Database Management Systems
L11

Umass Boston
Summer 2023
Cristina Maier

Stored Procedures

- Execute on DBMS
  - Encapsulate application logic
  - Execute close to the data
  - Allows reuse of common functionality by different clients
- Vendors introduced their own procedural extensions
  - E.g.: Oracle's PL/SQL

PL/SQL
(Procedural Language SQLP)

- SQL Procedural extension developed by Oracle
  - Most prominent procedural language
    - Another language is T-SQL from Microsoft (SQL-Server)
- Basic program structure is a block
  - There could be nested blocks
- PL/SQL is not case-sensitive ()

PL/SQL Program Structure

```
DECLARE
variable_declarations
BEGIN
procedural_code
EXCEPTION
error_handling
END;
```

PL/SQL

- Declaration section
  - Optional section that starts with DECLARE keyword
  - Defines all variables, cursors, subprograms that are used in the program
- Executable Commands section
  - Code between BEGIN and END
  - Contains the PL/SQL statements of the program
  - Must have at least one line of code (could be NULL command to indicate that nothing is to execute)
- Exception handling
  - Starts with keyword EXCEPTION
  - Optional section that could contain code that handles exceptions from the program

Simple example SQL PLUS

```
DECLARE
    message varchar(20):='Hello world!';
BEGIN
    dbms_output.put_line(message);
END;
/
```

Prints on screen Hello world!

/ on the last line to run the code

Before you run this, to ensure the output goes to the screen run

SET SERVER OUTPUT ON

PL/SQL Syntax

- Declare variable
  - varname datatype:=value
  - e.g. message varchar(20):=‘hello'
  - e.g. age int:=20
- Assignment
  - A :=B + C
- Data types
  - Can use Oracle data types: varchar, number etc.
-

PL/SQL Syntax

- Branch

IF condition THEN statements;
ELSIF (condition) statements;
ELSIF …
ELSE statements
END IF;

Store Procedures (cont.)

- PL/SQL allows to directly refer to a column type

tablename.columnname%TYPE

E.g. SAILORS.SNAME%TYPE

- It's also possible to define a row type

tablename%ROWTYPE

- Declare a variable varname TYPE

sailor_rec SAILORS%ROWTYPE

- We can later refer to individual fields using column names

```
DBMS_OUTPUT.PUT_LINE('Name: ' || sailor_rec.name || 'Age:' || sailor_rec.age);
```

- Note the || is used to concatenate strings (similar to + in Java)

PL/SQL Syntax

- Branch

```
IF condition THEN
    statements;
ELSIF (condition)
    statements;
ELSIF …
ELSE
     statements
END IF;
```

Example with Branch

```
DECLARE
    A NUMBER(6) := 10;
    B NUMBER(6);
BEGIN
    A := 23;
    B := A * 5;
    IF A < B THEN
      DBMS_OUTPUT.PUT_LINE(A || ' is less than ' || B);
    ELSE
      DBMS_OUTPUT.PUT_LINE(B || ' is less-or-equal than ' || A);
    END IF;
END;
```

- OUTPUT IS: 23 is less than 115

Example #2 with Branch

```
DECLARE
   NGRADE NUMBER;
   LGRADE CHAR(2);
BEGIN
   NGRADE := 82.5;
   IF NGRADE > 95 THEN
        LGRADE := 'A+';
   ELSIF NGRADE > 90 THEN
        LGRADE := 'A';
   ELSIF NGRADE > 85 THEN
        LGRADE := 'B+';
   ELSIF NGRADE > 80 THEN
        LGRADE := 'B';
   ELSE
        LGRADE := 'F';
   END IF
END;
```

LOOPS

```
LOOP
        statements
IF condition THEN
        EXIT;
END IF;
         statements
END LOOP;
```

LOOPS

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

Loop Example

```
DECLARE
   J NUMBER(6);
BEGIN
   J := 1;
   LOOP
      DBMS_OUTPUT.PUT_LINE('J= ' || J);
      J := J + 1;
      EXIT WHEN J > 5;
      DBMS_OUTPUT.PUT_LINE('J= ' || J);
   END LOOP;
END;
```

Output = ?

Loop Variants

```
WHILE condition
LOOP
    various_statements
END LOOP;
```

```
FOR counter IN startvalue .. endvalue
LOOP
  various_statements
END LOOP;
```

For Loop Example

```
BEGIN
   FOR K IN 1..5
   LOOP
      DBMS_OUTPUT.PUT_LINE('K= ' || K);
   END LOOP;
END;
```

Using SQL Statements

- Data can be manipulated (DML = Data Manipulation Language)

```
DECLARE
   SID NUMBER(6);
BEGIN
   SID := 20;
   INSERT INTO Sailors (sid,name)VALUES (SID,'Rusty');
   SID := SID + 1;
   INSERT INTO Sailors (sid,name)VALUES (SID,'Yuppy');
END;
```

SQL Statement - retrieving data

- Single row result

SELECT selectfields INTO declared_variables

FROM table_list WHERE search_criteria;

```
DECLARE
   VAR_NAME Sailors.sname%TYPE;
   VAR_AGE Sailors.age%TYPE;
BEGIN
   SELECT name, age INTO VAR_NAME, VAR_AGE
  FROM SailorsWHERE SID = 10;
  DBMS_OUTPUT.PUT_LINE('Age of ' || VAR_NAME || ' is ' || VAR_AGE);
END;
```

SQL Statement - retrieving data

- Multi row result: a structure called **CURSOR** is needed!

CURSOR cursorname IS SELECT_statement;

OPEN cursorname;
FETCH cursorname INTO variable_list;

CLOSE cursorname;

Example with Cursor

```
DECLARE
   S Sailors%ROWTYPE;
   CURSOR SAILORCURSOR IS
       SELECT * FROM Sailors;
BEGIN
   OPEN SAILORCURSOR;
   LOOP
      FETCH SAILORCURSOR INTO S;
      EXIT WHEN SAILORCURSOR%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('AGE OF ' || S.sname || ' IS ' || S.age);
   END LOOP;
   CLOSE SAILORCURSOR;
END;
```

Cursor attributes

- %NOTFOUND: evaluates to TRUE when cursor has no more rows to read. FALSE otherwise.
- %FOUND: evaluates to TRUE if last FETCH was successful. FALSE otherwise
- %ROWCOUNT: returns the number of rows that the cursor has already fetched from the database
- %ISOPEN: returns TRUE if cursor is already open. FALSE otherwise.

Declaring a Stored Procedure

```
CREATE OR REPLACE
PROCEDURE procedure_name ( parameters ) IS
  variable declarations
BEGIN
  procedure_body
END;
```

- Params can be IN, OUT, INOUT. Default is IN

```
CREATE OR REPLACE
PROCEDURE SUM_AB (A INT, B INT, C OUT INT) IS
BEGIN
  C := A + B;
END;
```

Declaring a Function

```
CREATE OR REPLACE
FUNCTION function_name (function_params) RETURN return_type IS
    variable declarations
BEGIN
    function_body
    RETURN something_of_return_type;
END;
```

- Example

```
CREATE OR REPLACE
FUNCTION ADD_TWO (A INT,B INT) RETURN INT IS
BEGIN
   RETURN (A + B);
END;
```

Exceptions

- Exceptions defined per block (similar to Java)
  - Each BEGIN … END has its own exception handling
  - If blocks are nested, exceptions are handled in an "inside to outside" fashion
  - If no block on the nesting handles the exception, a runtime error occurs
- There are multiple types of exceptions
  - Named system exceptions (most frequent) - we will only cover these ones
  - Unnamed system exceptions
  - User-defined exceptions

Exceptions

```
DECLARE ...
BEGIN EXCEPTION
  WHEN ex_name1 THEN
       error handling statements
  WHEN ex_name2 THEN
       error handling statements
   ...
  WHEN Others THEN
       error handling statements
END;
```

Named Systems Exceptions

Named System Exceptions

Exception Name	Reason	Error Number
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.	ORA-06511
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor or fetch data from a cursor that is not opened.	ORA-01001
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.	ORA-01403
TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.	ORA-01422
ZERO_DIVIDE	When you attempt to divide a number by zero.	ORA-01476

PostgreSQL

Questions?