CSC 355 Database Systems
Lecture 15

Eric J. Schwabe

School of Computing, DePaul University

Spring 2020

Today:

- Database Programming in PL/SQL
- Back to Triggers

Database Programming

- Three main approaches:
  1. Embed database commands in a general programming language
  2. Create a library of database functions in an API (e.g., JDBC)
  3. Design a general programming language that includes database commands (e.g., PL/SQL)
- First two approaches can suffer from *impedance mismatch*

PL/SQL

- PL/SQL is Oracle's version of the SQL/PSM ("Persistent Stored Modules") standard
- PL/SQL is a procedural programming language that includes SQL – it can:
  - create and issue SQL statements
  - store and process the results of queries
  - define triggers to respond to database events

Database Programming in PL/SQL

- Three places PL/SQL code can go:
  1. Within a trigger that is executed in response to database events
  2. Within a procedure or function that is executed when called by name
  3. Within an *anonymous block* that is executed directly by a user

Anonymous Block

- Will be executed directly (like a script):

```
declare
        -- variable declarations
begin
        -- PL/SQL statements to execute
        -- each statement must end with a semicolon
exception
        -- exception handling (optional)
end;
/
```

Output

- To display output:

dbms_output.put_line(*string*);

- Output buffer is displayed in Dbms Output tab when anonymous block is completed
  - Use View → Dbms Output and '+' to open tab
- Concatenation of strings uses ||

Variables

- All variables must be declared:

  varName dataType [:= *initialValue*];

- SQL data types are available (e.g., number, char, varchar2), plus binary_integer and boolean
- Assignments use :=, and PL/SQL has typical arithmetic operations

Variables

- Only one variable can be declared per line, but variable types can be given in terms of the domain of another variable or attribute:

  varName otherVar%type;
  varName TABLE.Attribute%type;

- Can use *substitution variables* (e.g., &X) to prompt user for values

Branching

- if-then:

  if *condition* then

  …*'true' statements*…

  end if;

- if-else:

  if *condition* then

  …*'true' statements*…

  else

  …*'false' statements*…

  end if;

Branching

- if-elsif:

if *condition1* then
    … *'t' statements*…
elsif *condition2* then
    … *'f-t' statements*…
elsif *condition3* then
    … *'f-f-t' statements*…
(… *as many times as needed*…)
else
    … *'all f' statements*…
end if;

- case:

case *variable*
when *value1* then
    … *'value1' statements*…
when *value2* then
    … *'value2' statements*…
(… *as many times as needed*…)
else
    … *'nomatch' statements*…
end case;

Loops

- General loop:

  loop
  …*loop body*…
  end loop;

- Repeats until exit; is executed in loop body

- While loop:

  while *condition* loop
  …*loop body*…
  end loop;

- Repeats until *condition* is false

Loops

- For loop:

for *variable* in [reverse] *lower..upper* loop
…loop body…
end loop;

- Can only increment/decrement by one
- lower always appears before upper in header

Incorporating SQL Queries

- Result of a query can be stored in a set of variables by adding INTO clause to query
  - Variable types must match attribute types
  - Query must return a single record

SELECT *list of attributes*
INTO *list of variables*
FROM *list of tables*
...

Cursors

- A *cursor* represents a pointer into a set of records returned by a query

  cursor *name* is *query*;

- cursor *name* can be used to iterate through the records returned by *query*

Cursor Commands/Expressions

- open *name*; -- initializes to beginning of set
- fetch *name* into *variableList*;
  -- reads the next record into the variables
- close *name*; -- closes the cursor
- *name*%found
  -- true if last call to fetch succeeded
- *name*%rowcount
  -- number of records successfully fetched

Records

- Can declare a record with the same structure as a table row (fields are table attributes)

  *recordName TABLE%*rowtype;

- Can select a row of a table directly into a record, then access individual fields with

  recordName.Attribute

Cursor For Loop

- To iterate through all of the rows returned by a query:

for *recordName* in *cursorName*
    …loop body…
end loop;

- The needed record must be declared, but open/fetch/close can be omitted in this loop

Database Programming References

- Ullman/Widom, Section 9.4
- Oracle's "PL/SQL User's Guide and Reference", Chapters 1-6 (link posted)
- Stanford Infolab's "Using Oracle PL/SQL" (link posted)
- Murach's "Oracle SQL and PL/SQL for Developers (second edition)", Chapters 13 and 16 (recommended text)

Oracle Trigger Syntax

CREATE [OR REPLACE] TRIGGER *Name*
BEFORE/AFTER
    INSERT OR DELETE OR UPDATE [OF *Attribute*] ON *TABLE*
[REFERENCING
    OLD AS *OldName*
    NEW AS *NewName*]
[FOR EACH ROW]
[WHEN (*condition*)]
DECLARE
    …variable declarations…
BEGIN
    …PL/SQL statements…
END;
/

Trigger Examples

- Salary Cap:
  - Trigger cancels any operation that causes the company's total budget for salaries to exceed $1,000,000 (statement-level)
- Departmental Budgets:
  - Trigger maintains current totals of the salaries of all employees in each department (row-level)

Next:

- Trigger Examples