Chapter 5

Advanced SQL

Slides by Silbershatz, Modifications by Rogers and Brown

Procedural Extensions and Stored Procedures

- SQL Provides a module language
  - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc
- Stored Procedures
  - Can store procedures in the database
    - Then execute them using the call statement
  - Permit external applications to operate on the database without knowing about internal details
- Object-oriented aspects of these features are covered in Chapter 22 (Object Oriented Databases)
  - We will not get THAT far this term

Functions and Procedures

- SQL:1999 supports functions and procedures
  - Functions/Procedures can be written in SQL itself, or in an external programming language
  - Functions are particularly useful with specialized data types such as images and geometric objects
    - Example: Functions to check if polygons overlap or to compare images for similarity
  - Some database systems support table-valued functions, which can return a relation as a result
- SQL:1999 also supports a rich set of imperative constructs, including :
  - Loops
  - If-Then-Else
  - Assignment
- Many databases have proprietary procedural extensions to SQL that differ from SQL:1999

SQL Functions

- Define a function that, given the name of a department, returns the count of the 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
```

- Find the department name and budget of all departments with more than 12 instructors

```
select dept_name, budget
from department
where dept_count (dept_name) > 1
```

Table Functions

- SQL:2003 added functions that return a relation as a result
- Example: Return all accounts owned by a given customer

```
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)
```

- Usage

```
select *
from table (instructors_of ('Music'))
```

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
```

- 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);
```

  Procedures and functions can be invoked also from dynamic SQL
- SQL:1999 allows more than one function/procedure of the same name (called overloading), as long as the number of arguments differ or at least the types of arguments differ

Procedural Constructs

- Warning: most database systems implement their own variant of the standard sytax
  - Read the system manual for the system you are using for any variations
- Compound statement: begin … end,
  - May contain multiple SQL statements between begin and end
  - Local variables can be declared within a compound statements
- While and repeat statements:

```
declare n integer default 0;
while n < 10 do
        set n = n + 1
end while

repeat
        set n = n – 1
until n = 0
end repeat
```

Procedural Constructs

- For loop
  - Permits iteration over all results of a query
  - Example:

```
declare n integer default 0;
for r as
        select budget from department
        where dept_name = 'Music'
do
        set n = n – r.budget
end for
```

Procedural Constructs

- Conditional Statements (if-then-else)
  SQL:1999 also supports case statements similar to C++ case statement
- Example procedure: registers a student after ensuring classroom capacity is not exceeded
  - Returns a 0 on success, and -1 if capacity would be exceeded
  - See the book for more details
- Signaling of exception conditions, and declaring handlers for exceptions

```
declare out_of_classroom_seats condition
declare exit handler for out_of_classroom_seats
begin
        ...
        ..signal out_of_classroom_seats
end
```

  - The handler here is exit – causes enclosing begin..end statements to be exited
  - Other actions possible on exception

External Language Functions/Procedures

- SQL:1999 permits the use of functions and procedures written in other languages such as C or C++
- Declaring external language procedures and functions

```
create procedure dept_count_proc (in dept_name varchar (20),
                                  out count integer)
    language C
    external name        '/usr/avi/bin/dept_count_proc'

    create function dept_count (dept_name varchar (20) )
    returns integer
    language C
    external name '/usr/avi/bin/dept_count'
```

External Language Functions/Procedures

- Benefits of external language functions/procedures:
  - More efficient for many operations, and more expressive power.
- Drawbacks
  - Code to implement function may need to be loaded into database system and executed in the database system's address space.
    - Risk of accidental corruption of database structures
    - Security risk, allowing users access to unauthorized data
  - There are alternatives, which give good security at the cost of potentially worse performance.
  - Direct execution in the database system's space is used when efficiency is more important than security.

Security with External Language Routines

- To deal with security problems
  - Use sandbox techniques
    - That is, use a safe language like Java, which cannot be used to access/damage other parts of the database code.
  - Or, run external language functions/procedures in a separate process, with no access to the database process' memory.
    - Parameters and results communicated via inter-process communication
- Both have performance overheads
- Many database systems support both above approaches as well as direct executing in database system address space.

Triggers

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Note : Syntax illustrated here may not work exactly on your database system; check the system manuals

Trigger Example

- E.g. time_slot_id is not a primary key of timeslot, so we cannot create a foreign key constraint from section to timeslot
- Alternative: use triggers on section and timeslot to enforce integrity constraints

```
create trigger timeslot_check1 after insert on section
referencing new row as nrow
for each row
when (nrow.time_slot_id not in (
        select time_slot_id
        from time_slot))      /* time_slot_id not present in time_slot */
begin
        rollback
end;
```

Trigger Example (cont)

- create trigger timeslot_check2 after delete on timeslot
  referencing old row as orow
  for each row
  when (orow.time_slot_id not in (
  select time_slot_id
  from time_slot)
  /* last tuple for time_slot_id deleted from time_slot */
  and orow.time_slot_id in (
  select time_slot_id
  /* and time_slot_id still referenced from section */
  from section))
  begin
  rollback
  end;

Triggering Events and Actions in SQL

- Triggering event can be insert, delete or update
- Triggers on update can be restricted to specific attributes
  - E.g., after update of takes on grade
- Values of attributes before and after an update can be referenced
  - referencing old row as : for deletes and updates
  - referencing new row as : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blank grades to null.

```
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
when (nrow.grade = ' ')
begin atomic
    set nrow.grade = null;
end;
```

Trigger to Maintain credits_earned value

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
and (orow.grade = 'F' or orow.grade is null)
begin atomic
        update student
        set tot_cred = tot_cred +
                (select credits
                 from course
                 where course.course_id= nrow.course_id)
        where student.id = nrow.id;
end;
```

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use for each statement instead of for each row
  - Use referencing old table or referencing new table to refer to temporary tables (called transition tables) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

When Not to Use Triggers

- Triggers were used earlier for tasks such as
  - Maintaining summary data (e.g., total salary of each department)
  - Replicating databases by recording changes to special relations (called change or delta relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

When Not to Use Triggers

- Risk of unintended execution of triggers, for example, when
  - Loading data from a backup copy
  - Replicating updates at a remote site
  - Trigger execution can be disabled before such actions.
- Other risks with triggers:
  - Error leading to failure of critical transactions that set off the trigger
  - Cascading execution

End of Chapter 5