, UPDATE)
- Restrictions
  - ALGORITHM = MERGE
  - No JOINs, No Subqueries in SELECT list
  - No updatable View in FROM clause
  - No dependent Subqueries
- Insertable if
  - No duplicate view column names
  - Only pure column names

CHECK OPTION

- Limit updatability of the base table to the WHERE condition of the view
- WITH [LOCAL | CASCADED] CHECK OPTION

```
UPDATE user_email
    SET email = 'jan@mysql.com'
 WHERE user_id = 1;
# ok
UPDATE user_email
    SET email = 'jan'
 WHERE user_id = 1;
# failure
```

Using VIEWs

- Working with complex table-schema like the INFORMATION_SCHEMA
-

Summary

- MySQL 5.0 provides the expected set of enterprise features
- VIEWs, Stored Procedures and Triggers are working as expected
- Some bugs have to be shaken out until the stable version

Questions

- Any questions left ?
- Feel free to ask me in the lobby
- Or send a mail to jan@mysql.com

More on this topic:

- UDF in MySQL