Seminar Report

SQL as a Programming Language

David Kofler
D.Kofler@student.uibk.ac.at

16 February 2013

Supervisor: Sarah Winkler

Abstract

SQL is commonly used to create relational databases and to insert, query and modify data. This thesis describes procedural extensions to SQL which allow processing of data in more complex ways and to run arbitrary code inside the database.

Contents

1 Introduction

SQL is the language used to query and to administrate the dominant family of Database Management System (DBMS)s, the relational ones. By itself, its functionality is limited to accessing and manipulating relational data in ways specified by the Structured Query Language (SQL) standard. Because of this many real-world applications use SQL solely to store and retrieve their data.

This report describes use cases for extensions to SQL which allow to run arbitrary code inside the DBMS. To keep the presentation concise the features provided by MySQL are discussed, although interesting features other relational DBMSs implement will be mentioned where appropriate.

2 History

SQL was created at IBM as part of the first relational DBMS, called System R [1]. Initially it was called SEQUEL, but this name had to be abandoned because the British aircraft company Hawker Siddeley already hold the trademark. The name was then shortened to SQL.

In the next decade, it became increasingly difficult to hire staff able to properly use the systems and to migrate data between different relational DBMSs. Since every vendor implemented them a bit differently, each system had (and still has) some features which are largely incompatible with each other. Oracle for example uses the name MINUS for the EXCEPT operator [3]. The details of handling NULL values is another source of diversion and confusion.

The SQL-86 and SQL-87 standards were adopted by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) respectively in the years 1986 and 1987. Today's most popular relational DBMSs support most of the SQL-92 standard, introduced in 1992. The SQL-1999 standard brought SQL Persistent stored modules (SQL-PSM), recursive queries and triggers, among other features. Later standards (SQL-2003, SQL-2008, SQL-2011) further clarified and extended these features. The latter four standards are split into parts where many of them are optional because they contain advanced features, for example streaming to support multimedia databases.

3 Relational databases

Relational DBMSs operate on tables containing rows. Each row consists of multiple statically-typed entries. SQL queries (to be precise, the subset called Data Manipulation Language (DML))

- select those rows,
- perform calculations on the values,
- accumulate them up,
- insert new tuples,

- modify tuples,
- delete tuples

.

Databases and tables are created, modified and deleted using the Data Definition Language (DDL). Also the commands used to create and delete stored procedures, functions and triggers and to schedule/cancel events are DML.

Since the examples in this report often use the SELECT statement it is necessary to describe it.

3.1 The SELECT statement

The SELECT statement consists of several clauses where only the first one is strictly necessary.

The SELECT clause consists of several expressions which are evaluated for each row of the final result set. It allows to rename columns, to omit some of them and to perform calculations with their values.

Using aggregate functions like COUNT (counts the number of rows in the result set or SUM (sums up all rows) it is possible to perform some kinds of reductions on the result set.

The FROM clause specifies the source of the rows to process. Joins are specified by using the JOIN operators. Also, other SELECT statements may be used in this clause.

The WHERE clause is applied on each row of the data source. If it evaluates to "0", then the row is not going to be part of the result set. The WHERE clause also allows subselects. With the ALL, ANY and EXISTS operators a powerful subset of predicate logic may be used.

The GROUP BY clause is used to partition the result set by the values of some specified columns (or expressions they are part of). If aggregate functions are used in the SELECT clause then they process each partition and produce a row for each partition.

The HAVING clause filters rows of the final result set after the GROUP BY clause has been applied. The ORDER BY clause specifies some ordering constraints on the final result set. The LIMIT clause is applied after the ordering constraints and allow to select only a slice of the result set, specified by numeric indices and offsets.

SELECT statements may be combined to bigger expressions either by nesting them, by using them in expressions or by using the UNION, INTERSECT and EXCEPT operators. They perform the set union, intersection and difference on the mentioned result sets respectively. The SQL standard also specifies recursive SELECT statements, but MySQL doesn't support these.

4 The need for procedural extensions

The system described in the previous section is, by itself, static. Its contents are determined strictly by the sequence of DDL and DML statements which were executed.

For some applications this is not enough. There may be consistency requirements on the data which are not easily verified by using the check clause [1]. Also it is inefficient to fetch large amounts of data, to marshal them into the types of the programming language used for the task and to put them back into the database.

The ability to run programs inside the DBMS would solve many of these problems. In this section, possible use cases for procedural extensions will be discussed.

4.1 Checking data

SQL-1999 introduced a concept known as trigger. A trigger is a small program which is executed whenever the specified table changes. This program would have access to the old data and the new data and be able to perform checks not easily coverable by the DBMSs built-in constraints on the table. Another application would be to keep a log of all performed actions in some tables reserved for this purpose.

Checks which involve costly operations must not defined inside a trigger because they would slow down database operations. If it is acceptable to perform such integrity checks periodically, then the event scheduler framework may be used. It executes some code at defined events (for example, at midnight).

4.2 Access restrictions and business rules

Authorization constraints on the data may be too complex for the DBMSs security model. For example, the right to change certain data may depend on the current time or some credentials which have to be provided by the user. By using stored procedures complex access restrictions and other business rules may be enforced inside the DBMS.

In many cases it is useful to don't store data completely normalized. That means that some redundancies are introduced which allow faster queries on the data. This has a drawback: if data is going to be updated these redundancies have to be kept intact. By using some routine crafted for this task to update such redundant relations data integrity may be ensured.

4.3 Domain logic

For certain applications it would be quite handy to factor out common operations (for example, the distance between two points [2]) from SQL queries. That would make them far easier to understand and hence improve correctness and maintainability.

[1] By the way, MySQL doesn't support it

[2] Nowadays, many DBMSs provide geo-spatial extensions

5 The SQL-PSM language

SQL-PSM (SQL Persistent Stored Modules) programs consist of statements which are executed in order. They are organized in blocks and delimited by semicolons. Each block may be labelled. Formatting is, as long as all tokens are separated, irrelevant to the interpreter.

The default MySQL client uses the semicolon to determine the end of the current command. This is undesirable since the semicolon is part of SQL-PSMs syntax. To work around this issue the delimiter used by the client has to be changed.

Example 5.1.

```
DELIMITER //
CREATE PROCEDURE test ()
BEGIN
  SELECT 1;
  SELECT 2;
END //
DELIMITER ;
```

The LEAVE statement is used to continue execution after the end of a block. By specifying the block name multiple levels of blocks may be left.

SQL-PSM uses the usual expression syntax used by SQL statements.

5.1 Stored procedures

Stored procedures are programs which are executed using the CALL statement. They take a number of input and output arguments. Modifications to IN arguments are not visible to the caller. Changes to OUT arguments are visible to the caller, but to the callee their initial value is NULL regardless of what was stored before calling. There exists a third argument type: the INOUT argument. Its value may be read and modified by the stored procedure. Of course, the OUT and INOUT arguments have to be variables, otherwise the stored procedure would have a hard time manipulating them and the SQL interpreter would complain. If the argument type is not specified, the argument is regarded as an input argument.

Example 5.2.

```
CALL myProcedure(@aVariable, 2, ``a string'');
```

Stored procedures must not use the RETURN statement. Instead, OUT arguments have to be used. Another possibility would be to use SELECT statements without a cursor or a INTO clause to produce a result set. Usually, if this option is used, multiple result sets are returned to the client, therefore a driver supporting this feature is required.

Stored functions belong to a database. As other database objects, by creating them they are put into the default database if there is one. As with tables, by putting a database name in front of the identifier and separating the names with a dot (called “qualifying”) the database the stored procedure shall belong to is specified. To refer to stored procedures of another database, the same syntax has to be used.

There are characteristics which may be set. These are

- a documentation comment
- whether the procedure is executed with the rights of its creator or with the caller’s rights,
- determinism (whether its output is dependent only of its input and there are no side-effects)
- which kind of SQL statements is contained.

The last two characteristics are not checked nor enforced, but incorrectly specifying the may confuse the optimizer and produce wrong results.

Example 5.3.
```
CREATE DEFINER = ‘mario‘@‘dbserver‘
  PROCEDURE increment (INOUT val)
    COMMENT ‘Increments a variable‘
    NOT DETERMINISTIC NO SQL
    SQL SECURITY DEFINER
  SET val = val + 1
```

The declaration `DEFINER = ‘mario‘@‘dbserver‘`, together with the clause SQL SECURITY DEFINER, determines that the procedure is always called with the rights of this user. The other allowed value for SQL SECURITY is CALLER.

5.2 Stored functions

Stored functions are similar to stored procedures. A return type has to be declared and all syntactically possible execution paths have to end with a RETURN statement. However, it is not allowed to begin or end transactions, to pass OUT or INOUT parameters. Also, the class of allowed SQL statements is severely restricted. SELECT queries have to be used with a cursor or with a INTO clause and it is not allowed to call stored procedures which produce a result set.

Stored functions are used inside expressions just like the DBMSs built-in features. Use qualified names to access other databases’ functions.

Stored functions support the same characteristics and declarations as stored procedures.

Example 5.4.

```
CREATE FUNCTION distance (x1, y1, x2, y2)
    COMMENT 'Calculates the distance between two points'
    DETERMINISTIC NO SQL
  RETURN sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2))
```

5.3 Events

The event scheduler may be used to execute a piece of SQL-PSM code at some point in the future. It may be specified when and how often the code shall be executed, together with the time intervals between the events.

Example 5.5.

```
CREATE EVENT logCurrentTime
  ON SCHEDULE EVERY 1 SECOND
    STARTS CURRENT_TIMESTAMP + 1 HOUR
    ENDS CURRENT_TIMESTAMP + 2 DAY
  DO BEGIN
    INSERT INTO times (time, what)
      VALUES (CURRENT_TIMESTAMP, 'Nothing');
  END
```

Since events return no value it is not allowed to use the RETURN statement.

5.4 Triggers

Triggers execute a piece of code whenever a table is modified. There are three kinds of modifications a trigger reacts to:

- INSERT (whenever a new row is inserted into the table)
- UPDATE (whenever a row gets changed)
- DELETE (whenever a row gets deleted)

Attention must be paid to the fact that these events don't correspond with the similarly named SQL statements. For example, the LOAD DATA and REPLACE statements cause INSERT triggers to execute. On the other hand, the DROP TABLE statement doesn't trigger DELETE triggers because it uses another mechanism.

For each table, there is at most one trigger per event type and activation time. For example, there may be only one BEFORE UPDATE trigger. This is rarely a limitation though as long as the code is known.

Triggers may access the contents of the old and new row, as long as there is an old or new row. This is done using the syntax `OLD.<column>` and `NEW.<column>`.

In a BEFORE UPDATE trigger, the contents of the new row may be changed using the SET statement.

A BEFORE trigger is executed by the attempt to modify a table. If it succeeds then the modification is performed. Afterwards, the AFTER trigger is executed. If one of the triggers fails, then the outcome depends whether there is an active transactions. If there is one, a ROLLBACK is performed. Otherwise the changes performed so far by triggers and statements are kept.

Triggers may not begin or commit transactions. Also, it is not allowed to produce result sets or to call stored procedures which produce some.

5.5 Variable declarations

In SQL-PSM variables are declared at the beginning of the block they are visible in. If there are multiple variable declarations which refer to the same name, the innermost declaration is in effect.

Variables are statically typed and may be initialized with a default value. The expression used to initialize the variable is evaluated as the block is entered.

Since there are no object-relational features in MySQL, only basic SQL data types are allowed. Or course that makes many common tasks usually solved using data structures way more complicated than they need to be.

To change the value of variable the SET statement has to be used. Many variables may be changed simultaneously. The exact order of assignments is irrelevant since all right-hand sides are evaluated before performing the assignment.

For an example see 5.6.3.

5.6 Control flow statements

Conditional expressions are ordinary expressions. Their result is interpreted as FALSE if it is equal to “0” or NULL.

5.6.1 The IF statement

The IF statement is used to redirect control flow according to a conditional expression. To reduce clutter on checking multiple conditional expressions, many of them may be checked. Finally, an ELSE branch may be specified which is executed if all conditional expression evaluated to “0”.

Example 5.6.

```
IF a > b THEN
  SET a = a - b;
ELSEIF b > a THEN
  SET b = b - a;
ELSE
  RETURN a;
END IF
```

5.6.2 The unconditional loop

The unconditional loop (the LOOP statement) simply executes the contained statements infinitely. To terminate it a LEAVE statement has to be used. It is useful to process cursors or if the loop condition is too complex for a conditional loop. For an example see 5.8.

5.6.3 The conditional loops

The unconditional loop also has two variants. The first variant (the WHILE ... DO statement) checks the conditional expression before performing an iteration and terminates if if evaluates to FALSE.

Example 5.7.
```
CREATE FUNCTION factorial (n INT) RETURNS INT
BEGIN
  DECLARE res INT DEFAULT 1;

  WHILE n > 1 DO
    SET res = res * n, n = n - 1;
  END WHILE;

  RETURN res;
END
```

The second variant (the REPEAT ... UNTIL statement) checks the conditional expression after performing each iteration and terminates if it doesn't evaluate to FALSE.

Example 5.8.
```
CREATE FUNCTION factorial (n INT) RETURNS INT
BEGIN
  DECLARE res INT DEFAULT 1;

  REPEAT
    SET res = res * n, n = n - 1;
  UNTIL n < 2 END REPEAT;

  RETURN res;
END
```

5.7 Conditions

Conditions are used by MySQL to indicate successful or unsuccessful execution of SQL statements. They are identified by SQLSTATE values, which are numbers consisting of five digits. The first two are used to indicate the type of condition. Based on how MySQL handles them, there are four of them.

If the two leading two digits are "00" then the execution of the previous statement was successful. It is not allowed to use them because it would serve no purpose.

The leading digits of the first type are "01". These indicate warnings. They do not terminate execution of the current procedure, but are remembered and presented to the user after terminating the statement.

The NOT FOUND condition (leading digits: "02") is emitted to indicate that the query (usually a SELECT statement) has an empty result set or that the result set is exhausted. (see 5.8). In these cases execution would proceed. If such a condition is thrown using SIGNAL or RESIGNAL, execution terminates.

All other SQLSTATE values indicate errors. Many of these values are reserved by MySQL. Applications may emit them, but most likely they will only confuse users.

To improve maintainability of the code it is possible to define a name for a SQLSTATE value. This is done using the DECLARE CONDITION statement. It is also possible to refer to an error code defined by the DBMS at hand.

Example 5.9.
```
DECLARE CONDITION illegalIndex FOR SQLSTATE 45001;
```

5.7.1 Emitting conditions

The SIGNAL statement is used to emit a signal. Aside from specifying the SQLSTATE value the signal items may be set. They are bits of information which are stored together with the condition. In MySQL most of them are VARCHAR(60) fields and therefore quite flexible and useful for some purposes. They are reported later to the user. Since MySQL v5.6 it is possible to use the GET DIAGNOSTICS statements to query signal items.

The three ways to specify the SQLSTATE value are

- the five-digit number itself
- an error number defined by the DBMS at hand
- a name defined by a preceding DECLARE CONDITION statement.

The RESIGNAL statement may used inside of a handler to emit a signal. The only difference from SIGNAL is that signal items not set by the programmer are copied from the condition the handler reacts to. It is not allowed to use this statement outside of a handler context.

5.7.2 Handling conditions

To handle a condition a handler has to be specified. This is done using the DECLARE HANDLER statement which has to appear at the beginning of a block after all other declare statements.

Aside from the statements to execute as the condition is handled, the handler type has to be specified. There are three of them:

- CONTINUE: Execution continues after the statement which caused the condition.
- EXIT: Execution continues after the block where the handler was declared.
- UNDO: MySQL doesn't support these.
  In DB2 blocks may be declared atomic which begins and commits an implicit session as execution enters and leaves the block. If an UNDO handler is encountered then a rollback is performed. Otherwise it behaves as an EXIT handler.

5.8 Cursors

Cursors are used to access rows in the result set of SQL statements. They are fetched sequentially. If there are no more rows then a NOT FOUND condition is emitted (SQLSTATE "20000"). This can be detected by using a handler. Usually the handler sets a flag which the code reacts to.

To process data from cursors it is advisable to not use the conditional iteration statements. Then the flag set by the handler has to be checked only in one place.

Example 5.10.

```
CREATE PROCEDURE tableSum (OUT sum INT)
COMMENT 'Calculates a sum of all numbers in a table'
BEGIN
  DECLARE done BOOL DEFAULT FALSE;
  DECLARE val INT DEFAULT 0;
  DECLARE values CURSOR FOR SELECT val FROM numbers;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET done = TRUE;

  processing: LOOP
    FETCH values INTO val;

    IF done THEN
      LEAVE processing;
    END IF;

    SET sum = sum + val;
  END LOOP;
END
```

If multiple cursors are accessed then it is difficult to determine which one was exhausted. The flag has to be queried after each FETCH statement because the handler is not able to tell the exact source of the NOT FOUND condition.

Cursor declarations have to appear before handler declarations and after variable and condition declarations. They contain the SQL query whose result set shall be accessed. SELECT statements must not use the INTO clause.

Before a cursor may be used it has to be opened. This is done with the OPEN statement. It is possible to re-open a closed cursor. This way it is possible to process a result set repeatedly.

After processing the cursor it has to be closed with the CLOSE statement. It may be omitted since cursor are closed automatically at the end of the block declaring them.

6 Comparison with other programming languages

SQL-PSM shares similarities with the programming language Ada [4], from which it borrowed large parts of its syntax. As such, it is also similar to other programming languages related to Pascal [2].

The imperative style and the presence of side-effects make it different from most functional languages. It is interesting that large parts of relational algebra are present in functional languages.

Since MySQL implements few, if any, object-oriented features, SQL-PSM shares similarities only with the imperative core of most object-oriented languages.

Since version 3.0, C# supports an embedded query language known as Language-integrated native queries (LINQ) which is basically a `SELECT` statement, with the important difference that the order of clauses is permuted: their order indicates now the actual order in which they are executed. This language allows access to collections and other data sources for many common tasks in a uniform way.

The framework is much more extensive and this syntax is only able to access some of its features.

Example 5.11.

```
CREATE FUNCTION compare (string1Id INT, string2Id INT)
COMMENT 'Compares two strings'
BEGIN
  DECLARE oldDone, done BOOL DEFAULT FALSE;
  DECLARE c1, c2 CHAR(1);
  DECLARE CURSOR string1 FOR
    SELECT c FROM STRINGS WHERE id = string1Id ORDER BY pos ASC;
  DECLARE CURSOR string2 FOR
    SELECT c FROM STRINGS WHERE id = string2Id ORDER BY pos ASC;
  DECLARE HANDLER FOR NOT FOUND
    SET done = TRUE;

  OPEN string1;
  OPEN string2;

  LOOP
    FETCH string1 INTO c1;
    SET oldDone = done;
    FETCH string2 INTO c2;

    IF done AND oldDone THEN
      RETURN 0;
    END IF;

    IF done THEN
      RETURN 1;
    END IF;

    IF oldDone THEN
      RETURN -1;
    END IF;

    IF CHR(c1) < CHR(c2) THEN
      RETURN -1;
    ELSEIF CHR(c1) > CHR(c2) THEN
      RETURN 1;
    END IF;
  END LOOP;
END
```

Example 6.1.

```
var list = new List<int>{ 2, 7, 1, 3, 9 };
var result = from i in list
             where i > 1
             select i;
```

References

[1] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[2] K. Jensen and N. Wirth. *PASCAL: user manual and report.* Lecture notes in computer science. Springer-Verlag, 1978.

[3] A. Kemper and A. Eickler. *Datenbanksysteme.* Oldenbourg, 2006.

[4] U. S. D. of Defense and E. U. d'Amèrica. Department of Defense. *Ada Programming Language: Military Standard.* Military standard. Department of Defense, 1980.

ANSI	American National Standards Institute
DBMS	Database Management System
DDL	Data Definition Language
DML	Data Manipulation Language
ISO	International Organization for Standardization
LINQ	Language-integrated native queries
SQL-PSM	SQL Persistent stored modules
SQL	Structured Query Language