Tutorial 4

Revising PL/SQL

Introducing Procedures and Parameters

- A procedure is a block that is declared to be a procedure and usually has parameters.
- The parameters can be of three modes:
  - IN parameters are passed into the procedure.
  - OUT parameters are returned from the procedure.
  - IN OUT parameters are passed both ways.
- The procedure
  - The procedure is a block.
  - Before the BEGIN statement:
    The phrase
    PROCEDURE <procedure-name>
    ([{parameter-name mode datatype}]) IS
    BEGIN…
    END [<procedure-name>]

Get customer details

- It takes IN a customer ID and RETURNS a customer name and address.

```
PROCEDURE get_cust_details (
   cust_no   IN  builder2.customer.customer_id%TYPE,
   cust_name OUT builder2.customer.customer_name%TYPE,
   cust_addr OUT builder2.customer.customer_address%TYPE,
   status    OUT BOOLEAN) IS
   BEGIN
     SELECT customer_name,customer_address INTO
            cust_name,    cust_addr
     FROM builder2.customer
     WHERE customer_id = cust_no;
     status := true;
   EXCEPTION
   WHEN NO_DATA_FOUND THEN
      status := false;
   END get_cust_details;
```

Where is it?

- The procedure can be nested in an anonymous block.
  - It goes in the DECLARE section.
- The procedure can be CREATEd in the database.
  - In this way it becomes part of the persistent store and can be called by any user with the privileges to do so.

Example

- The following procedure
  - Takes a customer id
  - Requests the customer details from the builder2 table customer
  - If successful
    - Displays the customer details
  - Else
    - Displays an error message.

Calling it

- It is called getcust.sql
- It is stored on my C drive.
- To call it, I open SQL*Plus
  SQL>start c:getcust
- *There is a copy of it both on the web page and on WebCourses.*

```
DECLARE
   cnum builder2.customer.customer_id%TYPE;
   cname builder2.customer.customer_name%TYPE;
   caddr builder2.customer.customer_address%TYPE;
   status BOOLEAN;
   PROCEDURE get_cust_details (
    cust_no IN builder2.customer.customer_id%TYPE,
    cust_name OUT builder2.customer.customer_name%TYPE,
    cust_addr OUT builder2.customer.customer_address%TYPE,
     status OUT BOOLEAN) IS
   BEGIN
     SELECT customer_name,customer_address INTO cust_name,
   cust_addr
     FROM builder2.customer
     WHERE customer_id = cust_no;
     status := true;
   EXCEPTION
     WHEN NO_DATA_FOUND THEN
       status := false;
   END get_cust_details;
BEGIN
   get_cust_details(&cnum, cname, caddr, status);
   IF (status) THEN
     DBMS_OUTPUT.PUT_LINE(cnum || ' ' || cname ||
     ' ' || caddr);
   ELSE
     DBMS_OUTPUT.PUT_LINE('Customer ' ||
     cnum || ' not found');
   END IF;
END;
```

… and independently

```
CREATE OR REPLACE PROCEDURE get_cust_details (
cust_no IN builder2.customer.customer_id%TYPE,
cust_name OUT
builder2.customer.customer_name%TYPE,
cust_addr OUT
builder2.customer.customer_address%TYPE,
status OUT BOOLEAN) AS
BEGIN
  SELECT customer_name,customer_address
  INTO   cust_name, cust_addr
  FROM builder2.customer
  WHERE customer_id = cust_no;
  status := true;
EXCEPTION
WHEN NO_DATA_FOUND THEN
status := false;
END get_cust_details;
```

Exceptions and functions

Other exceptional circumstances

- When adding a student, we want to:
  - Be sure that there is a place left on the programme stage.
- When selling stock, we should only sell stock if the quantity available is less than the quantity we are seeking.
- These are BUSINESS rules, so we need to provide logic to cause an exception.

Why an exception?

- We could just use an ‘IF…THEN…ELSE…’ to avoid doing an update.
- The reason we use an exception, is so that we can propagate the error back to the calling environment.
  - This is because we almost NEVER use an application from SQL*Plus or iSQLPlus.

Named Programmer-Defined Exceptions

- Application-specific exceptions
  - E.g.
    - Negative balance in account
    - Team cannot play against itself
    - Cannot stock a negative number of items
- Programmer can define these errors, trap them and handle them.
- To do this:
  - Name the error
  - Check for the error and raise it
  - Handle the error in the EXCEPTION section

Add an orderline (builder2) (1 of 2)

```
-- Version 1 --
DECLARE
 onum   builder2.corder.corderno%type:=&onum;
 scode  builder2.stock.stock_code%type:=&scode;
 qtyreq
      builder2.corderline.quantityrequired%type:=&qtyreq;
 cur_q  builder2.stock.stock_level%type;
 lev    builder2.stock.reorder_level%type;
```

(2 of 2)

```
  begin
   select stock_level, reorder_level into cur_q, lev
   from builder2.stock  where stock_code = scode;
    if (cur_q > qtyreq) then
     update builder2.stock
       set stock_level = stock_level – qtyreq
     where stock_code = scode;
     if (cur_q - qtyreq) < lev then
         DBMS_OUTPUT.PUT_line
        ('Fallen below reorder quantity - reorder!');
     end if;
     insert into builder2.corderline values
      (qtyreq,onum, scode);
  else
  DBMS_OUTPUT.PUT_LINE(sysdate||' Order '||onum||
      ' Stock item '||scode||'Do not have enough to sell');
  end if;
exception
when others then
  rollback work;
  DBMS_OUTPUT.PUT_LINE(sysdate||' Failed when adding '||scode||' to order '||onum);
end;
```

VERSION 2 (1 of 2) New exception

```
DECLARE
 onum   builder2.corder.corderno%TYPE:= &onum;
 scode  builder2.stock.stock_code%TYPE:= &scode;
 qtyreq builder2.corderline.quantityrequired%TYPE:= &qtyreq;
 cur_q  builder2.stock.stock_level%TYPE;
 lev    builder2.stock.reorder_level%TYPE;
 not_enough_stock EXCEPTION;
begin
    SELECT stock_level, reorder_level
    INTO cur_q, lev
    FROM builder2.stock
    WHERE stock_code = scode;
```

2 OF 2 (new exception)

```
  IF (cur_q > qtyreq) THEN
    UPDATE builder2.stock
    SET stock_level = stock_level - qtyreq
    WHERE stock_code = scode;
    IF (cur_q - qtyreq) < lev THEN
 DBMS_OUTPUT.PUT_LINE('Fallen below reorder quantity - reorder!');
    END IF;
    INSERT INTO builder2.corderline VALUES (qtyreq,onum, scode);
  ELSE
       RAISE not_enough_stock;
  END IF;
EXCEPTION
WHEN not_enough_stock THEN
 DBMS_OUTPUT.PUT_LINE(sysdate||' Order '||onum||' Stock item '||scode||
 'Do not have enough to sell');
 ROLLBACK WORK;
WHEN OTHERS THEN
    ROLLBACK WORK;
   DBMS_OUTPUT.PUT_LINE(sysdate||' Failed when adding '||scode||' to
 order '||onum);
END;
```

Example 3 (1 of 3)standard exceptions

```
DECLARE
 onum   builder2.corder.corderno%TYPE:= :onum;
 scode  builder2.stock.stock_code%TYPE:= :scode;
 qtyreq      builder2.corderline.quantityrequired%TYPE:=
   :qtyreq;
 cur_q  builder2.stock.stock_level%TYPE;
 lev    builder2.stock.reorder_level%TYPE;
 not_enough_stock EXCEPTION;
begin
    SELECT stock_level, reorder_level
    INTO cur_q, lev
    FROM builder2.stock
    WHERE stock_code = scode;
```

2 of 3 Standard exceptions

```
IF (cur_q > qtyreq) THEN
    UPDATE builder2.stock
      SET stock_level = stock_level - qtyreq
    WHERE stock_code = scode;
    IF (cur_q - qtyreq) < lev THEN
   DBMS_OUTPUT.PUT_LINE('Fallen below reorder quantity -
  reorder!');
    END IF;
    INSERT INTO builder2.corderline VALUES (qtyreq,onum,
  scode);
 ELSE
      RAISE not_enough_stock;
 END IF;
```

3 of 3 standard exceptions

```
EXCEPTION
  WHEN not_enough_stock THEN
   DBMS_OUTPUT.PUT_LINE(sysdate||' Order '||onum||' Stock item
   '||scode||
   'Do not have enough to sell');
   ROLLBACK WORK;
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE(sysdate||' Either the order '||onum||' or
   the Stock item '||scode||
   ' does not exist');
  WHEN DUP_VAL_ON_INDEX THEN
   DBMS_OUTPUT.PUT_LINE(sysdate||' Order '||onum||' already has an
   entry for Stock item '||scode);
   ROLLBACK WORK;
  WHEN OTHERS THEN
      ROLLBACK WORK;
     DBMS_OUTPUT.PUT_LINE(sysdate||' Failed when adding
   '||scode||' to order '||onum);
  END;
```

Points to ponder

- What happened to the errors that are handled in version 3, when we run version 1?
- Why is there no ‘rollback’ on the ‘no_data_found’ exception?
- Why is there a rollback on the other exceptions?
- Given that there is a rollback, what other vital SQL command is missing from our example?

Two more functions…

- These functions should only be used within the Exception Handling section of your code.
- SQLERRM
  - returns the error ***message*** associated with the most recently raised error exception.
- SQLCODE
  - returns the error ***number*** associated with the most recently raised error exception.

Use in context.

- **You could use them to output an error to the server as follows:**

```
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('The SQL Error with code '||SQLCODE||'
   has occurred.');
   DBMS_OUTPUT.PUT_LINE('This means '||SQLERRM);
 END;
```

- **Or you could log the error to a table as follows**:

```
EXCEPTION
      WHEN OTHERS THEN
         err_code := SQLCODE;
         err_msg := substr(SQLERRM, 1, 200);
       INSERT INTO audit_table (error_number, error_message)
        VALUES (err_code, err_msg);
   END;
```

- **Because they are inbuilt functions, you don’t need to declare them.**

Example 4 (1 of 3) sqlerrm

```
DECLARE
 onum   builder2.corder.corderno%TYPE:= :onum;
 scode  builder2.stock.stock_code%TYPE:= :scode;
 qtyreq      builder2.corderline.quantityrequired%TYPE:=
   :qtyreq;
 cur_q  builder2.stock.stock_level%TYPE;
 lev    builder2.stock.reorder_level%TYPE;
 not_enough_stock EXCEPTION;
begin
    SELECT stock_level, reorder_level
    INTO cur_q, lev
    FROM builder2.stock
    WHERE stock_code = scode;
```

2 of 3 sqlerrm

```
IF (cur_q > qtyreq) THEN
    UPDATE builder2.stock
      SET stock_level = stock_level - qtyreq
    WHERE stock_code = scode;
    IF (cur_q - qtyreq) < lev THEN
   DBMS_OUTPUT.PUT_LINE('Fallen below reorder quantity -
  reorder!');
    END IF;
    INSERT INTO builder2.corderline VALUES (qtyreq,onum,
  scode);
 ELSE
      RAISE not_enough_stock;
 END IF;
```

3 of 3 sqlerrm

```
EXCEPTION
  WHEN not_enough_stock THEN
    DBMS_OUTPUT.PUT_LINE(sysdate||' Order '||onum||' Stock item '||scode||
    'Do not have enough to sell');
    ROLLBACK WORK;
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE(sysdate||' Either the order '||onum||' or the Stock item
    '||scode||
    ' does not exist');
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE(sysdate||' Order '||onum||' already has an entry for
    Stock item '||scode);
    ROLLBACK WORK;
  WHEN OTHERS THEN
    ROLLBACK WORK;
    DBMS_OUTPUT.PUT_LINE(sysdate||' Failed adding '||scode||' to order '||onum);
    DBMS_OUTPUT.PUT_LINE('The SQL Error with code '||SQLCODE||' has occurred.');
    DBMS_OUTPUT.PUT_LINE('This means '||SQLERRM);
END;
```

Using functions

- Functions are very easy to call in PL/SQL and can even be called within an SQL statement.
- If we look at the previous example, it would be preferrable to find out whether or not the order or the stock code existed, before writing an orderline.
- We could also check to see if a previous orderline existed for the order and the stock code.

Let’s start with the stock code

- We want to write a function to find out if a stock code exists:
  - Take in the stock code (a parameter)
  - Return a boolean (True if it exists)
- To do this:
  - We try to read the stock row corresponding to the stock code value.
  - If it succeeds, we return TRUE
  - If it fails, we return FALSE.
    - There is no need to bother with the contents of the stock row, but we must provide somewhere into which the contents can be read.

Function Code

```
FUNCTION stock_exists(
scod builder2.stock.stock_code%TYPE) RETURN BOOLEAN
IS
  nam stock.stock_description%TYPE;
BEGIN
   SELECT stock_description
    INTO nam
    FROM stock
   WHERE stock_code = scod;
   DBMS_OUTPUT.PUT_LINE ('This stock number, '||scod||
   ' is already in use');
   RETURN TRUE;
EXCEPTION
WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE ('This stock number, '||scod||
  ' is not in use yet');
  RETURN FALSE;
END stock_exists;
```

We can…

- Put it into the DECLARE section
- OR
- Save it in the DBMS.
- To call it, in the executable section, we can say
  - IF stock_exists(scod) THEN…
- This will work if…
  - It is embedded in the code
  - It is saved in the schema in which we are working.

To save it in the DBMS

- Replace the first line…
  - `FUNCTION stock_exists(scod builder2.stock.stock_code%TYPE) RETURN BOOLEAN`
- With…
  - `CREATE OR REPLACE FUNCTION stock_exists(scod builder2.stock.stock_code%TYPE) RETURN BOOLEAN`