Procedural SQL

SCS2209-Database II
Mr. Rangana Jayashanka

Procedural SQL

- Thus far, you have learned to use SQL to read, write, and delete data in the database.
- Example: you learned to update values in a record, to add records, and to delete records.
- Unfortunately, SQL does not support the conditional execution of procedures that are typically supported by a programming language using the general format:

```
IF<condition>
        THEN <perform procedure>
             ELSE <perform alternate procedure>
END IF
```

Procedural SQL

- SQL also fails to support looping operations in programming languages that permit the execution of repetitive actions typically encountered in a programming environment. The typical format is:

```
DO WHILE
            <perform procedure>
END DO
```


Procedural SQL

- Traditionally, if you wanted to perform a conditional or looping type of operation (that is, a procedural type of programming using an IF-THEN-ELSE or DO-WHILE statement), you would use a programming language such as Visual Basic .NET, C#, or Java.
- This is why many older (so-called legacy) business applications are based on extensive number of COBOL program lines.
- An environment characterized by such redundancies often creates data management problems.

Procedural SQL

- A better approach is to isolate critical code and then have all application programs call the shared code.
- The advantage of this modular approach is that the application code is isolated in a single program, thus yielding better maintenance and logic control.
- To meet that requirement, most RDBMS vendors created numerous programming extensions. Those extensions include:
  - Flow-control procedural programming structures (IF-THEN-ELSE, DO-WHILE) for logic representation.
  - Variable declaration and designation within the procedures.
  - Error management.

Procedural SQL

- SQL-99 standared defined the use of persistent stored modules.
- A persistent stored module (PSM) is a block of code containing standard SQL statements and procedural extensions that is stored and executed at the DBMS server.

- THE PSM represents business logic that can be encapsulated, stored, and shared among multiple database users.

- Oracle implements PSMs through its procedural SQL language.

Procedural SQL

- 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), loops, and error trapping.

- The procedural code is executed as a unit by the DBMS when it is invoked (deirectly or indirectly) by the end user.

Procedural SQL

End users can use PL/SQL to create:

- Anonymous PL/SQL blocks.
- Triggers.
- Stored Procedures
- PL/SQL functions

Procedural SQL

- Using Oracle SQL*Plus, you can write PL/SQL code block by enclosing the commands inside BEGIN and END clauses.

```
BEGIN
     INSERT INTO VENDOR
     VALUES (25678, 'Microsoft Corp', 'Bill Gates', '765', 'WA')
END;
```

- THE PL/SQL block is an **anonymous PL/SQL** block because it has not been given a specific name.

SQL Stored Procedures

- A Stored Procedure is a named collection if procedural SQL statements.
- Just like database triggers, stored procedures are stored in the database.
- One of major advantages of stored procedures is that they can be used to encapsulate and represent business transactions.
- You can create a stored procedure to represent a product sale, a credit update, or the addition of a new customer.
- By doing that you can encapsulate SQL statements within a single stored procedure and execute them as a single transaction.

SQL Stored Procedures

- There are two clear advantages to the use from stored procedures:

1. Stored procedures 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. The use of stored procedures improves system performance because all transactions are executed locally on the RDBMS, so each SQL statement does not have to travel over the network.
2. Stored procedures help reduce code duplication by means of code isolation and code sharing, thereby minimizing the chance of errors and the cost of application development and maintenance.

SQL Stored Procedures

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

- So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

- You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

SQL Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```


Demo Database

Customer ID	Customer Name	Contact No	Age	City	Postal Code
001	Roshan	0710567898	34	Matara	800
002	Supun	0775678549	27	Kandy	500
003	Kasun	0748765431	40	Galle	300
004	Oshan	0716587542	32	Kurunegala	200
005	Geethika	0774534231	28	Kurunegala	200

Stored Procedure Example

- The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

- Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers;
```


Stored Procedure With One Parameter

- The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

- Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'Galle';
```

Stored Procedure With Multiple Parameters

- The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30),
@PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode =
@PostalCode
GO;
```

- Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'Kurunegala', @PostalCode =
'200';
```


Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Syntax

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.

Syntax

- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example 02

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

```
Select * from customers;
```


Example 2

Create a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger should display the salary difference between the old values and new values.

Example 2

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```


Example 2

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Example 2

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00);

Old salary:
New salary: 7500
Salary difference:

UPDATE customers
SET salary = salary + 500
WHERE id = 2;

Old salary: 1500
New salary: 2000
Salary difference: 500

Exam 2015

Consider the following schema to answer the given questions.

```
Employee (eno, ename, hireDate, Salary)
Project (projectID, projName, budget)
Emp_Proj (eno, ProjID, assignedDate)
```

Write a trigger to calculate and print the different between the old and new salary when the salary is updated.

Exam 2015

```
CREATE trigger R1
After update ON Employee
FOR EACH ROW
BEGIN
      declare sal.dif int;
       if (new.Salary >= old.Salary) then
       set sal.dif = new.Salary - old.Salary;
       else
       set sal.dif = old.Salary - new.Salary;
        end if;
       dbms_output.put_line(sal.dif);
END//
```