

SIT103/SIT772 Database Fundamentals

Week 8

Advanced SQL

(PL/SQL & Embedded SQL)

Dr Iynkaran Natgunanathan

OnTrack Tasks - IMPORTANT



- Many students are **yet to submit early week tasks**
- At this stage, those students are looking like **failing the unit**
- We are very concerned that you **may not have enough time to complete all tasks before the end of the trimester**
- Please seek help from tutors (workshop tutor, OnTrack marking tutor, and/or Helphub tutor) if you need help or their feedback on your tasks.

- Data Definition Language (DDL)
 - CREATE TABLE
 - ALTER TABLE
 - DROP TABLE
- Data Manipulation Language (DML)
 - INSERT
 - UPDATE
 - DELETE
- COMMIT and ROLLBACK
- Views

Any Questions?

Last Week's OnTrack Task



- 7.1P DML and DDL Commands
 - SQL with Other DML commands and DDL commands
- 7.2C Miniproject 2 - Part 2: Database Implementation
 - Implementing Normalised Database Designed in Task 4.2C

Advanced SQL

- Procedural Language (PL) SQL (PL/SQL)
 - Anonymous PL/SQL blocks
 - Triggers
 - Stored Procedures
 - PL/SQL Functions
- Embedded SQL

For PL/SQL, we will use ORACLE

- PL/SQL is ORACLE's proprietary procedural extension of SQL.
- Other RDMS (e.g., MySQL & MS SQL) have their own procedural extensions but they are not as comprehensive as ORACLE's PL/SQL
- Opportunity to work with ORACLE

Procedural Language SQL (PL/SQL)



- SQL does not support programming constructs like:
- Conditional execution of procedures

IF <condition>

THEN <perform procedure>

ELSE <perform alternate procedure>

END IF

- Looping operations for execution of repetitive actions

DO WHILE

<perform procedure>

END DO

- Procedural Language SQL (PL/SQL) is a language that makes it possible
 - to use and store procedural code and SQL statements within the database; and
 - to merge SQL and traditional programming constructs, such as variables, conditional processing (IF-THEN-ELSE), basic loops (FOR and WHILE loops), and error trapping.
- PL/SQL performs a conditional or looping operation by isolating critical code and making all application programs call the shared code
 - Better maintenance and logic control
- **Persistent Stored Module (PSM)** is a block of code that:
 - contains standard SQL statements and procedural extensions that is stored and executed at the DBMS server
 - represents business logic that can be encapsulated , stored, and shared among multiple database users

- End users can use PL/SQL to create
 - Anonymous PL/SQL blocks
 - Stored procedures
 - PL/SQL functions
 - Triggers
- Don't confuse PL/SQL functions with SQL's built-in functions such as MIN, MAX, etc.
 - Built-in functions can be used only within SQL statements
 - PL/SQL functions are mainly invoked within PL/SQL programs (e.g., triggers and procedures)
 - PL/SQL functions can be called within SQL statements provided that they conform to very specific rules that are dependent on DBMS environment

PL/SQL: Anonymous PL/SQL block



- Block of code that allows to use procedural constructs but it is not given a specific name
- Executes the block as soon as hit “Enter”
- Useful for testing purposes and to display specific message when an SQL is executed.
- Syntax:

[DECLARE

--- declare variables]

BEGIN

--- SQL and PL/SQL code

END;

/

A screenshot of a SQL Plus window titled 'SQL Plus'. The window shows the execution of two PL/SQL blocks. The first block inserts a record into the VENDOR table. The second block inserts another record and prints a message. Finally, a query is executed to select all records from the VENDOR table, displaying a table with 13 rows.

```
SQL> BEGIN
2  INSERT INTO VENDOR
3  VALUES (25678, 'Microsoft Corp.', 'Bill Gates', '765', '546-8484', 'WA', 'N');
4  END;
5  /

PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL>
SQL> BEGIN
2  INSERT INTO VENDOR
3  VALUES (25772, 'Clue Store', 'Issac Hayes', '456', '323-2009', 'VA', 'N');
4  DBMS_OUTPUT.PUT_LINE('New Vendor Added!');
5  END;
6  /
New Vendor Added!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM VENDOR;

  V_CODE V_NAME                V_CONTACT      V_A V_PHONE V_ V
-----
21225 Bryson, Inc.             Smithson        615 223-3234 TN Y
21226 SuperLoo, Inc.          Flushing       904 215-8995 FL N
21231 D&E Supply              Singh          615 228-3245 TN Y
21344 Gomez Bros.            Ortega         615 889-2546 KY N
22567 Dome Supply            Smith          901 678-1419 GA N
23119 Randsets Ltd.          Anderson       901 678-3998 GA Y
24004 Brackman Bros.         Browning       615 228-1410 TN N
24288 ORDVA, Inc.            Hakford        615 898-1234 TN Y
25443 B&K, Inc.               Smith          904 227-0093 FL N
25501 Damal Supplies         Smythe         615 890-3529 TN N
25595 Rubicon Systems        Orton          904 456-0092 FL Y
25678 Microsoft Corp.       Bill Gates     765 546-8484 WA N
25772 Clue Store              Issac Hayes    456 323-2009 VA N

13 rows selected.

SQL> _
```

Anonymous PL/SQL block (2)



SET SERVEROUTPUT ON

- This command enables the client console (SQL*Plus) to receive messages from the server side (Oracle DBMS).
- To stop receiving messages from the server, you would enter **SET SERVEROUTPUT OFF**.

DBMS_OUTPUT.PUT_LINE ()

- To send messages from the PL/SQL block to the console

```
SQL Plus

SQL> BEGIN
2  INSERT INTO VENDOR
3  VALUES (25678, 'Microsoft Corp.', 'Bill Gates', '765', '546-8484', 'WA', 'N');
4  END;
5  /

PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL>
SQL> BEGIN
2  INSERT INTO VENDOR
3  VALUES (25772, 'Clue Store', 'Issac Hayes', '456', '323-2009', 'VA', 'N');
4  DBMS_OUTPUT.PUT_LINE('New Vendor Added!');
5  END;
6  /

New Vendor Added!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM VENDOR;

  V_CODE V_NAME                                V_CONTACT      V_A V_PHONE  V_ V
-----
21225 Bryson, Inc.                            Smithson        615 223-3234 TN Y
21226 SuperLoo, Inc.                          Flushing       904 215-8995 FL N
21231 D&E Supply                              Singh          615 228-3245 TN Y
21344 Gomez Bros.                            Ortega         615 889-2546 KY N
22567 Dome Supply                            Smith          901 678-1419 GA N
23119 Randsets Ltd.                          Anderson       901 678-3998 GA Y
24004 Brackman Bros.                         Browning       615 228-1410 TN N
24288 ORDVA, Inc.                            Hakford        615 898-1234 TN Y
25443 B&K, Inc.                              Smith          904 227-0093 FL N
25501 Damal Supplies                         Smythe        615 890-3529 TN N
25595 Rubicon Systems                       Orton         904 456-0092 FL Y
25678 Microsoft Corp.                       Bill Gates     765 546-8484 WA N
25772 Clue Store                             Issac Hayes    456 323-2009 VA N

13 rows selected.

SQL>
```

Anonymous PL/SQL block - Example



```
DECLARE
```

```
W_P1 NUMBER(3) := 0;
```

```
W_P2 NUMBER(3) := 10;
```

```
W_NUM NUMBER(2) := 0;
```

```
BEGIN
```

```
WHILE W_P2 < 300 LOOP
```

```
    SELECT COUNT(P_CODE) INTO W_NUM FROM PRODUCT WHERE P_PRICE BETWEEN W_P1 AND W_P2;
```

```
    DBMS_OUTPUT.PUT_LINE('There are ' || W_NUM || ' Products with price between ' ||  
W_P1 || ' and ' || W_P2);
```

```
    W_P1 := W_P2 + 1;
```

```
    W_P2 := W_P2 + 50;
```

```
END LOOP;
```

```
END;
```

```
/
```

Variables to use in the PL/SQL Block

Loop terminating condition

SQL statement

Assigns SQL result into a local variable

Console output

Loop variables update

- The SELECT statement uses the INTO keyword to assign the output of the query to a PL/SQL variable.
- You can use the INTO keyword only inside a PL/SQL block of code.
- If the SELECT statement returns more than one value, you will get an error. (we will discuss later, Cursor)
- Note the use of the string concatenation symbol (||) to display the output.
- Each statement inside the PL/SQL code must end with a semicolon (;).

Anonymous PL/SQL block – Example (2)

```
SQL Plus

SQL> DECLARE
  2  W_P1  NUMBER(3) := 0;
  3  W_P2  NUMBER(3) := 10;
  4  W_NUM NUMBER(2) := 0;
  5  BEGIN
  6  WHILE W_P2 < 300 LOOP
  7      SELECT COUNT(P_CODE) INTO W_NUM FROM PRODUCT
  8      WHERE P_PRICE BETWEEN W_P1 AND W_P2;
  9      DBMS_OUTPUT.PUT_LINE('There are ' || W_NUM || ' Products with price between ' || W_P1 || ' and ' || W_P2);
 10      W_P1 := W_P2 + 1;
 11      W_P2 := W_P2 + 50;
 12  END LOOP;
 13  END;
 14  /

There are 5 Products with price between 0 and 10
There are 6 Products with price between 11 and 60
There are 3 Products with price between 61 and 110
There are 1 Products with price between 111 and 160
There are 0 Products with price between 161 and 210
There are 1 Products with price between 211 and 260

PL/SQL procedure successfully completed.

SQL> ■
```

TABLE 8.4

PL/SQL BASIC DATA TYPES

DATA TYPE	DESCRIPTION
CHAR	Character values of a fixed length; for example: W_ZIP CHAR(5)
VARCHAR2	Variable-length character values; for example: W_FNAME VARCHAR2(15)
NUMBER	Numeric values; for example: W_PRICE NUMBER(6,2)
DATE	Date values; for example: W_EMP_DOB DATE
%TYPE	Inherits the data type from a variable that you declared previously or from an attribute of a database table; for example: W_PRICE PRODUCT.P_PRICE%TYPE Assigns W_PRICE the same data type as the P_PRICE column in the PRODUCT table

PL/SQL: Stored Procedure



- A named collection of procedural and SQL statements - stored in the database
- Stored procedure can be used to encapsulate SQL statements for a business activity that can be executed as a single transaction
 - e.g., you can create a stored procedure to represent a product sale, a credit update, or the addition of a new customer.

Advantages:

1. Substantially reduce network traffic and increase performance
 - Because the procedure is stored at the server, there is no transmission of individual SQL statements over the network, executed locally on the RDBMS
2. Reduce code duplication by means of code isolation and code sharing
 - creates unique PL/SQL modules that can be called by application programs
 - minimizes the chance of errors and the cost of application development and maintenance.

PL/SQL: Stored Procedure (2)



Syntax:

```
CREATE OR REPLACE PROCEDURE procedure_name [(argument [IN/OUT/IN OUT]
data-type, ... )]
    [IS/AS]
    [variable_name data type[:=initial_value] ]
BEGIN
    PL/SQL or SQL statements;
    ...
END;
```

Execute the procedure:

```
EXEC procedure_name[(parameter_list);
```

- Argument specifies the parameters that are passed to the stored procedure.
- A stored procedure could have zero or more arguments or parameters.
- IN/OUT/ IN OUT indicates whether the parameter is for input, output, or both.
- Data-type is one of the procedural SQL data types used in the RDBMS.
- The data types normally match those used in the RDBMS table creation statement.
- Variables can be declared between the keywords IS and BEGIN.
- You must specify the variable name, its data type, and (optionally) an initial value.

Stored Procedure Example



- Create a procedure `PRC_PROD_DISCOUNT` to assign an additional 5 percent discount for all products with the `P_QOH` is more than or equal to twice the `P_MIN`.
 - uses `DBMS_OUTPUT.PUT_LINE()` to display a message when the procedure executes
 - assumes that you previously ran `SET SERVEROUTPUT ON`

A screenshot of a SQL Plus window titled "SQL Plus". The window contains the following SQL code:

```
SQL> CREATE OR REPLACE PROCEDURE PRC_PROD_DISCOUNT
2  AS BEGIN
3      UPDATE PRODUCT
4      SET P_DISCOUNT = P_DISCOUNT + .05
5      WHERE P_QOH >= P_MIN * 2;
6
7      DBMS_OUTPUT.PUT_LINE('** Update finished **');
8  END;
9  /
```

Below the code, the text "Procedure created." is displayed. The prompt "SQL> " is shown at the bottom of the window with a cursor.

Stored Procedure Example (2)

Execution of procedure
PRC_PROD_DISCOUNT

Results before and after
execution

The screenshot shows a SQL Plus window with the following content:

```
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;
```

P_CODE	P_DESCRIPT	P_QOH	P_MIN	P_DISCOUNT
11QER/31	Power painter, 15 psi., 3-nozzle	29	5	0.00
13-Q2/P2	7.25-in. pwr. saw blade	32	15	0.05
14-Q1/L3	9.00-in. pwr. saw blade	18	12	0.00
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15	8	0.00
1558-QW1	Hrd. cloth, 1/2-in., 3x50	23	5	0.00
2232/QTY	B&D jigsaw, 12-in. blade	8	5	0.05
2232/QWE	B&D jigsaw, 8-in. blade	6	7	0.05
2238/QPD	B&D cordless drill, 1/2-in.	12	5	0.05
23109-HB	Claw hammer	23	10	0.10
23114-AA	Sledge hammer, 12 lb.	8	10	0.05
54778-2T	Rat-tail file, 1/8-in. fine	43	20	0.00
89-WRE-Q	Hicut chain saw, 16 in.	11	5	0.05
PVC23DRT	PVC pipe, 3.5-in., 8-ft	188	75	0.00
SM-18277	1.25-in. metal screw, 25	172	75	0.00
SW-23116	2.5-in. wd. screw, 50	237	100	0.00
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	18	5	0.10

16 rows selected.

```
SQL> EXEC PRC_PROD_DISCOUNT;
** Update finished **

PL/SQL procedure successfully completed.
```

```
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;
```

P_CODE	P_DESCRIPT	P_QOH	P_MIN	P_DISCOUNT
11QER/31	Power painter, 15 psi., 3-nozzle	29	5	0.05
13-Q2/P2	7.25-in. pwr. saw blade	32	15	0.10
14-Q1/L3	9.00-in. pwr. saw blade	18	12	0.00
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15	8	0.00
1558-QW1	Hrd. cloth, 1/2-in., 3x50	23	5	0.05
2232/QTY	B&D jigsaw, 12-in. blade	8	5	0.05
2232/QWE	B&D jigsaw, 8-in. blade	6	7	0.05
2238/QPD	B&D cordless drill, 1/2-in.	12	5	0.10
23109-HB	Claw hammer	23	10	0.15
23114-AA	Sledge hammer, 12 lb.	8	10	0.05
54778-2T	Rat-tail file, 1/8-in. fine	43	20	0.05
89-WRE-Q	Hicut chain saw, 16 in.	11	5	0.10
PVC23DRT	PVC pipe, 3.5-in., 8-ft	188	75	0.05
SM-18277	1.25-in. metal screw, 25	172	75	0.05
SW-23116	2.5-in. wd. screw, 50	237	100	0.05
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	18	5	0.15

16 rows selected.

```
SQL> _
```

Stored Procedure (3)



- The previous PRC_PROD_DISCOUNT procedure worked well, but it is **not flexible**
 - what if you want to increase the discount by a different percent?
- We can pass **an argument to define the rate of increase** to the procedure.
- Multiple arguments must be enclosed in parentheses and separated by commas.
- IN/OUT/ IN OUT indicates whether the parameter is for input, output, or both.

```
SQL> CREATE OR REPLACE PROCEDURE PRC_PROD_DISCOUNT (WPI IN NUMBER) AS
2 BEGIN
3   IF ((WPI <= 0) OR (WPI >= 1)) THEN --validate WPI parameter
4     DBMS_OUTPUT.PUT_LINE('Error: Value must be greater than 0 and less than 1');
5   ELSE -- if value greater than 0 and less than 1
6     UPDATE PRODUCT
7     SET P_DISCOUNT = P_DISCOUNT + WPI
8     WHERE P_QOH >= P_MIN * 2;
9     DBMS_OUTPUT.PUT_LINE('** Update finished **');
10  END IF;
11 END;
12 /

Procedure created.

SQL> _
```

IN argument WPI of type NUMBER

Validate if argument is acceptable

Apply the change if it is acceptable

Stored Procedure



Execute the procedure with value for the input argument:

- WPI=1.5
- WPI=0.05

A screenshot of a SQL Plus window titled "SQL Plus". The window contains the following text:

```
SQL> EXEC PRC_PROD_DISCOUNT(1.5);  
Error: Value must be greater than 0 and less than 1  
  
PL/SQL procedure successfully completed.  
  
SQL> EXEC PRC_PROD_DISCOUNT(.05);  
** Update finished **  
  
PL/SQL procedure successfully completed.  
  
SQL> _
```

PL/SQL: Stored Procedure - Cursor



- All of the SQL statements we have used so far inside a PL/SQL block (stored procedure) have returned a single value.
 - If the SQL statement returns more than one value, it will generate an error.
- To handle this we need to use a **Cursor**.
- A **cursor** is a special construct used in PL/SQL to hold the data rows returned by an SQL query.
 - You can think of a cursor as a reserved area of memory in which the output of the query is stored, like an array holding columns and rows.
- Cursors are held in a reserved memory area in the DBMS server, not in the client computer.

- There are two types of cursors: 1) Implicit; and 2) Explicit
- **Implicit:** An implicit cursor is automatically created in PL/SQL when the SQL statement returns only one value.
- **Explicit:** An explicit cursor is created to hold the output of a SQL statement that may return two or more rows (but could return zero rows or only one).
- To create an explicit cursor, you use the following syntax inside a PL/SQL DECLARE section:

CURSOR *cursor_name* IS *select-query*;

- Once you have declared a cursor, you can use specific PL/SQL cursor processing commands
 - OPEN, FETCH, and CLOSE anywhere between the BEGIN and END keywords of the PL/SQL block

TABLE 8.5

CURSOR PROCESSING COMMANDS

CURSOR COMMAND	EXPLANATION
OPEN	<p>Opening the cursor executes the SQL command and populates the cursor with data, opening the cursor for processing. The cursor declaration command only reserves a named memory area for the cursor; it does not populate the cursor with the data. Before you can use a cursor, you need to open it. For example:</p> <pre>OPEN <i>cursor_name</i></pre>
FETCH	<p>Once the cursor is opened, you can use the FETCH command to retrieve data from the cursor and copy it to the PL/SQL variables for processing. The syntax is:</p> <pre>FETCH <i>cursor_name</i> INTO variable1 [, variable2, ...]</pre> <p>The PL/SQL variables used to hold the data must be declared in the DECLARE section and must have data types compatible with the columns retrieved by the SQL command. If the cursor's SQL statement returns five columns, there must be five PL/SQL variables to receive the data from the cursor.</p> <p>This type of processing resembles the one-record-at-a-time processing used in previous database models. The first time you fetch a row from the cursor, the first row of data from the cursor is copied to the PL/SQL variables; the second time you fetch a row from the cursor, the second row of data is placed in the PL/SQL variables; and so on.</p>
CLOSE	<p>The CLOSE command closes the cursor for processing.</p>

- How do you know what number of rows are there in the cursor?
- Or how do you know when you have reached the end of the cursor?
- Cursors have special attributes to convey such important information.

TABLE 8.6

CURSOR ATTRIBUTES

ATTRIBUTE	DESCRIPTION
%ROWCOUNT	Returns the number of rows fetched so far. If the cursor is not OPEN, it returns an error. If no FETCH has been done but the cursor is OPEN, it returns 0.
%FOUND	Returns TRUE if the last FETCH returned a row, and FALSE if not. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL.
%NOTFOUND	Returns TRUE if the last FETCH did not return any row, and FALSE if it did. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL.
%ISOPEN	Returns TRUE if the cursor is open (ready for processing) or FALSE if the cursor is closed. Remember, before you can use a cursor, you must open it.

Cursor (5)

Cursor declaration

Products QoH greater than the average QoH for all products.

Start of PL/SQL block

LOOP to access each record

Close the cursor

```
SQL Plus

SQL> CREATE OR REPLACE PROCEDURE PRC_CURSOR_EXAMPLE IS
2  W_P_CODE    PRODUCT.P_CODE%TYPE;
3  W_P_DESCRIPT PRODUCT.P_DESCRIPT%TYPE;
4  W_TOT       NUMBER(3);
5  CURSOR PROD_CURSOR IS
6      SELECT P_CODE, P_DESCRIPT
7      FROM PRODUCT
8      WHERE P_QOH > (SELECT AVG(P_QOH) FROM PRODUCT);
9  BEGIN
10     DBMS_OUTPUT.PUT_LINE('PRODUCTS WITH P_QOH > AVG(P_QOH)');
11     DBMS_OUTPUT.PUT_LINE('=====');
12     OPEN PROD_CURSOR;
13     LOOP
14         FETCH PROD_CURSOR INTO W_P_CODE, W_P_DESCRIPT;
15         EXIT WHEN PROD_CURSOR%NOTFOUND;
16         DBMS_OUTPUT.PUT_LINE(W_P_CODE || ' -> ' || W_P_DESCRIPT );
17     END LOOP;
18     DBMS_OUTPUT.PUT_LINE('=====');
19     DBMS_OUTPUT.PUT_LINE('TOTAL PRODUCT PROCESSED ' || PROD_CURSOR%ROWCOUNT);
20     DBMS_OUTPUT.PUT_LINE('--- END OF REPORT ---');
21     CLOSE PROD_CURSOR;
22 END;
23 /

Procedure created.

SQL> EXEC PRC_CURSOR_EXAMPLE;
PRODUCTS WITH P_QOH > AVG(P_QOH)
=====
PVC23DRT -> PVC pipe, 3.5-in., 8-ft
SM-18277 -> 1.25-in. metal screw, 25
SW-23116 -> 2.5-in. wd. screw, 50
=====
TOTAL PRODUCT PROCESSED 3
--- END OF REPORT ---

PL/SQL procedure successfully completed.

SQL> _
```

Inherit datatype

Concatenation

Exit when there is no record in the cursor

Fetch the first record in the cursor and place values in a local PL/SQL variables

PL/SQL Stored Functions



- Named group of procedural and SQL statements that returns a value
 - Indicated by a RETURN statement in its program
- Can be invoked from PL/SQL blocks, can only be invoked from SQL statements if the function follows some very specific compliance rules

Syntax:

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

Stored Function - Example



```
CREATE OR REPLACE FUNCTION get_total_sales(in_year INTEGER)
RETURN NUMBER IS
    total_sales NUMBER := 0;
BEGIN
    SELECT SUM(unit_price * quantity) INTO total_sales
    FROM order_items
    INNER JOIN orders USING(order_id)
    WHERE status = 'Shipped'
    GROUP BY EXTRACT(YEAR FROM order_date)
    HAVING EXTRACT(YEAR FROM order_date) = in_year;
RETURN total_sales;
END;
/
```

Argument

Return datatype

Assign SQL result into a local variable

Tables

Attributes

SQL statement to get total sales of the given year

Return the total sale

Stored Function – Example (2)



Calling a PL/SQL function

- In a PL/SQL block

```
DECLARE
    l_sales_2017 NUMBER := 0;
BEGIN
    l_sales_2017 := get_total_sales(2017);
    DBMS_OUTPUT.PUT_LINE('Sales 2017: ' || l_sales_2017);
END;
```

- In a SQL statement

```
SELECT get_total_sales(2017) FROM dual;
```

- A trigger is a procedural SQL code that is automatically invoked by RDBMS when given **data manipulation event occurs**
 - It is associated with a database table; each table may have one or more triggers.
 - It is executed as part of the transaction that triggered it. Triggers are critical to proper database operation and management.
 - It is invoked **before or after a data row is inserted, updated, or deleted.**
- Triggers can be used to update table values, insert records in tables, and call other stored procedures.
- Triggers are used for (some examples):
 - Enforcement of business or security constraints
 - Creating audit logs
 - Automatic generation of derived column values
 - Creating replica tables for backup

A trigger example



- In Sales/Inventory management system, each time a product is sold
 - The system must update the product's P_QOH
 - The system should also automatically check and send a reorder request to the vendor if P_QOH falls below its reorder threshold P_MIN.
- It can be done by executing multiple SQLs
 - the SQLs should be executed in correct order
 - someone must remember to run them in the right order
 - this multistage process is not efficient and effective
- It can be done automatically using triggers.

Trigger



Syntax to create a trigger:

```
CREATE OR REPLACE TRIGGER trigger_name
[BEFORE/AFTER] [DELETE/INSERT/UPDATE OF column_name] ON table_name
[FOR EACH ROW]
[DECLARE]
[variable_name data_type[:=initial_value] ]
BEGIN
PL/SQL instructions;

...
END;
/
```

Delete Trigger:

```
DROP TRIGGER trigger_name
```

Trigger (2)

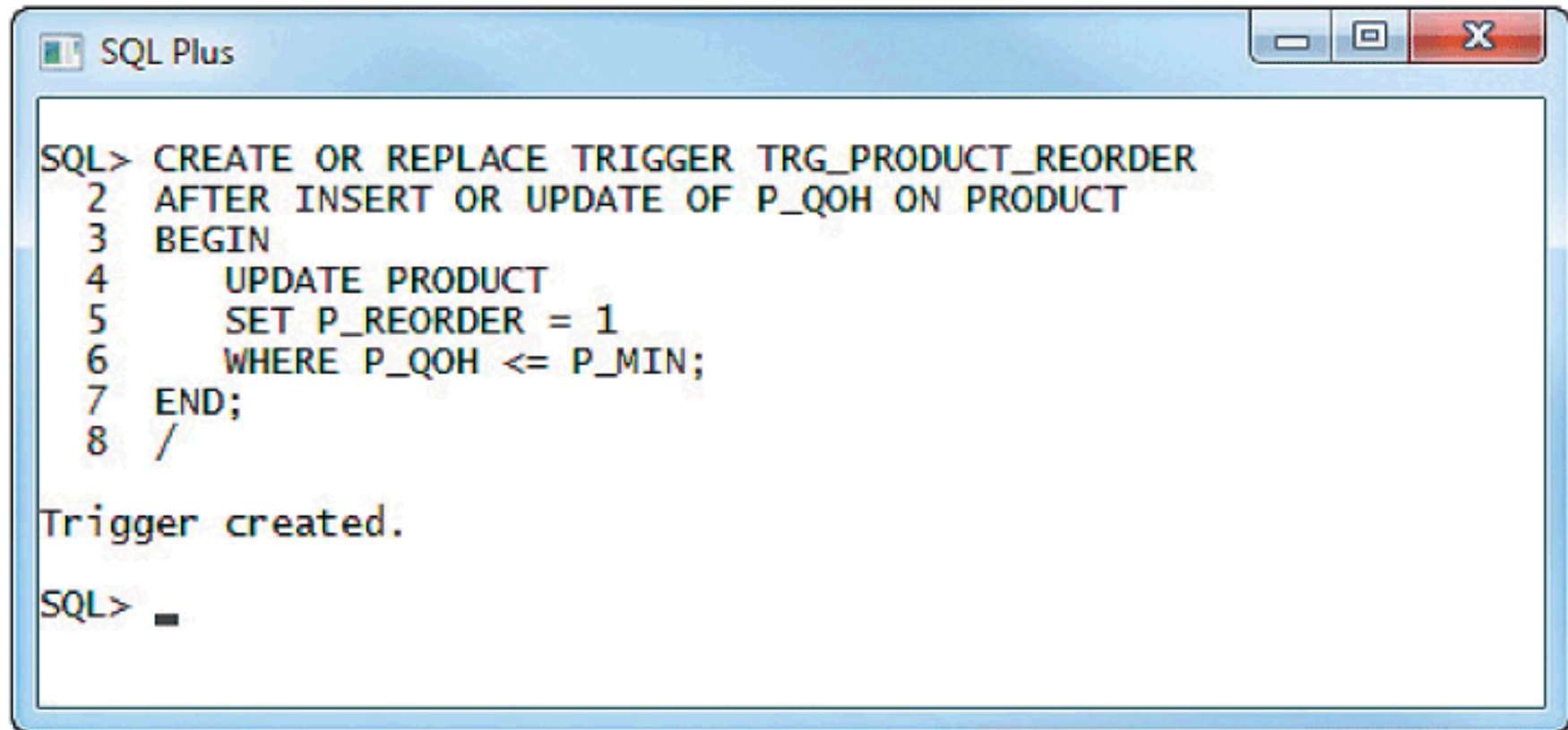


- **The triggering timing:** BEFORE or AFTER.
 - This timing indicates when the trigger's PL/SQL code executes—in this case, before or after the triggering statement is completed.
- **The triggering event:**
 - The statement that causes the trigger to execute (INSERT, UPDATE, or DELETE).
- **The triggering level:** two types
 - **A statement-level trigger:** default case if you omit the FOR EACH ROW keywords.
 - This type of trigger is executed once, before or after the triggering statement is completed.
 - **A row-level trigger:** This requires use of the FOR EACH ROW keywords.
 - This type of trigger is executed once for each row affected by the triggering statement.
- **The triggering action:**
 - The PL/SQL code enclosed between the BEGIN and END keywords.
 - Each statement inside the PL/SQL code must end with a semicolon (;).

Trigger Example



- If the value of P_QOH is equal to or less than P_MIN (AFTER INSERT or UPDATE), the trigger UPDATES the P_REORDER to 1.

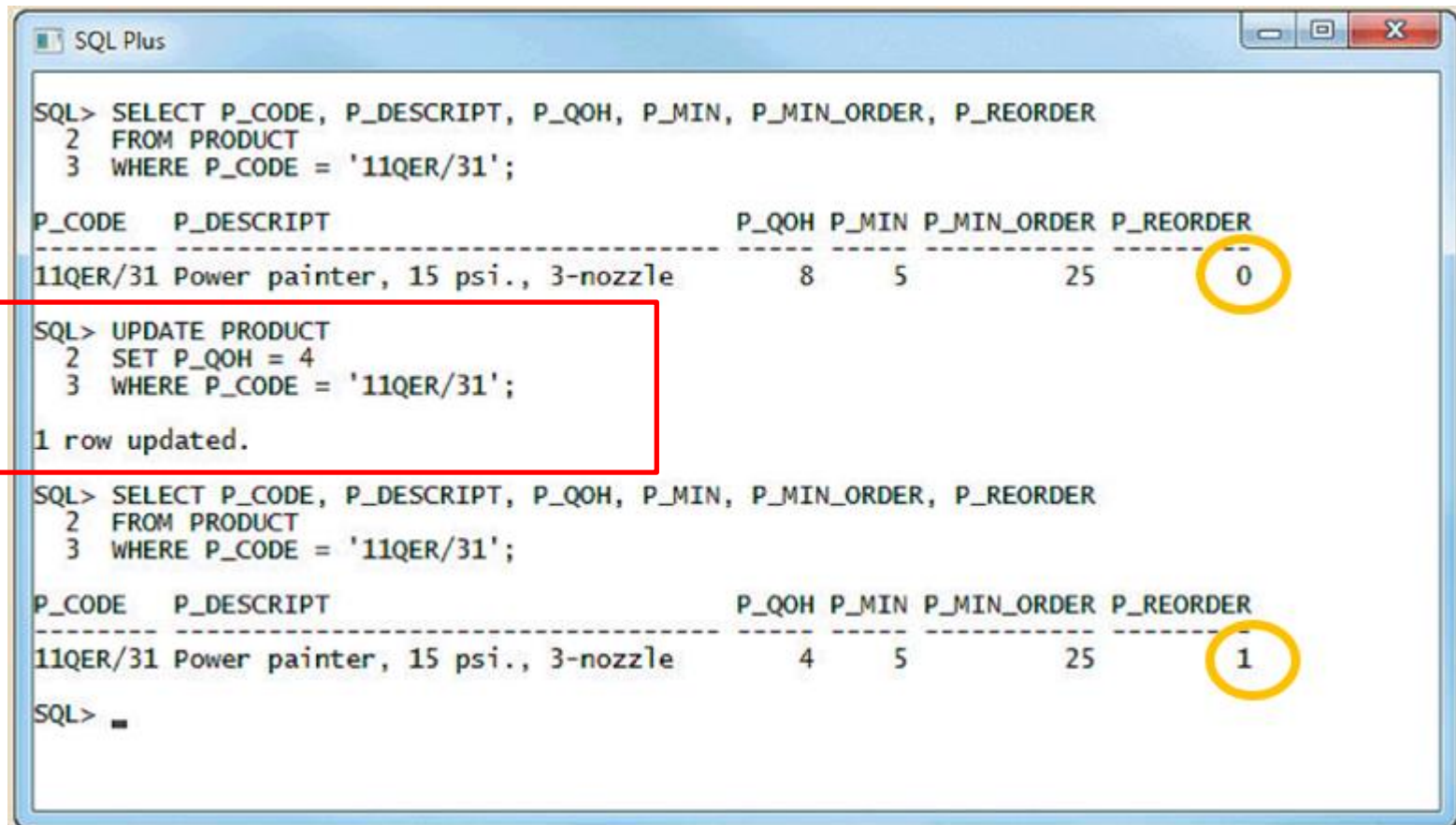
A screenshot of a SQL Plus window titled 'SQL Plus'. The window contains the following SQL code:

```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
2  AFTER INSERT OR UPDATE OF P_QOH ON PRODUCT
3  BEGIN
4      UPDATE PRODUCT
5      SET P_REORDER = 1
6      WHERE P_QOH <= P_MIN;
7  END;
8  /
```

Below the code, the message 'Trigger created.' is displayed. The prompt 'SQL>' is followed by a cursor line.

Trigger Example (2)

- Testing TRG_PRODUCT_REORDER trigger
- Updating the P_QOH of product 11QER/31 to 4 automatically fires the trigger and updates P_REORDER to 1.



```
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
2 FROM PRODUCT
3 WHERE P_CODE = '11QER/31';

P_CODE  P_DESCRIPT                                P_QOH P_MIN P_MIN_ORDER P_REORDER
-----
11QER/31 Power painter, 15 psi., 3-nozzle      8     5         25         0

SQL> UPDATE PRODUCT
2 SET P_QOH = 4
3 WHERE P_CODE = '11QER/31';

1 row updated.

SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
2 FROM PRODUCT
3 WHERE P_CODE = '11QER/31';

P_CODE  P_DESCRIPT                                P_QOH P_MIN P_MIN_ORDER P_REORDER
-----
11QER/31 Power painter, 15 psi., 3-nozzle      4     5         25         1

SQL> _
```

Update
P_QOH

- In real-world, databases are part of systems developed using application programming languages such as PHP, VB, .Net, Python, JAVA, etc.
 - SQL statements are contained within those languages
 - **Host language:** any language that contains embedded SQL statements
- Differences between SQL and procedural languages

Run-time mismatch

SQL is a **non-procedural, interpreted language**, i.e., each instruction is parsed, its syntax is checked, and it is executed one instruction at a time. All processing takes place at the server side.

Meanwhile, **the host language is generally a binary-executable program** (also known as a compiled program). It typically runs at the client side in its own memory space, which is different from the DBMS environment.

Data type mismatch

Data types provided by SQL might not match data types used in different host languages

Embedded SQL: A brief Intro (2)



- Hosting of SQL in another application program
 - Basis for applications to communicate with DBMS
 - JSP, ASP, PHP and more support embedded SQL
- Two main Types:
 - A. Hard-coded SQL:** programmer uses predefined SQL statements and parameters
 - SQL statements will not change while application is running
 - B. Dynamic SQL:** SQL statement is generated at run time
 - Attribute list and condition are not known until end user specifies them
 - Slower than static SQL
 - Requires more computer resources
 - Inconsistent levels of support and incompatibilities among DBMS vendors

Example: Embedded SQL: Hard-coded



- Retrieve data from Oracle database via PHP

```
<?php
```

```
$dbuser = "rad";
```

```
// database username
```

```
$dbpass = "password4rad";
```

```
// database password
```

```
$db = "SSID";
```

```
// database name
```

```
$connect = oci_connect($dbuser, $dbpass, $db);
```

```
// logon to the database
```

```
$query = "SELECT * FROM PRODUCT";
```

```
// build a SQL statement
```

```
$stmt = oci_parse($connect, $query);
```

```
// Prepares an Oracle statement for execution
```

```
oci_execute($stmt);
```

```
// execute the SQL statement
```

```
?>
```

Example: Embedded SQL: Dynamic



```
<?php
```

```
$errorMessage = $query = "";
```

```
$keyword = $_POST["keyword"];
```

Obtain a keyword from the web page.

```
if (preg_match("/^[a-zA-Z]+$/", $keyword))
```

```
    $query = "SELECT P_DESCRIPT " .
```

```
        "FROM PRODUCT " .
```

```
        "WHERE P_DESCRIPT LIKE '%" . $keyword . "%'";
```

Place that keyword into the SELECT statement at runtime.

```
else
```

```
    $errorMessage = "Only letters allowed";
```

```
?>
```

Example: Embedded SQL: Demo



- Web form to add a student

Student form

Below is an example form How to connect HTML to database with MySQL using PHP? An example

Your ID

Your First Name

Your Last Name

Your Email

[Add me to student](#)

Source: <https://www.raghwendra.com/blog/how-to-connect-html-to-database-with-mysql-using-php-example/>

Example: Embedded SQL: Demo (2)



PHP Code to
get form data
and insert a
record in the
STUDENT
table

```
<?php
// database connection code
if(isset($_POST['txtID']))
{
    // $con = mysqli_connect('localhost', 'database_user', 'database_password','database');
    $con = mysqli_connect('localhost', 'root', '', 'sunil_demo');

    // get the post records

    $txtID = $_POST['txtID'];
    $txtFName = $_POST['txtFName'];
    $txtLName = $_POST['txtLName'];
    $txtEmail = $_POST['txtEmail'];

    // database insert SQL code
    $sql = "INSERT INTO `student` (`ID`, `FNAME`, `LNAME`, `EMAIL`)
VALUES ($txtID, '$txtFName', '$txtLName', '$txtEmail');";

    // insert in database
    $rs = mysqli_query($con, $sql);
    if($rs)
    {
        echo "Student Record Inserted";
    }
    else {
        echo "Something wrong, record not inserted";
    }
}
else
{
    echo "txtID not set";
}
?>
```

Example: Embedded SQL: Demo (3)



Python code to read records from the STUDENT table

```
import mysql.connector as db

#establishing the connection
conn = db.connect(user='root', password='', host='127.0.0.1', database='sunil_demo')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Executing an MYSQL function using the execute() method
cursor.execute("SELECT * FROM STUDENT")

# Fetch each row.
for row in cursor.fetchall():
    print(str(row[0]) + ": " + row[1] + " " + row[2] + " (" + row[3] + ")")

#Closing the connection
conn.close()
```


- Procedural Language SQL (PL/SQL)
 - Anonymous PL/SQL blocks
 - Triggers
 - Stored Procedures
 - Cursors
 - PL/SQL Functions
- Embedded SQL: A brief introduction

This Week's OnTrack Task



- **No Pass Task**

- Students aiming for a 'P' are encouraged to play with ORACLE and practice PL/SQL in ORACLE

- 8.1C Online Quiz 2 (the same as 5.2C Online Quiz 1)

- Do the online Quiz 2 in the CloudDeakin site
 - Submit the screenshot of your Quiz Result (80% or more)
 - Two attempts, 1.5 hours (90 mins) to complete once started

- 8.2D PL/SQL Exercise

Next Week



- Business Intelligence and Big Data

Thank you

See you next week

Any questions/comments?

Readings and References:



- Chapter 8

Database Systems : Design, Implementation, & Management
13TH EDITION, by Carlos Coronel, Steven Morris