Database Management Systems

Sumayyea Salahuddin (Lecturer)
Dept. of Computer Systems Eng.
UET Peshawar

Objectives

- Advance SQL (Part II)
  - Stored Program
    - Stored Procedures
    - Stored Functions
    - Triggers
  - Why use Stored Programs?
  - A Quick Tour via Examples
  - Resources

Stored Programs

- Also known as stored module or stored routines
- It is stored within and executes within database server. On execution, it is executed within the memory address of a database server process or thread
- Three major types of MySQL stored programs:
  - Stored Procedures
    - The most common type of stored program
    - Generic program unit that is executed on request & that can accept multiple input and output parameters
  - Stored Functions
    - Similar to stored procedure but their execution results in return of a single value
    - It can be used within a standard SQL statement.
  - Triggers

Stored Programs (Cont.)

- Triggers
  - These are activated in response to, or are triggered by an activity within the database.
  - Invoked in response to a DML operation (insert, update, delete) against a database table
  - These can be used for data validation.

Why use Stored Procedures?

1) Use of stored programs can lead to a more secure database.
2) Stored programs offer a mechanism to abstract data access routines, which can improve the maintainability of your code as underlying data structures evolve.
3) Stored programs can reduce network traffic, as program can work on data from within the server, rather than having to transfer data across the network.
4) Stored programs can be used to implement common routines accessible from multiple applications possibly using otherwise incompatible frameworks executed either within or from outside the database server.
5) Database-centric logic can be isolated in stored programs & implemented by programmers with more specialized, database experience.
6) The use of stored programs can, under some circumstances, improve the portability of your application.

Why use Stored Procedures? (Cont.)

- These can improve the performance, security, maintainability, and reliability of applications.

Stored Procedure – Example 1

- Embedding SQL in a stored program

```
1  CREATE PROCEDURE example1(  )
2  BEGIN
3    DECLARE
 l_book_count INTEGER;
4
5    SELECT COUNT(*)
6      INTO l_book_count
7      FROM books
8     WHERE author LIKE '%HARRISON,GUY%';
9
10   SELECT CONCAT('Guy has written (or co-written) ',
11          l_book_count ,
12          ' books.');
13
14    -- Oh, and I changed my name, so...
15    UPDATE books
16       SET author = REPLACE (author, 'GUY', 'GUILLERMO')
17     WHERE author LIKE '%HARRISON,GUY%';
18
19 END
```

Stored Procedure – Example 1 (Cont.)

- Example Explanation

Line (s)	Explanation
1	This section, the header of the program, defines the name (`example1`) and type (`PROCEDURE`) of our stored program.
2	This `BEGIN` keyword indicates the beginning of the *program body*, which contains the declarations and executable code that constitutes the procedure. If the program body contains more than one statement (as in this program), the multiple statements are enclosed in a `BEGIN-END` block.
3	Here we declare an integer variable to hold the results of a database query that we will subsequently execute.
5-8	We run a query to determine the total number of books that Guy has authored or coauthored. Pay special attention to line 6: the `INTO` clause that appears within the `SELECT` serves as the "bridge" from the database to the local stored program language variables.
10-12	We use a simple `SELECT` statement (e.g., one without a `FROM` clause) to display the number of books. When we issue a `SELECT` without an `INTO` clause, the results are returned directly to the calling program. This is a non-ANSI extension that allows stored programs to easily return result sets (a common scenario when working with SQL Server and other RDBMSs).
14	This single-line comment explains the purpose of the `UPDATE`.
15-17	Guy has decided to change the spelling of his first name to "Guillermo" he's probably being stalked by fans of his Oracle bookso we issue an `UPDATE` against the `books` table. We take advantage of the built-in `REPLACE` function to locate all instances of "GUY" and replace them with "GUILLERMO".

Stored Procedure – Example 2

- Stored Procedure with Control and Conditional Logic

```
1  CREATE PROCEDURE pay_out_balance
2       (account_id_in INT)
3
4  BEGIN
5
6  DECLARE l_balance_remaining NUMERIC(10,2);
7
8  payout_loop:LOOP
9    SET l_balance_remaining = account_balance(account_id_in);
10
11    IF l_balance_remaining < 1000 THEN
12     LEAVE payout_loop;
13
14    ELSE
15      CALL apply_balance(account_id_in, l_balance_remaining);
16    END IF;
17
18  END LOOP;
19
20 END
```

Stored Procedure – Example 2 (Cont.)

- Example Explanation

Line (s)	Explanation
1-3	This is the header of our procedure; line 2 contains the parameter list of the procedure, which in this case consists of a single incoming value (the identification number of the account).
6	Declare a variable to hold the remaining balance for an account.
8-18	This simple loop (named so because it is started simply with the keyword `LOOP`, as opposed to `WHILE` or `REPEAT`) iterates until the account balance falls below 1000. In MySQL, we can name the loop (line 8, `payout_loop`), which then allows us to use the `LEAVE` statement (see line 12) to terminate that particular loop. After leaving a loop, the MySQL engine will then proceed to the next executable statement following the `END LOOP`; statement (line 18).
9	Call the `account_balance` function (which must have been previously defined) to retrieve the balance for this account. MySQL allows you to call a stored program from within another stored program, thus facilitating reuse of code. Since this program is a function, it returns a value and can therefore be called from within a MySQL `SET` assignment.
11-16	This `IF` statement causes the loop to terminate if the account balance falls below $1,000. Otherwise (the `ELSE` clause), it applies the balance to the next charge. You can construct much more complex Boolean expressions with `ELSEIF` clauses, as well.
15	Call the `apply_balance` procedure. This is an example of code reuse; rather than repeating the logic of `apply_balance` in this procedure, we call a common routine.

Stored Procedure – Example 3

- Error Handling in a Stored Program

```
1  CREATE PROCEDURE sp_product_code
2      (in_product_code VARCHAR(2),
3       in_product_name VARCHAR(30))
4
5  BEGIN
6
7    DECLARE l_dupkey_indicator INT DEFAULT 0;
8    DECLARE duplicate_key CONDITION FOR 1062;
9    DECLARE CONTINUE HANDLER FOR duplicate_key SET l_dupkey_indicator =1;
10
11   INSERT INTO product_codes (product_code, product_name)
12   VALUES (in_product_code, in_product_name);
13
14   IF l dupkey_indicator THEN
15     UPDATE product_codes
16        SET product_name=in_product_name
17      WHERE product_code=in_product_code;
18   END IF;
19
20 END
```

Stored Procedure – Example 3 (Cont.)

- Example Explanation

Line (s)	Explanation
1-4	This is the header of the stored procedure, accepting two `IN` parameters: product code and product name.
7	Declare a variable that we will use to detect the occurrence of a duplicate key violation. The variable is initialized with a value of 0 (false); subsequent code will ensure that it gets set to a value of 1 (true) only if a duplicate key violation takes place.
8	Define a named condition, `duplicate_key`, that is associated with MySQL error 1062. While this step is not strictly necessary, we recommend that you define such conditions to improve the readability of your code (you can now reference the error by name instead of by number).
9	Define an error handler that will trap the duplicate key error and then set the value of the variable `l_dupkey_indicator` to 1 (true) if a duplicate key violation is encountered anywhere in the subsequent code.
11-12	Insert a new product with the user-provided code and name.
14	Check the value of the `l_dupkey_indicator` variable. If it is still 0, then the `INSERT` was successful and we are done. If the value has been changed to 1 (true), we know that there has been a duplicate key violation. We then run the `UPDATE` statement in lines 15-17 to change the name of the product with the specified code.

Stored Procedure – Example 4

```
mysql> delimiter /
mysql> create procedure tour_entry()
    -> begin
    -> select TourName, TourType, year
    -> from tour natural join entry;
    -> end/
Query OK, 0 rows affected (0.12 sec)

mysql> delimiter ;
mysql> call tour_entry();
+-------------+----------+------+
| TourName    | TourType | year |
+-------------+----------+------+
| Leeston     | Social   | 2013 |
| Leeston     | Social   | 2014 |
| Kaiapoi     | Social   | 2014 |
| WestCoast   | Social   | 2014 |
| Canterburry | Open     | 2012 |
| Canterburry | Open     | 2014 |
| Otago       | Open     | 2013 |
| Otago       | Open     | 2014 |
+-------------+----------+------+
8 rows in set (0.09 sec)
```

Stored Procedure – Example 5

```
delimiter /
create procedure member_contact(in fn varchar(15), in ln varchar(15))
begin
select Phone from member
where FirstName = fn and LastName = ln;
end/
```

```
mysql> call member_contact('Brenda', 'Nolan');
+--------+
| Phone  |
+--------+
| 442649 |
+--------+
1 row in set (0.00 sec)
```

Stored Function – Example 1

- Stored Function to Calculate Age from Date of Birth

```
1 CREATE FUNCTION f_age (in_dob datetime) returns int
2   NO SQL
3 BEGIN
4   DECLARE l_age INT;
5   IF DATE_FORMAT(NOW(  ),'00-%m-%d') >= DATE_FORMAT(in_dob,'00-%m-%d') THEN
6      -- This person has had a birthday this year
7      SET l_age=DATE_FORMAT(NOW(  ),'%Y')-DATE_FORMAT(in_dob,'%Y');
8    ELSE
9      -- Yet to have a birthday this year
10      SET l_age=DATE_FORMAT(NOW(  ),'%Y')-DATE_FORMAT(in_dob,'%Y')-1;
11   END IF;
12   RETURN(l_age);

END;
```

Stored Function – Example 1 (Cont.)

- Example Explanation

Lines (s)	Explanation
1	Define the function: its name, input parameters (a single date), and return value (an integer).
2	This function contains no SQL statements. There's some controversy about the use of this clause see Chapters 3 and 10 for more discussion.
4	Declare a local variable to hold the results of our age calculation.
5-11	This `IF-ELSE-END IF` block checks to see if the birth date in question has occurred yet this year.
7	If the birth date has, in fact, passed in the current year, we can calculate the age by simply subtracting the year of birth from the current year.
10	Otherwise (i.e., the birth date is yet to occur this year), we need to subtract an additional year from our age calculation.
12	Return the age as calculated to the calling program.

Stored Function – Example 2

- Using a Stored Function within a SQL Statement

```
mysql> SELECT firstname,surname, date_of_birth, f_age(date_of_birth) AS age
    ->   FROM employees LIMIT 5;
+-----------+---------+---------------------+------+
| firstname | surname | date_of_birth       | age  |
+-----------+---------+---------------------+------+
| LUCAS     | FERRIS  | 1984-04-17 07:04:27 |   21 |
| STAFFORD  | KIPP    | 1953-04-22 06:04:50 |   52 |
| GUTHREY   | HOLMES  | 1974-09-12 08:09:22 |   31 |
| TALIA     | KNOX    | 1966-08-14 11:08:14 |   39 |
| JOHN      | MORALES | 1956-06-22 07:06:14 |   49 |
+-----------+---------+---------------------+------+
```

Stored Function – Example 2

```
mysql> delimiter /
mysql> create function search_emp(id int) returns varchar(30)
    -> begin
    -> declare name_found varchar(30) default "";
    -> select emp_name into name_found from employee where emp_id = id;
    -> return name_found;
    -> end/
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> select search_emp(18);
+----------------+
| search_emp(18) |
+----------------+
| HBV            |
+----------------+
1 row in set (0.00 sec)
```

Stored Function – Example 3

```
mysql> create function memberDuration (d year) returns int
    -> begin
    -> declare a int;
    -> select year(current_date) into a;
    -> set a = a - d;
    -> return(a);
    -> end/
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select concat(FirstName, ' ', LastName) as Name,
    -> memberDuration(year(JoinDate)) as MemberDuration from member;
+------------------+----------------+
| Name             | MemberDuration |
+------------------+----------------+
| Melissa McKenzie |              8 |
| Michael Stone    |              4 |
| Brenda Nolan     |              7 |
| Helen Branch     |              2 |
| Sarah Beck       |              3 |
| Thomas Spence    |              7 |
| Sandra Burton    |              2 |
| William Cooper   |              5 |
| Barbara Olson    |              2 |
| Robert Pollard   |              2 |
+------------------+----------------+
10 rows in set (0.00 sec)
```

Stored Function – Example 4

```
mysql> delimiter /
mysql> create function member_con(fn varchar(15),
    -> ln varchar(15)) returns int(11)
    -> begin
    -> declare contact int default 0;
    -> select Phone into contact
    -> from member
    -> where FirstName = fn and LastName = ln;
    -> return contact;
    -> end/
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> select member_con('Brenda', 'Nolan');
+-------------------------------+
| member_con('Brenda', 'Nolan') |
+-------------------------------+
|                        442649 |
+-------------------------------+
1 row in set (0.00 sec)
```

Triggers

- Basic Trigger Syntax

```
CREATE
    TRIGGER `event_name` BEFORE/AFTER INSERT/UPDATE/DELETE
    ON `database`.`table`
    FOR EACH ROW BEGIN
                -- trigger body
                -- this code is applied to every
                -- inserted/updated/deleted row
    END;
```

Triggers – Example 1

- Trigger to Maintain a Derived Column Value

```
1  CREATE TRIGGER employees_trg_bu
2       BEFORE UPDATE ON employees
3       FOR EACH ROW
4    BEGIN
5      IF NEW.salary <50000 THEN
6        SET NEW.contrib_401K=500;
7     ELSE
8       SET NEW.contrib_401K=500+(NEW.salary-50000)*.01;
9     END IF;
10  END
```

Triggers – Example 1 (Cont.)

- Example Explanation

Line (s)	Explanation
1	A trigger has a unique name. Typically, you will want to name the trigger so as to reveal its nature. For example, the "`bu`" in the trigger's name indicates that this is a `BEFORE UPDATE` TRigger.
2	Define the conditions that will cause the trigger to fire. In this case, the trigger code will execute prior to an `UPDATE` statement on the `employees` table.
3	`FOR EACH ROW` indicates that the trigger code will be executed once for each row being affected by the DML statement. This clause is mandatory in the current MySQL 5 trigger implementation.
4-10	This `BEGIN-END` block defines the code that will run when the trigger is fired.
5-9	Automatically populate the `contrib_401K` column in the `employees` table. If the new value for the `salary` column is less than 50000, the `contrib._401K` column will be set to 500. Otherwise, the value will be calculated as shown in line 8.

Triggers – Example 2

```
mysql> select * from employee_psh;
+--------+----------+----------------------+--------+--------------+
| emp_id | emp_name | emp_job              | dep_ID | emp_domicile |
+--------+----------+----------------------+--------+--------------+
|      1 | BCA      | Lecturer             |     10 | Peshawar     |
|      3 | XYZ      | Assitant Professor   |     10 | Peshawar     |
|      4 | CVA      | Lecturer             |     50 | Peshawar     |
|      5 | VAC      | Assistant Professor  |     50 | Peshawar     |
|     11 | FGJ      | Assistant Professor  |     25 | Peshawar     |
|     16 | JKF      | Lecturer             |     60 | Peshawar     |
|      4 | CVA      | Lecturer             |     50 | Mardan       |
+--------+----------+----------------------+--------+--------------+
7 rows in set (0.12 sec)

mysql> alter table employee_psh add column salary float;
Query OK, 0 rows affected (0.57 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table employee_psh add column bonus float;
Query OK, 0 rows affected (0.51 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Triggers – Example 2 (Cont.)

```
+--------+-------+
| salary | bonus |
+--------+-------+
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
+--------+-------+
```

```
mysql> delimiter /
mysql> create trigger emp_bonus
    -> before update on employee_psh
    -> for each row
    -> begin
    -> if new.salary < 25000 then
    -> set new.bonus = 500;
    -> else
    -> set new.bonus = 500 + (new.salary*0.05);
    -> end if;
    -> end/
Query OK, 0 rows affected (0.18 sec)
```

```
mysql> update employee_psh
    -> set salary = 105000
    -> where emp_id = 3;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
+--------+-------+
| salary | bonus |
+--------+-------+
|  24000 |   500 |
| 105000 |  5750 |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
|   NULL |  NULL |
+--------+-------+
```

Resources

- Books
  1) MySQL Stored Procedure Programming, by Guy Harrison with Steven Feuerstein
  2) MySQL in a Nutshell, by Russell Dyer
  3) Web Database Applications with PHP and MySQL, by Hugh Williams and David Lane
  4) MySQL, by Paul DuBois
  5) High Performance MySQL, by Jeremy Zawodny and Derek Balling
  6) MySQL Cookbook, by Paul DuBois
  7) Pro MySQL, by Michael Krukenberg and Jay Pipes
  8) MySQL Design and Tuning, by Robert D. Schneider
  9) SQL in a Nutshell, by Kevin Kline, et al.
  10) Learning SQL, by Alan Beaulieu

Resources (Cont.)

- Internet Resources
  1) MySQL - Start at http://www.mysql.com
  2) MySQL Developer Zone - http://dev.mysql.com/
  3) MySQL Online Documentation - http://dev.mysql.com/doc/
  4) MySQL Forums - http://www.planetmysql.org/
  5) MySQL Stored Routines Library - http://savannah.nongnu.org/projects/mysql-sr-lib/