Oracle PL/SQL

PL/SQL

- Originally modeled after ADA
  - Created for Dept. of Defense
- Allows expanded functionality of database applications
- Continues to improve with each new database release

PL/SQL

- Features
  - Tight integration with SQL
    - Supports data types, functions, pseudo-columns, etc.
  - Increased performance
    - A block of statements sent as a single statement
  - Increased productivity
    - Same techniques can be used with most Oracle products
  - Portability
    - Works on any Oracle platform
  - Tighter security
    - Users may access database objects without granted privileges

PL/SQL Programs

- Declaration section (optional)
  - Any needed variables declared here
- Executable or begin section
  - Program code such as statements to retrieve or manipulate data in a table
- Exception section (optional)
  - Error traps can catch situations which might ordinarily crash the program

PL/SQL Block Structure

```
DECLARE
   variables used in this program unit are declared here
BEGIN
   executable sections containing PL/SQL and SQL statements
EXCEPTION
   statements for dealing with errors
END;
/
```

PL/SQL Variables

- Variables are local to the code block
- Names can be up to 30 characters long and must begin with a character
- Declaration is like that in a table
  - Name then data type the semi-colon
  - Can be initialized using := operator in the declaration
  - Can be changed with := in the begin section
  - Can use constraints
- Variables can be composite or collection types
  - Multiple values of different or same type

Common PL/SQL Data Types

- CHAR (max_length)
- VARCHAR2 (max_length)
- NUMBER (precision, scale)
- BINARY_INTEGER – more efficient than number
- RAW (max_length)
- DATE
- BOOLEAN (true, false, null)
- Also LONG, LONG RAW and LOB types but the capacity is usually less in PL/SQL than SQL

PL/SQL Variable Constraints

- NOT NULL
  - Can not be empty
- CONSTANT
  - Can not be changed

PL/SQL Variables Examples

```
Age number;
Last char ( 10 );
DVal Date := Sysdate;
SID number not null;
Adjust constant number := 1;
CanLoop boolean := true
```

Predefined Exceptions

- INVALID_NUMBER (ORA-01722)
  - Attempted to store non-numeric data in a variable with a numeric data type
- NO_DATA_FOUND (ORA-01403)
  - Query resulted in no rows being found
- NOT_LOGGED_ON (ORA-01012)
  - Not currently connected to an Oracle database
- TOO_MANY_ROWS (ORA-01422)
  - A SELECT INTO statement returned more than one row

Predefined Exceptions (cont.)

- DUP_VALUE_ON_INDEX (ORA-00001)
  - Value inserted for a primary key is not unique
- VALUE_ERROR (ORA-06502)
  - The value being placed in a variable is the wrong length or data type
- ZERO_DIVIDE (ORA-01476)
  - An attempt was made to divide a number by zero

Structure of Exception Section

```
EXCEPTION
   WHEN [exception name] THEN
      action to take when exception occurs;
   WHEN [exception name] THEN
      action to take when exception occurs;
   WHEN OTHERS THEN
      action to take if any other errors occur;
END;
```

Conditional Structures

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSIF
  - An alternative to nested IF-THEN_ELSE

IF-THEN Structure

```
IF [T/F condition] THEN
   statements to perform when condition is true;
END IF;
```

IF-THEN-ELSE Structure

```
IF [T/F condition] THEN
   statements to perform when condition is true;
ELSE
   statements to perform when condition is false;
END IF;
```

IF-THEN-ELSIF Structure

```
IF [first T/F condition] THEN
   statements to perform when first condition is true;
ELSIF [second T/F condition] THEN
   statements to perform when second condition is true;
ELSIF [third T/F condition] THEN
   statements to perform when third condition is true;
ELSE
   statements to perform when all conditions are false;
END IF;
```

Stored Procedures

```
CREATE PROCEDURE ProcedureName(parameter1 datatype, parameter2 datatype, ...) AS
   Additional declarations of local variables
BEGIN
   executable section
EXCEPTION
   Optional exception section
END;
```

Stored Procedures

- The first line is called the Procedure Specification
- The remainder is the Procedure Body
- A procedure is compiled and loaded in the database as an object
- Procedures can have parameters passed to them

Stored Procedures

- Run a procedure with the PL/SQL EXECUTE command
- Parameters are enclosed in parentheses

Stored Functions

- Like a procedure except they return a single value

Triggers

- Associated with a particular table
- Automatically executed when a particular event occurs
  - Insert
  - Update
  - Delete
  - Others

Triggers vs. Procedures

- Procedures are explicitly executed by a user or application
- Triggers are implicitly executed (fired) when the triggering event occurs
- Triggers should not be used as a lazy way to invoke a procedure as they are fired <u>every</u> time the event occurs

Triggers

```
CREATE TRIGGER TriggerName
BEFORE [AFTER] event[s] ON TableName
[FOR EACH ROW]
DECLARE
   Declaration of any local variables
BEGIN
   Statements in Executable section
EXCEPTION
   Statements in optional Exception section
END;
/
```

Triggers

- The trigger specification names the trigger and indicates when it will fire
- The trigger body contains the PL/SQL code to accomplish whatever task(s) need to be performed

Triggers

```
CREATE TRIGGER TriggerName
BEFORE [AFTER] event[s] ON TableName
[FOR EACH ROW]
DECLARE
   Declaration of any local variables
BEGIN
   Statements in Executable section
EXCEPTION
   Statements in optional Exception section
END;
/
```

Triggers Timing

- A triggers timing has to be specified first
  - Before (most common)
    - Trigger should be fired before the operation
      - i.e. before an insert
  - After
    - Trigger should be fired after the operation
      - i.e. after a delete is performed

Trigger Events

- Three types of events are available
  - DML events
  - DDL events
  - Database events

DML Events

- Changes to data in a table
  - Insert
  - Update
  - Delete

DDL Events

- Changes to the definition of objects
  - Tables
  - Indexes
  - Procedures
  - Functions
  - Others
    - Include CREATE, ALTER and DROP statements on these objects

Database Events

- Server Errors
- Users Log On or Off
- Database Started or Stopped

Trigger DML Events

- Can specify one or more events in the specification
  - i.e. INSERT OR UPDATE OR DELETE
- Can specify one or more columns to be associated with a type of event
  - i.e. BEFORE UPDATE OF SID OR SNAME

Table Name

- The next item in the trigger is the name of the table to be affected

Trigger Level

- Two levels for Triggers
  - Row-level trigger
    - Requires FOR EACH ROW clause
      - If operation affects multiple rows, trigger fires once for each row affected
  - Statement-level trigger
  - DML triggers should be row-level
  - DDL and Database triggers should not be row-level

Event Examples

```
Example 1:
CREATE TRIGGER NameChange
BEFORE UPDATE OF STUDENT_FIRST_NAME, STUDENT_LAST_NAME ON STUDENT
FOR EACH ROW

Example 2:

CREATE TRIGGER AlterStudent
AFTER INSERT OR UPDATE OR DELETE ON STUDENT
FOR EACH ROW

Example 3:

CREATE TRIGGER ErrorLog
AFTER SERVERERROR ON DATABASE

Example 4:

CREATE Trigger TrackChanges
AFTER CREATE ON SCHEMA
```

Triggers

- Conditions Available So Multiple Operations Can Be Dealt With In Same Trigger
  - Inserting, Updating, Deleting
- Column Prefixes Allow Identification Of Value Changes
  - New, Old

Triggers Exceptions

- EXCEPTION Data Type Allows Custom Exceptions
- RAISE Allows An Exception To Be Manually Occur
- RAISE_APPLICATION_ERROR Allows Termination Using A Custom Error Message
  - Must Be Between -20000 and -20999
  - Message Can Be Up to 512 Bytes

Cursors

- Cursors Hold Result of an SQL Statement
- Two Types of Cursors in PL/SQL
  - Implicit – Automatically Created When a Query or Manipulation is for a Single Row
  - Explicit – Must Be Declared by the User
    - Creates a Unit of Storage Called a Result Set

Cursors

Result Set

MIS380	DATABASE DESIGN	4	
MIS202	INFORMATION SYSTEMS	3	<Cursor
MIS485	MANAGING TECHNOLOGY	4	
MIS480	ADVANCED DATABASE	4	

Cursors

- Declaring an Explicit Cursor

  CURSOR CursorName IS SelectStatement;

- Opening an Explicit Cursor

  OPEN CursorName;

- Accessing Rows from an Explicit Cursor

  FETCH CursorName INTO RowVariables;

Cursors

- Declaring Variables of the Proper Type with %TYPE

  VarName TableName.FieldName%TYPE;
- Declaring Variables to Hold An Entire Row

  VarName CursorName%ROWTYPE;
- Releasing the Storage Area Used by an Explicit Cursor

  CLOSE CursorName;

Iterative Structures

- LOOP … EXIT … END LOOP
  - EXIT with an If Avoids Infinite Loop
- LOOP … EXIT WHEN … END LOOP
  - Do Not Need An If to Control EXIT
- WHILE … LOOP … END LOOP
  - Eliminates Need for EXIT
- FOR … IN … END LOOP
  - Eliminates Need for Initialization of Counter

Cursor Control With Loops

- Need a Way to Fetch Repetitively
- Need a Way to Determine How Many Rows to Process With a Cursor
  - Cursor Attributes
    - **CursorName%ROWCOUNT – Number of Rows in a Result Set**
    - **CursorName%FOUND – True if a Fetch Returns a Row**
    - **CursorName%NOTFOUND – True if Fetch Goes Past Last Row**

Cursor For Loop

- Processing an Entire Result Set Common
- Special Form of FOR … IN to Manage Cursors
- No Need for Separate OPEN, FETCH and CLOSE statements
- Requires %ROWTYPE Variable

Thank You

- www.srinimf.com