Module 7: Advanced SQL

Edited by Radhika Sukapuram

Database System Concepts, 7th Ed.

Functions and Procedures

Functions and Stored Procedures

- Functions and procedures allow "business logic" to be stored in the database and executed from SQL statements
  - A rule on the minimum/maximum number of courses a student can take in a semester
  - Minimum number of courses an instructor must teach in a year
  - Maximum number of leaves an employee can take

Functions and Procedures contd.

- Advantages
  - Minimizes data transfer between application and database
  - Utilizes power of DBMS server
  - Different users can reuse the stored procedure – reduces application logic
  - Needs change only in one place if business logic changes
- These can be defined either by
  - the procedural component of SQL or
  - by an external programming language - Java, C, or C++.
- The syntax we present here is defined by the SQL standard.
  - Most databases implement nonstandard versions
  - Oracle : PL/SQL, Microsoft SQL Server: TransactSQL , PostgreSQL(PL/pgSQL)
  - MySQL: https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

Declaring SQL Functions

- Define a function:
  - input - name of a department
  - output- number of instructors in that department.

```
create function dept_count (dept_name varchar(20))
        returns integer
        begin
        declare d_count  integer;
              select count (* ) into d_count
              from instructor
              where instructor.dept_name = dept_name;
        return d_count;
        end
```

Calling a function

- The function *dept_*count can be used to
  - find the department names and budget of all departments with more than 12 instructors.

    select *dept_name, budget*
    from *department*
    where *dept_*count (*dept_name*) > 12;
- A function can be used in an expression

Table Functions

- **table functions :** functions that can return tables as results
- Example: Return all instructors in a given department

```
create function instructors_of (dept_name char(20))
        returns table  (
                ID varchar(5),
                name varchar(20),
                dept_name varchar(20),
                salary numeric(8,2))
        return table
                (select ID, name, dept_name, salary
                 from instructor
                 where instructor.dept_name = instructors_of.dept_name);
```

Passed parameter

- Usage

```
select *
from table (instructor_of ( 'Music' ));
```

- Table functions can be thought of as parameterized views

SQL Procedures

- The *dept_count* function could instead be written as a procedure:

```
create procedure dept_count_proc (in dept_name varchar(20),
                                  out d_count integer)
  begin
     select count(*) into d_count
     from instructor
     where instructor.dept_name = dept_count_proc.dept_name;
  end
```

- **in** and **out** are parameters
  - Values are assigned to **in** parameters by the caller
  - **out** parameters have values are set in the procedure in order to return results.
- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the **call** statement.

```
declare d_count integer;
call dept_count_proc( 'Physics', d_count);
```

SQL Procedures (Cont.)

- Procedures and functions can be invoked also from dynamic SQL
- SQL allows more than one procedure of the same name so long as the number of arguments of the procedures with the same name is different.
- The name, along with the number of arguments, is used to identify the procedure.

Language Constructs for Procedures & Functions

- SQL supports constructs that gives it almost all the power of a general-purpose programming language (Persistent Storage Module of SQL standard)
  - Warning: most database systems implement their own variant of the standard syntax below.
- **declare** – to declare variables
- **set** – to assign values to variables
- Compound statement: **begin** … **end,**
  - May contain multiple SQL statements between **begin** and **end**.
  - Local variables can be declared within compound statements
  - **begin atomic** … **end** for atomic transactions

Language Constructs – if-then-else

- Conditional statements (**if-then-else**)

if *boolean expression*
then *statement or compound statement*
elseif *boolean expression*
then *statement or compound statement*
else *statement or compound statement*
end if

```
mysql> delimiter //
mysql> CREATE FUNCTION isodd(input_number int)
    ->           RETURNS int
    -> BEGIN
    ->           DECLARE v_isodd INT;
    ->
    ->           IF MOD(input_number,2)=0 THEN
    ->                   SET v_isodd=FALSE;
    ->           ELSE
    ->                   SET v_isodd=TRUE;
    ->           END IF;
    ->
    ->           RETURN(v_isodd);
    ->
    -> END ;
    -> //
Query OK, 0 rows affected (0.05 sec)

mysql> select isodd(5)//
+----------+
| isodd(5) |
+----------+
```

Language Constructs for Procedures & Functions cont.

- While and repeat statements:
  - **while** boolean expression **do**
    sequence of statements ;
    end while
  - **repeat**
    sequence of statements ;
    until boolean expression
    end repeat

Language Constructs (Cont.)

- **for** loop
  - Permits iteration over all results of a query
- Example: Find the budget of all departments

```
declare n integer default 0;
for r as
        select budget from department  /*Fetches one row at a time*/
        where dept_name = 'Music'
 do
                set n = n + r.budget
 end for
```

- **leave** to exit the loop //Like a break statement in C
- **iterate** starts on the next tuple, from the beginning of the loop, skipping the remaining statements

Example 1: procedure

```
create procedure GetCustomerLevel(
   in  p_customerNumber int(11),
   out p_customerLevel  varchar(10))
begin
   declare creditlim integer;

   select creditlimit into creditlim
   from customers
   where customerNumber = p_customerNumber;

   if creditlim > 50000 then
      set p_customerLevel = 'PLATINUM';
   elseif (creditlim <= 50000 AND creditlim >= 10000) then
      set p_customerLevel = 'GOLD';
   elseif creditlim < 10000 then
      set p_customerLevel = 'SILVER';
   end if;

end;
```

Example 2: procedure

- Registers student for a course after ensuring classroom capacity is not exceeded
  - Returns 0 on success and -1 if capacity is exceeded
  - *takes (ID, course id, sec id, semester, year, grade)* –courses that students take
  - *classroom (building, room number, capacity)*
  - *section (course id, sec id, semester, year, building, room number, time slot id*)

```
-- Registers a student after ensuring classroom capacity is not exceeded
-- Returns 0 on success, and -1 if capacity is exceeded.
create function registerStudent(
                    in s_id varchar(5),
                    in s_courseid varchar (8),
                    in s_secid varchar (8),
                    in s_semester varchar (6),
                    in s_year numeric (4,0),
                    out errorMsg varchar(100)
returns integer
begin
    declare currEnrol int;
    select count(*) into currEnrol
        from takes
        where course_id = s_courseid and sec_id = s_secid
            and semester = s_semester and year = s_year;
    declare limit int;
    select capacity into limit
        from classroom natural join section
        where course_id = s_courseid and sec_id = s_secid
            and semester = s_semester and year = s_year;
    if (currEnrol < limit)
        begin
            insert into takes values
                (s_id, s_courseid, s_secid, s_semester, s_year, null);
            return(0);
        end
    -- Otherwise, section capacity limit already reached
    set errorMsg = 'Enrollment limit reached for course ' || s_courseid
        || ' section ' || s_secid;
    return(-1);
end;
```