Views, Stored Procedures, Functions, and Triggers

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations
  - Since the table may not physically be stored
- Allows full query operations

SQL Views: An Example

- Create a view for Department Managers:

```
CREATE VIEW MANAGER AS
    SELECT FNAME, LNAME, DName, Dnumber, SALARY
    FROM EMPLOYEE, DEPARTMENT
    WHERE SSN=MGRSSN AND DNO=DNUMBER;
```

- Find employees who earn more than their managers

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E, MANAGER M
WHERE E.DNO=M.DNUMBER AND E.SALARY > M.SALARY;
```

- When no longer needed, a view can be dropped:

```
DROP VIEW MANAGER;
```

View Implementation

- There are two ways to implement a view:
- Approach 1: Query modification
  - Modify the view query into a query on the underlying base tables
  - Example:

    SELECT * FROM Manager WHERE Salary > 100000

    becomes

    SELECT Fname, Lname, Dname, Dnumber, Salary
    FROM EMPLOYEE, DEPARTMENT
    WHERE SSN=MgrSSN AND Salary > 100000
  - Disadvantage:
    - Inefficient for views defined via complex queries

View Implementation

- Approach 2: View materialization
  - Involves physically creating and keeping a temporary table
  - Concerns:
    - Maintaining correspondence between the base table and the view when the base table is updated

- ORACLE

  CREATE MATERIALIZED VIEW or CREATE SNAPSHOT

Update Views

- Update on a view can be implemented by mapping it to an update on the underlying base table

```
UPDATE MANAGER
SET Salary = 1.1*Salary
WHERE Dname = 'Research';
```

  - Becomes:

```
UPDATE EMPLOYEE
SET Salary = 1.1*Salary
WHERE SSN in (SELECT MgrSSN
                FROM DEPARTMENT
                WHERE DName = 'Research');
```

- Updating views involving joins are not always possible
  - Views defined using groups and aggregate functions are not updateable
- For mySQL, the keyword **"`WITH CHECK OPTION`"** must be added to the view definition if the view is to be updated

Stored Procedures in MySQL

- A stored procedure contains a sequence of SQL commands stored in the database catalog so that it can be invoked later by a program

- Stored procedures are declared using the following syntax:

  Create Procedure <proc-name>

  (param_spec$_1$, param_spec$_2$, …, param_spec$_n$)

  begin

  -- execution code

  end;

  where each param_spec is of the form:

  [in | out | inout] <param_name> <param_type>

  – in mode: allows you to pass values into the procedure,
  – out mode: allows you to pass value back from procedure to the calling program

Example

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
```

```
mysql> select * from department;
+---------+------------+
| dnumber | dname      |
+---------+------------+
|       1 | Payroll    |
|       2 | TechSupport |
|       3 | Research   |
+---------+------------+
```

- Suppose we want to keep track of the total salaries of employees working for each department

```
mysql> create table deptsal as
    -> select dnumber, 0 as totalsalary from department;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
```

We need to write a procedure to update the salaries in the deptsal table

Example

```
mysql> delimiter //
```

Step 1: Change the delimiter (i.e., terminating character) of SQL statement from semicolon (;) to something else (e.g., //)

So that you can distinguish between the semicolon of the SQL statements in the procedure and the terminating character of the procedure definition

Example

```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)
```

Step 2:

1. Define a procedure called updateSalary which takes as input a department number.
2. The body of the procedure is an SQL command to update the totalsalary column of the deptsal table.
3. Terminate the procedure definition using the delimiter you had defined in step 1 (//)

Example

```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
```

Step 3: Change the delimiter back to semicolon (;)

Example

```
mysql> call updateSalary(1);
Query OK, 0 rows affected (0.00 sec)

mysql> call updateSalary(2);
Query OK, 1 row affected (0.00 sec)

mysql> call updateSalary(3);
Query OK, 1 row affected (0.00 sec)
```

Step 4: Call the procedure to update the totalsalary for each department

Example

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Step 5: Show the updated total salary in the deptsal table

Stored Procedures in MySQL

- Use show procedure status to display the list of stored procedures you have created

```
mysql> show procedure status;
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| Db   | Name          | Type      | Definer | Modified            | Created             | Security_
type | Comment | character_set_client | collation_connection | Database Collation |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| ptan | updateSalary0 | PROCEDURE | ptan@%  | 2010-03-16 12:21:55 | 2010-03-16 12:21:55 | DEFINER
     |         | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
1 row in set (0.02 sec)
```

- Use drop procedure to remove a stored procedure

```
mysql> drop procedure updateSalary;
Query OK, 0 rows affected (0.00 sec)
```

Stored Procedures in MySQL

- You can declare variables in stored procedures
- You can use flow control statements (conditional IF-THEN-ELSE or loops such as WHILE and REPEAT)
- MySQL also supports cursors in stored procedures.
  - A cursor is used to iterate through a set of rows returned by a query so that we can process each individual row.
- To learn more about stored procedures, go to:

  http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

Example using Cursors

- The previous procedure updates one row in deptsal table based on input parameter
- Suppose we want to update all the rows in deptsal simultaneously
  - First, let's reset the totalsalary in deptsal to zero

```
mysql> update deptsal set totalsalary = 0;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 3  Changed: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Example using Cursors

```
mysql> delimiter $$
mysql> drop procedure if exists updateSalary$$
Query OK, 0 rows affected (0.00 sec)

mysql> create procedure updateSalary()
    -> begin
    ->          declare done int default 0;
    ->          declare current_dnum int;
    ->          declare dnumcur cursor for select dnumber from deptsal;
    ->          declare continue handler for not found set done = 1;
    ->
    ->          open dnumcur;
    ->
    ->          repeat
    ->                fetch dnumcur into current_dnum;
    ->                update deptsal
    ->                set totalsalary = (select sum(salary) from employee
    ->                                   where dno = current_dnum)
    ->                where dnumber = current_dnum;
    ->          until done
    ->          end repeat;
    ->
    ->          close dnumcur;
    -> end$$
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Drop the old procedure

Use cursor to iterate the rows

Example using Cursors

- Call procedure

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
3 rows in set (0.01 sec)

mysql> call updateSalary;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Another Example

- Create a procedure to give a raise to all employees

```
mysql> select * from emp;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL |  90000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)
```

Another Example

```
mysql> delimiter |
mysql> create procedure giveRaise (in amount double)
    -> begin
    ->         declare done int default 0;
    ->         declare eid int;
    ->         declare sal int;
    ->         declare emprec cursor for select id, salary from employee;
    ->         declare continue handler for not found set done = 1;
    ->
    ->         open emprec;
    ->         repeat
    ->                 fetch emprec into eid, sal;
    ->                 update employee
    ->                 set salary = sal + round(sal * amount)
    ->                 where id = eid;
    ->         until done
    ->         end repeat;
    -> end |
Query OK, 0 rows affected (0.00 sec)
```

Another Example

```
mysql> delimiter ;
mysql> call giveRaise(0.1);
Query OK, 0 rows affected (0.00 sec)

mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 110000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  55000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  88000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  55000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)
```

Functions

- Functions are declared using the following syntax:

function <function-name> (param_spec$_1$, …, param_spec$_k$)
returns <return_type>
[not] deterministic allow optimization if same output
for the same input (use RAND not deterministic)
Begin
-- execution code
end;

where param_spec is:
[in | out | in out] <param_name> <param_type>

  - You need ADMIN privilege to create functions on mysql-user server

Example of Functions

```
mysql> select * from employee;
+----+------+----------+--------+------------+------+
| id | name | superid  | salary | bdate      | dno  |
+----+------+----------+--------+------------+------+
|  1 | john |        3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |        3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |     NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |        1 |  50000 | 1970-01-17 |    2 |
|  5 | bill |     NULL |   NULL | 1985-01-20 |    1 |
+----+------+----------+--------+------------+------+
5 rows in set (0.00 sec)

mysql> deliniter |
mysql> create function giveRaise (oldval double, amount double
    -> returns double
    -> deterministic
    -> begin
    ->       declare newval double;
    ->       set newval = oldval * (1 + amount);
    ->       return newval;
    -> end |
Query OK, 0 rows affected (0.00 sec)

mysql> deliniter ;
```

Example of Functions

```
mysql> select name, salary, giveRaise(salary, 0.1) as newsal
    -> from employee;
+------+--------+--------+
| name | salary | newsal |
+------+--------+--------+
| john | 100000 | 110000 |
| mary |  50000 |  55000 |
| bob  |  80000 |  88000 |
| tom  |  50000 |  55000 |
| bill |   NULL |   NULL |
+------+--------+--------+
5 rows in set (0.00 sec)
```

SQL Triggers

- To monitor a database and take a corrective action when a condition occurs
  - Examples:
    - Charge $10 overdraft fee if the balance of an account after a withdrawal transaction is less than $500
    - Limit the salary increase of an employee to no more than 5% raise

```
CREATE TRIGGER trigger-name
    trigger-time trigger-event
    ON table-name
    FOR EACH ROW
          trigger-action;
```

  - trigger-time ∈ {BEFORE, AFTER}
  - trigger-event ∈ {INSERT,DELETE,UPDATE}

SQL Triggers: An Example

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

- We want to create a trigger to update the total salary of a department when a new employee is hired

SQL Triggers: An Example

- Create a trigger to update the total salary of a department when a new employee is hired:

```
mysql> delimiter !
mysql> create trigger update_salary
    -> after insert on employee
    -> for each row
    -> begin
    ->       if new.dno is not null then
    ->           update deptsal
    ->           set totalsalary = totalsalary + new.salary
    ->           where dnumber = new.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)

mysql> delimiter ;
```

- The keyword “new” refers to the new row inserted

SQL Triggers: An Example

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (6,'lucy',null,90000,'1981-01-01',1);
Query OK, 1 row affected (0.08 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (7,'george',null,45000,'1971-11-11',null);
Query OK, 1 row affected (0.02 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> drop trigger update_salary;
Query OK, 0 rows affected (0.00 sec)
```

totalsalary increases by 90K

totalsalary did not change

SQL Triggers: An Example

- A trigger to update the total salary of a department when an employee tuple is modified:

```
mysql> delimiter !
mysql> create trigger update_salary2
    -> after update on employee
    -> for each row
    -> begin
    ->       if old.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary - old.salary
    ->          where dnumber = old.dno;
    ->       end if;
    ->       if new.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary + new.salary
    ->          where dnumber = new.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)
```

SQL Triggers: An Example

```
mysql> delimiter ;
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL |  90000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> update employee set salary = 100000 where id = 6;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      200000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

SQL Triggers: An Example

- A trigger to update the total salary of a department when an employee tuple is deleted:

```
mysql> delimiter !
mysql> create trigger update_salary3
    -> before delete on employee
    -> for each row
    -> begin
    ->       if (old.dno is not null) then
    ->           update deptsal
    ->           set totalsalary = totalsalary - old.salary
    ->           where dnumber = old.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.08 sec)

mysql> delimiter ;
```

SQL Triggers: An Example

```
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL | 100000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)
```

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      200000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

```
mysql> delete from employee where id = 6;
Query OK, 1 row affected (0.02 sec)

mysql> delete from employee where id = 7;
Query OK, 1 row affected (0.03 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

SQL Triggers

- To list all the triggers you have created:

```
mysql> show triggers;
```