Chapter 8C

Objectives: to learn

• to create and use
  PL/SQL
  – triggers
  – stored procedures
  – Functions

Outline

PL/SQL

• Anonymous Blocks
• Triggers
• Procedures
• Functions

Procedural SQL - Overview

• Shortcomings of SQL – 4GL
  – SQL does not support conditional execution
  – Cannot isolate critical code to allow:
    • All applications access shared code
    • Better maintenance and logic control
• Persistent stored module (PSM) is a block of code containing:
  – Standard SQL statements
  – Procedural extensions
• Oracle’s combination of extensions with SQL is called Procedural Language SQL (PL/SQL).

Procedural SQL

• A PSM is stored and executed on the DBMS server
• Modules are executed by DBMS when it is
  – triggered(fired) or
  – invoked by the end user
• Oracle allows the following PSM types
  – Triggers
  – Stored procedures
  – PL/SQL functions (invoked by other procedural code – not SQL statements)
• Oracle allows a type that is not stored/persistent
  – Anonymous PL/SQL blocks

Procedural SQL

• Procedural Language/SQL structure
  Declare ...
  Begin ...
  End;

• Allows 3GL structures within the Begin End.
  – If ...Else ... End If
  – While (condition) Loop ... End Loop
• Allows variables to be declared and used.
Procedural SQL

- Declared variables within Triggers, Functions, and Procedures must be of the following types

<table>
<thead>
<tr>
<th>PL/SQL Basic Data Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVAR</td>
<td>Character values of a fixed length (for example: W_NAME, W_PHONE)</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>Variable length character values (for example: W_NAME, W_PHONE)</td>
</tr>
<tr>
<td>NUMBER</td>
<td>Numeric values (for example: W_PRICE)</td>
</tr>
<tr>
<td>DATE</td>
<td>Date values (for example: W_DATE)</td>
</tr>
<tr>
<td>SPOOL</td>
<td>Binary data type from a variable that you declared previously or from an attribute of a database table (for example: V_W_CODE, V_W_NAME)</td>
</tr>
</tbody>
</table>

Procedural SQL - Testing

- SHOW ERRORS
  - Can help diagnose errors found in PL/SQL blocks
  - Yields additional debugging information whenever an error is generated after creating or executing a PL/SQL block

- SET SERVEROUTPUT ON/OFF
  - Enables the client console (SQLPlus) to receive messages from the server side
  - Use DBMS_OUTPUT.PUT_LINE function from within a PL/SQL block

Procedural SQL – Anonymous Blocks

Anonymous PL/SQL Blocks

(For brief, non-production usage. The blocks are not named)

```sql
BEGIN
    insert into vendor values (25676,'MicroSoft Corp.', 'Bill Gates', '765','546-484','WA','N');
END;
/
```

```sql
SET serveroutputon
BEGIN
    insert into vendor values (25772,'Clue Store', 'Issac Hayes', '456', '323-3009', 'VA', 'N');
    DBMS_OUTPUT.PUT_LINE('Vendor Clue Store added');
END;
/```

Procedural SQL

- While Loop syntax
  - WHILE condition LOOP
    - PL/SQL statements;
  - END LOOP

- Selecting values & saving them with INTO
  - select count(p_code) into v_num from product;

- Variables must be declared
Procedural SQL - Anonymous Block

```sql
DECLARE
    V_P1 NUMBER(3) := 0;
    V_P2 NUMBER(3) := 10;
    V_NUM NUMBER(2) := 0;
BEGIN
    WHILE V_P2 < 300 LOOP
        SELECT COUNT(P_CODE) INTO V_NUM FROM PRODUCT
        WHERE P_PRICE BETWEEN V_P1 AND V_P2;
        DBMS_OUTPUT.PUT_LINE('There are ' || V_NUM || ' Products with price between ' || V_P1 || ' and ' || V_P2);
        V_P1 := V_P2 + 1;
        V_P2 := V_P2 + 50;
    END LOOP;
END;
/```

Procedural SQL - Triggers

- **A trigger is procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a data manipulation event.**
  - A trigger is always invoked before or after a data row is selected, inserted, or updated.
  - A trigger is always associated with a database table.
  - Each database table may have one or more triggers.
  - A trigger is executed as part of the transaction that triggered it.

- **Creating Triggers**
  - **Parts of a Trigger** definition:
    - Triggering timing: BEFORE or AFTER the event
    - Triggering event: INSERT, UPDATE, DELETE
    - Triggering level:
      - **Statement-level trigger**
      - **Row-level trigger**
  - Triggering action
  - Deleting a Trigger
    - DROP TRIGGER `trigger_name`

- **Procedural SQL code is automatically invoked by RDBMS on data manipulation event**
- **Roles of triggers**
  - Used to enforce constraints that cannot be enforced at the design and DB implementation levels.
  - Add functionality by automating critical actions and providing appropriate warnings and suggestions for remedial action.
  - Used to update table values, insert records in tables, and call other stored procedures.
  - To add processing power to the RDBMS and to the database system
Procedural SQL - Triggers

• SYNTAX –
  CREATE OR REPLACE TRIGGER <trigger_name>  
  [BEFORE/AFTER] [DELETE/INSERT/UPDATE] OF  
  <column_name> ON <table_name>  
  [FOR EACH ROW]  
  BEGIN  
  <PL/SQL instructions;>
  END;

• Example -
  Create or replace trigger TRG_PRODUCT_REORDER  
  after insert or update of p_qoh, p_min on Product  
  Begin  
  Update Product set p_reorder = 1  
  where p_qoh <= p_min;  
  End;

Procedural SQL - Triggers – version 1

• Problem: either P_QOH or P_min can affect the reorder flag. Solution:
  Create or replace trigger TRG_PRODUCT_REORDER  
  after insert or update of p_qoh, p_min on Product  
  Begin  
  Update Product set p_reorder = 1  
  where p_qoh <= p_min;  
  End;

Procedural SQL - Triggers – version 2
Procedural SQL - Triggers – version 3

- Problem: P_QOH might increase, not decrease.
  Example revisited using states
Create or replace trigger TRG_PRODUCT_REORDER
before insert or update of p_qoh, p_min on Product
for each row
Begin
  If :new.p_qoh < :new.p_min then
    :new.p_reorder := 1;
  else
    :new.p_reorder := 0;
  end if;
End;

Procedural SQL - Triggers – Customer Table

CREATE OR REPLACE TRIGGER TRG_LINE_CUS
AFTER INSERT ON LINE FOR EACH ROW
DECLARE
  W_CUS CUSTOMER.CUS_CODE%TYPE;
  W_TOT NUMBER:= 0; -- to compute total cost
BEGIN
  -- trigger fires up after an INSERT of a LINE & updates Cus_balance
  -- 1) get the CUS_CODE
  SELECT CUS_CODE INTO W_CUS
  FROM INVOICE
  WHERE INVOICE.INV_NUMBER = :NEW.INV_NUMBER;
  -- 2) compute the total of the current line
  W_TOT := :NEW.LINE_PRICE * :NEW.LINE_UNITS;
  -- 3) Update the CUS_BALANCE in CUSTOMER
  UPDATE CUSTOMER
  SET CUS_BALANCE = CUS_BALANCE + W_TOT
  WHERE CUS_CODE = W_CUS;
  DBMS_OUTPUT.PUT_LINE('*** Balance updated for customer:'||W_CUS);
END;
Procedural SQL - Stored Procedures

- A named collection of procedural and SQL statements which is stored in the database.

- Advantages
  - Substantially reduce network traffic and increase performance
  - No transmission of individual SQL statements over network
  - Help reduce code duplication by means of code isolation and code sharing
  - Minimize chance of errors and cost of application development and maintenance

Syntax to create a stored procedure

```
CREATE OR REPLACE PROCEDURE procedure_name
(argument IN, OUT, or IN OUT data-type, ...) IS/AS
DECLARE variable name data type;
BEGIN
PL/SQL or SQL statements;
END;
```

Syntax to invoke a stored procedure

```
>EXEC store_procedure_name (parameter, ...)
```

Create or replace procedure PROD_REORDER_SET as Begin
  Update Product set p_reorder = 1
  where p_qoh <= p_min;
  Update Product set p_reorder = 0
  where p_qoh > p_min;
End;

Running the procedure

```
>Exec prod_reorder_set;
```
Procedural SQL - Stored Procedures

• Example with parameters:
  >Create or replace procedure Prod_sale (code in varchar2, qtlsold in number)
  as Begin
    update product
    set p_qoh = p_qoh - qtlsold;
    where p_code = code;
  End;

• Running the procedure
  >Exec prod_sale(' WR3/TT3',20);

Procedural SQL – Stored Procedures

• Version 1: PRC_PROD_DISCOUNT
  Create or replace procedure prc_prod_discount
  As begin
    update product
    set p_discount = p_discount + .05
    where p_onhand >= p_min*2;
    dbms_output.put_line('Update finished');
  End;
 /

Procedural SQL – Stored Procedures

• Version 2: PRC_PROD_DISCOUNT
  Create or replace procedure prc_prod_discount
  (wpi in number) as
  Begin
    if ((wpi <= 0) or (wpi >= 1)) then
      DBMS_OUTPUT.PUT_LINE('Error, value must be
greater than 0 and less than 1')
    else
      Update Product
      set p_discount = p_discount + wpi
      where p_qoh >= p_min*2;
      DBMS_OUTPUT.PUT_LINE('**Update Finished**');
    End if;
  End;
  /

Procedural SQL - Stored Procedures

• Version 1: PRC_PROD_DISCOUNT
Create or replace procedure prc_prod_discount
As begin
update product
set p_discount = p_discount + .05
where p_onhand >= p_min*2;
dbms_output.put_line('Update finished');
End;
/

Procedural SQL - Stored Procedures

• Version 2: PRC_PROD_DISCOUNT
Create or replace procedure prc_prod_discount(wpi in number) as
Begin
if ((wpi <= 0) or (wpi >= 1)) then
  DBMS_OUTPUT.PUT_LINE('Error, value must be
  greater than 0 and less than 1')
else
  Update Product
  set p_discount = p_discount + wpi
  where p_qoh >= p_min*2;
  DBMS_OUTPUT.PUT_LINE('**Update Finished**');
End if;
End;
/
Procedural SQL – Stored Procedures

• PRC_CUS_ADD Stored Procedure
  Create or replace procedure prc_cus_add
  (w_ln in varchar, w_fn in varchar, w_init in varchar,
   w_ac in varchar, w_ph in varchar)
  As
  Begin
    -- note: procedure uses CUS_CODE_SEQ created earlier
    insert into customer(cus_code, cus_lname, cus_fname,
      cus_initial, cus_areacode, cus_phone)
    values (cus_code_seq.nextval, w_ln, w_fn, w_init,
      w_ac, w_ph);
    dbms_output.put_line ('Customer ' || w_ln|| ' added');
  end;
/

• PRC_INV_ADD
  Create or replace procedure prc_inv_add
  (w_cus_codein varchar2, w_date in date)
  As
  Begin
    insert into invoice values
      (inv_number_seq.nextval, w_cus_code, w_date);
    dbms_output.put_line('Invoice added');
  End;
/

• PRC_LINE_ADD
  Create or replace procedure prc_line_add
  (w_ln in number, w_p_code in varchar2, w_lu number)
  As
  Begin
    w_lp:= 0.00;
    select p_price into w_lp from product
    where p_code = w_p_code;
    insert into line values (inv_number.seq.currval,
      w_ln, w_p_code, w_lu, w_lp);
    dbms_output.put_line('Invoice line '||w_ln
      || ' added');
  End;
/
• Similar to a Stored Procedure except that it returns a single value, and can be embedded in another Trigger or Procedure.

• Syntax:
  
  CREATE FUNCTION function_name (argument IN data-type, ...) RETURN data-type [IS]
  BEGIN
    PL/SQL statements;
    ...
    RETURN (value or expression);
  END;

• Example
  
  CREATE or REPLACE FUNCTION cal_age (dob IN date) RETURN numeric
  AS
    BEGIN
      return round((sysdate - dob)/365,0);
    END;

• Function must be called from another stored procedure or trigger
**Procedural SQL - Testing**

- **SHOW ERRORS**
  - Can help diagnose errors found in PL/SQL blocks
  - Yields additional debugging information whenever error is generated after creating or executing a PL/SQL block

- **SET SERVEROUTPUT ON/OFF**
  - Enables the client console (SQLPlus) to receive message from the server side
  - Use DBMS_OUTPUT.PUT_LINE function from within a PL/SQL block

---

**Procedural SQL - Testing**

- **Verifying the Trigger Execution**
  - Create trigger
  - Test trigger/procedure
  - Display (select) before data
    - Example: update product set p_onhand = p_onhand;
  - Display (select) after data (maybe multiple tables)
  - Turn in all SOURCE CODE or ECHO plus the EXECUTION Spool.

---

**Chapter 8C Summary**

- **Persistent stored modules (PSM)** are a named collection of SQL statements and procedural code that is stored in the database.

- PL/SQL (Programming Language SQL) can be used to create triggers, stored procedures, and functions.

- A stored procedure is executed by the database and may be called by a trigger, a function, SqlPlus, or another stored procedure.