SQL as a Programming Language

99 Bottles of Beer

David Kofler
Supervisor: Sarah Winkler

University of Innsbruck
Institute of Computer Science

January 23, 2013

Outline

Outline

Outline

Outline

Outline

History

- SEQUEL (1975)
  - Query language for the first relational DBMS (System R)
  - Edgar F. Codd
  - Donald D. Chamberlin
  - Raymond F. Boyce
- SQL-86 (1986/87)
- SQL-92 (1992)
- SQL-1999
  - SQL/PSM
  - Triggers
  - Recursive queries
- SQL-2003, SQL-2008, SQL-2011

Tables

Relations

- Collection of tuples (called "rows")
- Schema determines shape of rows
- Primary Key
- Constraints

Tables

Relations

- Collection of tuples (called "rows")
- Schema determines shape of rows
- Primary Key
- Constraints

Relationships

- Elements of the tuples reference other tables
- Supported and enforced by the DBMS
- Normal forms

Handling databases and tables

Creating a database

```
CREATE DATABASE myImportantApp
```

Creating a table

```
CREATE TABLE testTable
  (field INT NULL) ENGINE = InnoDB
```

Handling databases and tables

Creating a database

```
CREATE DATABASE myImportantApp
```

Creating a table

```
CREATE TABLE testTable
  (field INT NULL) ENGINE = InnoDB
```

Getting rid of both of 'em

```
DROP TABLE testTable
DROP DATABASE myImportantApp
```

INSERT and SELECT

INSERT

```
INSERT INTO testTable (field)
  VALUES (2), (3), (4)
```

INSERT and SELECT

INSERT

```
INSERT INTO testTable (field)
  VALUES (2), (3), (4)
```

SELECT

```
SELECT AVG(field), COUNT(*) FROM testTable
```

UPDATE and DELETE

UPDATE

```
UPDATE testTable
  SET field = field / 2
  WHERE field % 2 != 0
```

UPDATE and DELETE

UPDATE

```
UPDATE testTable
  SET field = field / 2
  WHERE field % 2 != 0
```

DELETE

```
DELETE FROM testTable
  WHERE field % 2 == 0
```

Stored routines

- Contain procedural code
- Stored in the current database

Stored Procedures

- Used with the CALL statement
- No RETURN statement allowed

Stored Functions

- To be used in SQL queries
- No recursion
- No transaction commit / rollback

Stored Procedure syntax

⟨*ddl*⟩ := 'CREATE' 'PROCEDURE' ⟨*name*⟩ '(' ⟨*proc-parameters*⟩ ')'
⟨*characteristics*⟩* ⟨*stmt*⟩

⟨*proc-parameters*⟩ ::= ⟨*proc-parameter*⟩ (',' ⟨*proc-parameter*⟩)*

⟨*proc-parameter*⟩ ::= ('IN' | 'OUT' | 'INOUT') ⟨*name*⟩ ⟨*type*⟩

Example

```
DELIMITER //
CREATE PROCEDURE greetMe (IN name VARCHAR(30))
BEGIN
   SELECT CONCAT('Hello', name);
END; //
DELIMITER ;
```

Stored functions example

```
CREATE FUNCTION fac (n INT) RETURNS INT
BEGIN
  DECLARE acc INT DEFAULT 1;

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

  RETURN acc;
END
```

Variables

Variable declaration syntax

⟨*stmt*⟩ ::= 'DECLARE' <ident-list> <type> ('DEFAULT' ⟨*expression*⟩)?

⟨*ident-list*⟩ ::= ⟨*identifier*⟩ (',' ⟨*identifier*⟩)*

Remarks

- Located at the very beginning of each block
- Each block creates its own scope

Data Types

- All simple SQL data types
  - Integers, exact and inexact decimal numbers
  - Date/Time values
  - Strings and text
  - Binary data

Data Types

- All simple SQL data types
  - Integers, exact and inexact decimal numbers
  - Date/Time values
  - Strings and text
  - Binary data
- No compound types

Blocks

Blocks

- Blocks and loops may be named
- ITERATE starts an enclosing loop again
- LEAVE exits an enclosing block

A named block

```
blocklabel: BEGIN
  LEAVE blocklabel;
  SET a = 3;
END blocklabel
```

IF and WHILE

IF

⟨*stmt*⟩ ::= 'IF' ⟨*condition*⟩ 'THEN' ⟨*stmt-list*⟩
('ELSEIF' <condition> 'THEN' ⟨*stmt-list*⟩)*
('ELSE' <stmt-list>)? 'END' 'IF'

IF and WHILE

IF

⟨*stmt*⟩ ::= 'IF' ⟨*condition*⟩ 'THEN' ⟨*stmt-list*⟩
('ELSEIF' <condition> 'THEN' ⟨*stmt-list*⟩)*
('ELSE' <stmt-list>)? 'END' 'IF'

WHILE

⟨*stmt*⟩ ::= 'WHILE' ⟨*condition*⟩ 'DO' ⟨*stmt-list*⟩ 'END' 'WHILE'

Example

GCD (Greatest Common Divisor)

```
CREATE FUNCTION gcd (a INT, b INT)
RETURNS INT
BEGIN
  WHILE a <> b DO
    IF a > b THEN
      SET a = a - b;
    ELSE
      SET b = b - a;
    END IF;
  END WHILE;
  RETURN a;
END
```

Conditions

Overview

- Identified by a 5-digit number (called SQLSTATE)
- Carry up to 10 VARCHAR(64) fields
- Labeled using DECLARE ... CONDITION
- Handled using DECLARE ... HANDLER

Conditions

Overview

- Identified by a 5-digit number (called SQLSTATE)
- Carry up to 10 VARCHAR(64) fields
- Labeled using DECLARE ... CONDITION
- Handled using DECLARE ... HANDLER

Default behaviour

- Class: First two digits of SQLSTATE
- Class '01' (warnings): no action by default
- Class '02' (NOT FOUND): abort, cursors are left untouched
- Class greater than '02': abort, cursors are closed

Cursors

⟨*stmt*⟩ ::= ‘`DECLARE`’ ⟨*identifier*⟩ ‘`CURSOR`’ ‘`FOR`’ ⟨*query*⟩

⟨*stmt*⟩ ::= ‘`OPEN`’ ⟨*identifier*⟩

⟨*stmt*⟩ ::= ‘`FETCH`’ ⟨*identifier*⟩ ‘`INTO`’ ⟨*identifier-list*⟩

⟨*stmt*⟩ ::= ‘`CLOSE`’ ⟨*identifier*⟩

- Declaration before condition handler declarations
- Throws NOT FOUND condition if no more data available

Cursor and condition example

Setting up variables, cursors and handlers

```
CREATE PROCEDURE countGreaterZero
   (OUT count INT)
BEGIN
   DECLARE value INT;
   DECLARE done BOOL DEFAULT FALSE;

   DECLARE valuesCur CURSOR FOR
      SELECT field FROM testTable;

   DECLARE CONTINUE HANDLER FOR NOT FOUND
         SELECT TRUE INTO done
            FROM (SELECT 1) AS temp;
```

Cursor and condition example

Processing the results

```
SET count = 0;
OPEN valuesCur;

WHILE NOT done DO
  FETCH valuesCur INTO value;

  IF NOT done AND value > 0 THEN
    SET count = count + value;
  END IF;
END WHILE;
END
```

Today's use and relevance

- Business applications
  - Business rules at the database level
  - Allows access restrictions
- Ease writing of complex queries
- Saving network bandwidth
- Usefulness depends on the implementation

99 Bottles of Beer

```
CREATE PROCEDURE printBottles(IN max INT)
BEGIN
  DECLARE i INT DEFAULT max;

  WHILE i > 2 DO
    SELECT CONCAT(i, ' bottles of beers on',
      'the wall, ', i, ' bottles of beer.');
    SELECT CONCAT('Take one down and pass '
      'it around, ', i-1,
      ' bottles of beer on the wall.');

    SET i = i - 1;
  END WHILE;
```

99 Bottles of Beer continued

```
  SELECT CONCAT('2 bottles of beers on ',
    'the wall, 2 bottles of beer.');
  SELECT CONCAT('Take one down and pass it',
    ' around, 1 bottle of beer on the wall.');

  SELECT CONCAT('1 bottle of beer on ',
    'the wall, 1 bottle of beer.');
  SELECT CONCAT('Take one down and pass it around',
    ', no more bottles of beer on the wall.');

  SELECT CONCAT('No more bottles of beer on ',
    'the wall, no more bottles of beer.');
  SELECT CONCAT('Go to the store and buy some',
    ' more, ', max, ' bottles of beer on the wall.'
END
```

Thank you for your attention!