PL SQL Anonymous Block

These code blocks produce variables, views, and tables and work with triggers. But these are not saved in Oracle SQL Server. After a successful logout, your PL SQL blocks will not be there.

Structure

```
SET SERVEROUTPUT ON;

--Declaration
DECLARE

--Variable_NAME TYPE [NOT NULL] [:= | DEFAULT initial_value]
NAME VARCHAR2(20) := 'SHUVRO';
AMOUNT NUMBER(10,3) := 5000;
PORTION NUMBER(10,3) := AMOUNT / 5;

--Execution
BEGIN
DBMS_OUTPUT.PUT_LINE('WELCOME TO THE THUNDERDOME, ' || NAME);
DBMS_OUTPUT.PUT_LINE('YOUR TEAM HAS ' || AMOUNT || ' GOLD');
DBMS_OUTPUT.PUT_LINE('YOU HAVE ' || PORTION || ' GOLD');

END;

/
```

Output

```
WELCOME TO THE THUNDERDOME, SHUVRO
YOUR TEAM HAS 5000 GOLD
YOU HAVE 1000 GOLD
```

Let's break down this PL/SQL code into simple terms:

1. **Declaration Section**: This is where we declare our variables. Think of variables as containers where we can store information.
    - `NAME VARCHAR2(20):= 'SHUVRO';` Here, we're creating a variable called `NAME` that can hold text (up to 20 characters). We're putting the word 'SHUVRO' in it.
    - `AMOUNT NUMBER(10,3):= 5000;` We're creating a variable called `AMOUNT` that can hold numbers. We're putting the number 5000 in it.

- `PORTION NUMBER(10,3):= AMOUNT / 5;` We're creating a variable called `PORTION` that will hold the result of dividing the `AMOUNT` by 5.

2. **Execution Section**: This is where the action happens. We're telling the computer what to do with the variables we declared.
   - `DBMS_OUTPUT.PUT_LINE('WELCOME TO THE THUNDERDOME, ' || NAME);` Here, we're asking the computer to display a message that says 'WELCOME TO THE THUNDERDOME, ' followed by whatever is in the `NAME` variable (in this case, 'SHUVRO').
   - `DBMS_OUTPUT.PUT_LINE('YOUR TEAM has' || AMOUNT || ' GOLD');` Similarly, we're displaying a message that says 'YOUR TEAM has' followed by the `AMOUNT` (5000), and then ' GOLD'.
   - `DBMS_OUTPUT.PUT_LINE('YOU HAVE ' || PORTION || ' GOLD');` Finally, we're displaying 'YOU HAVE ' followed by the `PORTION` (which is `AMOUNT` divided by 5), and then ' GOLD'.
3. `END;` This tells the computer that we're done giving instructions.
4. `/` This is used to execute the block of code we've written.

So, in simple terms, this code is like a mini-script for a game, where a character named 'SHUVRO' enters the 'THUNDERDOME' with his team. They have 5000 gold, and 'SHUVRO' has a fifth of that amount. The code displays these details as messages.

Why do we need to state `SET SERVEROUTPUT ON;` before running pl/SQL code?

When you're writing PL/SQL code, you often want to display messages to understand what's happening, like leaving little breadcrumbs along the way. This is especially useful when you're debugging your code (fixing errors) or trying to understand how your code works.

In PL/SQL, we use `DBMS_OUTPUT.PUT_LINE` to display these messages. Think of `DBMS_OUTPUT.PUT_LINE` as a way of saying "Hey, computer, please show this message on the screen!"

But here's the thing: by default, the Oracle server doesn't display these messages. It's like the server has earmuffs on, and it can't hear us asking it to display the messages.

That's where `SET SERVEROUTPUT ON;` comes in. This command is like taking the earmuffs off the server. When we run `SET SERVEROUTPUT ON;`, we're telling the server "Hey, we're going to display some messages, so please pay attention and show them on the screen!"

So, before writing PL/SQL code that includes `DBMS_OUTPUT.PUT_LINE`, we run `SET SERVEROUTPUT ON;` to make sure our messages will be displayed. It's a bit like checking

that your microphone is on before giving a speech. You want to make sure your audience (in this case, the Oracle server) can hear you!

That's why we need to run `SET SERVEROUTPUT ON;` before writing PL/SQL. It ensures that our `DBMS_OUTPUT.PUT_LINE` messages will be heard and displayed.

To Edit A pl/SQL code

To edit a pl/SQL code after inserting it into the console, we just need to run the edit keyword to the console.

Steps,

- Write `edit` and hit enter on the console
- A .buf file will open on Notepad
- The .buf will already have the existing pl/SQL code in it.
- Edit the file as you want.
- Save and close
- The updates will run automatically

PL/SQL Anchored Declarations

For this topic, we will need a table,

```
Table Boys
```

ID	NAME	SEMESTER
1	AFLAN	7th
2	SAIDUL	8th
3	ANAS	8th

To create this table we are using normal SQL

```
CREATE TABLE Boys (
    id NUMBER PRIMARY KEY,
    name VARCHAR2(50),
    semester VARCHAR2(10)
);
```

```
INSERT INTO Boys (id, name, semester) VALUES (1, 'Aflan', '7th');
INSERT INTO Boys (id, name, semester) VALUES (2, 'Saidul', '8th');
INSERT INTO Boys (id, name, semester) VALUES (3, 'Anas', '8th')
```

Now call the value from the database and store it in a variable, we need to write,

```
DECLARE
--Variable_name table_name.column_name%TYPE
Var_Name Boys.NAME%TYPE;
Var_SEMESTER Boys.SEMESTER%TYPE;

BEGIN

-- SELECT statement
SELECT
    NAME, SEMESTER
INTO
    Var_Name, Var_SEMESTER
FROM
    Boys
WHERE
    Boys.ID = 2;

-- output
DBMS_OUTPUT.PUT_LINE('NAME: ' || Var_Name);
DBMS_OUTPUT.PUT_LINE('SEMESTER: ' || Var_SEMESTER);

END;
/
```

Output

```
NAME: Saidul
SEMESTER: 8th
```

This is a PL/SQL block of code. Here's what it does:

1. **Variable Declaration**: At the beginning of the block, two variables are declared, `Var_Name` and `Var_SEMESTER`. The `%TYPE` attribute is used to declare a variable of the same data type as a table's column. In this case, `Var_Name` and `Var_SEMESTER` are declared to be of the same type as `Boys.NAME` and `Boys.SEMESTER` respectively.

2. **SELECT INTO Statement**: This is a way to retrieve data from the database and store it directly into variables. The `SELECT NAME, SEMESTER INTO Var_Name, Var_SEMESTER FROM Boys WHERE Boys.ID = 2;` statement retrieves the `NAME` and `SEMESTER` of the boy whose `ID` is `2` from the `Boys` table and stores them into the `Var_Name` and `Var_SEMESTER` variables respectively.
3. **DBMS_OUTPUT.PUT_LINE**: This procedure in the `DBMS_OUTPUT` package is used to display output on the screen. The lines `DBMS_OUTPUT.PUT_LINE('NAME: ' || Var_Name);` and `DBMS_OUTPUT.PUT_LINE('SEMESTER: ' || Var_SEMESTER);` print the `NAME` and `SEMESTER` of the boy whose `ID` is `2`.
4. **END; /**: This marks the end of the PL/SQL block. The `/` on a line by itself tells SQL*Plus to execute the PL/SQL block.

So, in summary, this code retrieves the name and semester of the boy with ID 2 from the `Boys` table and prints them out.

PL/SQL if statements

We are going to show boys' grades based on their marks. So first we need to alter and update the table with marks.

```
ALTER TABLE Boys
ADD marks NUMBER;

UPDATE Boys
SET marks = CASE id
    WHEN 1 THEN 70
    WHEN 2 THEN 85
    WHEN 3 THEN 64
END;
```

Now the table looks like,

ID	NAME	SEMESTER	MARKS
1	AFLAN	7TH	70
2	SAIDUL	8TH	85
3	ANAS	8TH	64

Now we want to extract the name and grade of ID - 3. Logic will be

- $Mark > 80 = A$

- $80 > Mark > 70 = B$
- $70 > Mark = C$

So the PL/SQL block will look like,

```
DECLARE

Var_name Boys.NAME%TYPE;
Var_marks Boys.MARKS%TYPE;

BEGIN

SELECT NAME, MARKS
INTO Var_name, Var_marks
FROM Boys
WHERE Boys.ID = 3;

IF Var_marks > 80 THEN
    DBMS_OUTPUT.PUT_LINE('NAME: ' || Var_name);
    DBMS_OUTPUT.PUT_LINE('MARKS: ' || Var_marks);
    DBMS_OUTPUT.PUT_LINE('GRADE: A');

ELSIF Var_marks > 70 THEN
    DBMS_OUTPUT.PUT_LINE('NAME: ' || Var_name);
    DBMS_OUTPUT.PUT_LINE('MARKS: ' || Var_marks);
    DBMS_OUTPUT.PUT_LINE('GRADE: B');

ELSE
    DBMS_OUTPUT.PUT_LINE('NAME: ' || Var_name);
    DBMS_OUTPUT.PUT_LINE('MARKS: ' || Var_marks);
    DBMS_OUTPUT.PUT_LINE('GRADE: C');

END IF;

END;
/
```

Output

```
NAME: Anas
MARKS: 64
GRADE: C
```

1. **Variable Declaration**: The code starts by declaring two variables, `Var_name` and `Var_marks`. These variables are declared to be of the same type as the `NAME` and `MARKS` columns of the `Boys` table.

2. **SELECT INTO Statement**: This statement is used to fetch data from the database and store it directly into variables. The `SELECT NAME, MARKS INTO Var_name, Var_marks FROM Boys WHERE Boys.ID = 3;` statement fetches the `NAME` and `MARKS` of the boy whose `ID` is `3` from the `Boys` table and stores them into the `Var_name` and `Var_marks` variables respectively.
3. **IF-ELSIF-ELSE Statement**: This is a conditional statement that checks the value of `Var_marks` and executes different blocks of code depending on the result:
    - If `Var_marks` is greater than `80`, it prints the boy's name, and marks, and assigns him a grade of 'A'.
    - If `Var_marks` is not greater than `80` but is greater than `70`, it prints the boy's name, and marks, and assigns him a grade of 'B'.
    - If `Var_marks` is not greater than either `80` or `70`, it prints the boy's name, and marks, and assigns him a grade of 'C'.
4. **DBMS_OUTPUT.PUT_LINE**: This is a procedure used to display output on the screen. Depending on the value of `Var_marks`, it will print the boy's name, marks, and grade.
5. **END; /**: This marks the end of the PL/SQL block. The `/` on a line by itself tells SQL*Plus to execute the PL/SQL block.

So, in summary, this code fetches the name and marks of the boy with ID 3 from the `Boys` table, assigns a grade based on the marks, and prints the name, marks, and grade.

PL/SQL Named Functions

Here we are going to discuss using two tables,

First Table is Order Table

order_id	Client_id	Date
1	101	2022-01-01
2	102	2022-01-02
3	103	2022-01-03

Second Table is Order_Item table

order_id	Name	Quantity	PPU
1	item_1	10	5.99
2	item_2	5	12.99

Now we want to build a function that will take a date as input and returns the total sales income of that date. If I input 2022-01-01 as the date it will return,

$$10 \times 5.99 = 59.9$$

But if you look carefully we need to join tables here to perform the operation.

```
CREATE OR REPLACE FUNCTION get_total_earnings(date_in IN DATE)
RETURN NUMBER
IS
    total_earnings NUMBER := 0;
BEGIN
    SELECT SUM(oi.quantity * oi.PPU)
    INTO total_earnings
    FROM orders o
    JOIN order_items oi ON o.id = oi.id
    WHERE TRUNC(o.order_date) = TRUNC(date_in);

    RETURN total_earnings;
END;
/
```

This function will be stored in oracle database whenever we call the function it will do the job on these tables. We can create an anonymous block to call the function and see the result.

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Total earnings for 2022-01-01: ' ||
get_total_earnings(TO_DATE('2022-01-01', 'YYYY-MM-DD')));
End;
/
```

It consists of two parts: a function definition and a block of code that calls the function. Here's what each part does:

1. **Function Definition**: The `CREATE OR REPLACE FUNCTION get_total_earnings(date_in IN DATE) RETURN NUMBER` statement defines a function named `get_total_earnings` that takes one parameter, `date_in`, of type `DATE`, and returns a `NUMBER`.
   Inside the function, a variable `total_earnings` is declared and initialized to `0`. Then, a `SELECT` statement is used to calculate the total earnings for the given date by summing up the product of `quantity` and `PPU` (Price Per Unit) from the `order_items` table for all orders placed on that date. The result is stored in the

`total_earnings` variable.
The `RETURN` statement then returns the value of `total_earnings`.

2. **Function Call**: The `BEGIN...END;` block is used to call the function and display the result. The `DBMS_OUTPUT.PUT_LINE` procedure is used to print the total earnings for the date '2022-01-01'. The `TO_DATE` function is used to convert the string '2022-01-01' to a `DATE` value, which is then passed to the `get_total_earnings` function.

So, in summary, this code defines a function to calculate the total earnings for a given date and then calls this function to print the total earnings for '2022-01-01'. If you have the `orders` and `order_items` tables populated with data, it will give you the total earnings for that date.

Conclusion

In conclusion, Oracle PL/SQL is a powerful and flexible language that extends SQL by adding procedural capabilities, making it an invaluable tool for managing and manipulating data in Oracle databases. This article has provided an overview of key PL/SQL concepts such as anonymous blocks, IF statements, anchored declarations, and named functions.

Anonymous blocks allow us to execute PL/SQL code directly in a SQL environment, providing a quick and easy way to perform complex operations. IF statements introduce conditional logic into our PL/SQL programs, enabling us to control the flow of execution based on specific conditions.

Anchored declarations help us create robust and maintainable code by ensuring that our variables always have the correct data type, even if the database schema changes. Named functions, on the other hand, allow us to encapsulate reusable code and improve the readability and maintainability of our programs.

By understanding and applying these concepts, you can write more efficient, reliable, and performant PL/SQL code. Whether you're a beginner just starting out with PL/SQL or an experienced developer looking to brush up on your skills, I hope this article has been a valuable resource. Happy coding!