CS422 Principles of Database Systems
Stored Procedures and Triggers

Chengyu Sun
California State University, Los Angeles

Stored Procedures

- User-created functions that are stored in the database just like other schema elements
- Procedure vs. Function
  - A procedure does not return any value, while a function does
  - In PostgreSQL, a procedure is simply a function that returns `void`

Example: Hello World

```
create function hello() returns void as $$
begin
    raise notice 'Hello world in PL/pgSQL';
end;
$$ language plpgsql;
```

Example: Add10

```
create function add10( a integer ) returns integer as $$
declare
    b   integer;
begin
    b := 10;
    return a + b;
end;
$$ language plpgsql;
```

Procedural Language (PL)

- A programming language for writing stored procedures
- Usually based on some existing language like SQL, Java, C#, Perl, Python …
  - E.g. PL/SQL, PL/Java, PL/Perl …

Why Use Stored Procedures?

- Performance
  - compiled and optimized code
  - Save communication overhead
- Security
  - Access control
  - Less data transferred over the wire
- Simplify application code
- Triggers for data integrity

Why Not To Use Stored Procedures?

- Portability
- PL are generally more difficult to develop and maintain than conventional programming languages
  - Less language features
  - Less tool support

PostgreSQL PL/pgSQL

- SQL + things you would expect from a conventional programming language:
  - Variables and types
  - Control flow statements
  - Functions
- http://www.postgresql.org/docs/8.4/interactive/plpgsql.html

Elements of a Programming Language

- Comments
- Literals
- Variables and Types
- Operators and expressions
- Statements
  - Special statements, e.g. input and output
- Functions
- Classes
- Packages

Elements of PL/pgSQL

Comments Literals	Same as in SQL
Variables and types Operators and expression	Mostly the same as in SQL, with a few special types and operators
Statements Functions	
Classes Packages	Not supported

Basic Function Syntax

```
CREATE [OR REPLACE] FUNCTION name ( parameters )
      RETURNS type AS $$
DECLARE
      declarations
BEGIN
      statements
END;
$$ LANGUAGE plpgsql;


DROP FUNCTION name ( argtype [, ...]);
```

Examples: Basics

- Implement a function that takes two integer parameters and displays the sum

Basic Syntax and Output

- Variable declaration
- The assignment operator **:=**
- `RAISE`
  - Levels: DEBUG, LOG, INFO, NOTICE, WARNING, EXCEPTION
  - Format output with %
  - http://www.postgresql.org/docs/8.4/interactive/plpgsql-errors-and-messages.html

Examples: Statements

- Implement a function that returns the name of a student given the student's id
- Implement a function that calculates factorial

Naming Conventions

- We want to avoid using the same names for variables and table columns
- A simple naming convention:
  - Prefix parameters with **p_**
  - Prefix local variable with **l_**
  - Prefix package global variable with **g_**

SELECT…INTO

```
SELECT select_list INTO variable_list
    FROM table_list
    [WHERE condition]
    [ORDER BY order_list];
```

- SELECT result must be a *single row*.

Branch Statement

```
IF condition1 THEN
    statements1
ELSIF condition2 THEN
    statements2
ELSE
    statements3
END IF;
```

- NOTE: don't forget the semicolon (;) after END IF.

Loop Statements

```
LOOP
    statements
    EXIT WHEN condition;
    statements
END LOOP;
```

```
WHILE condition LOOP
    statements
END LOOP;
```

```
FOR loop_variable IN [REVERSE]
lower_bound..upper_bound LOOP
    statements
END LOOP;
```

Examples: Types

◈Implement a function that randomly returns two student records

Special Types

◈%TYPE
◈%ROWTYPE
◈Each table defines a *type*
◈SetOf
◈Cursor

Cursor

◈An iterator for a collection of records
◈We can use a cursor to process the rows returned by a SELECT statement

Using Cursors

◈Declaration
◈OPEN
◈FETCH
◈CLOSE

Cursor - Open

◈OPEN *cursor* FOR *query*
◈The query is executed
◈The position of the cursor is *before* the first row of the query results

Cursor - Fetch

◈FETCH *cursor* INTO *target*
◈Move the cursor to the next row
◈Return the row

Cursor - Fetch

❖ `FETCH cursor INTO target`
❖ Move the cursor to the next row
❖ Return the row

Cursor - Fetch

❖ If there is no next row
  ▪ `target` is set to NULL(s)
  ▪ A special variable `FOUND` is set to `false`

Cursor - Close

❖ `CLOSE cursor;`

Query FOR Loop

FOR *target* IN *query* LOOP
  statements
END LOOP;

Examples: Cursor and Query For Loop

❖ Implement a function that randomly select 20% of the students
  ▪ Using cursor
  ▪ Using query for loop

About PL Programming

❖ It's just programming like you always do
❖ Debug code one small piece at a time
❖ Ask "How to do X" questions in the class forum
❖ Avoid re-implementing SQL
  ▪ For example, to compute max(price), use SELECT MAX(price) instead of using a cursor to iterate through all rows

Triggers

- Procedures that are automatically invoked when data is *changed*, e.g. INSERT, DELETE, and UPDATE.
- Common use of triggers
  - Enforcing data integrity constraints
  - Auditing
  - Replication

Trigger Example

- Create a trigger that audit the changes to the grades in the `enrollment` table

```
create table grade_changes (
        enrollment_id        integer,
        old_grade_id         integer,
        new_grade_id         integer,
        timestamp            timestamp
);
```

Trigger Example: Trigger

```
create trigger grade_audit
        after update
        on enrollment
        for each row
        execute procedure grade_audit();
```

Trigger Syntax

```
CREATE TRIGGER name
        { BEFORE | AFTER } { event [ OR ... ] }
        ON table
        [ FOR EACH { ROW | STATEMENT } ]
        EXECUTE PROCEDURE funcname ( arguments );


DROP TRIGGER name ON table;
```

Triggering Events

- INSERT
- DELETE
- UPDATE

Before or After

- `BEFORE`: trigger *fires* before the triggering event
- `AFTER`: trigger fires after the event

Statement Trigger vs. Row Trigger

◈ Statement Trigger
  - Default
  - Fires once per statement
◈ Row Trigger
  - FOR EACH ROW
  - Fires once per row

Trigger Example: Function

```
create or replace function grade_audit()
        returns trigger as $$
begin
  if new.id = old.id and new.grade_id <> old.grade_id then
     insert into grade_changes values (
        new.id, old.grade_id, new.grade_id,
        current_timestamp );
  end if;
  return null;
end;
$$ language plpgsql;
```

About Trigger Functions

◈ No parameters
◈ Return type must be trigger
◈ Special variables
  - NEW, OLD
  - Others:
    http://www.postgresql.org/docs/8.4/interactive/plpgsql-trigger.html

Return Value of a Trigger Function

◈ Statement triggers and after-row triggers should return NULL
◈ Before-row trigger can return NULL to skip the operation on the current row
◈ For before-row insert and update triggers, the returned row becomes the row that will be inserted or will replace the row being updated

Examples: Enforce Data Integrity Constraints

◈ Create a trigger to enforce the constraint that the size of a Database class cannot exceed 30
  - RAISE EXCEPTION would abort the statement