Oracle Stored Procedures and Functions

What can you do with PL/SQL?

- Allows sophisticated data processing
- Build complex business logic in a modular fashion
- Use over and over
- Execute rapidly – little network traffic
  - Stored procedures
  - Functions
  - Triggers

Stored Procedures

- Defined set of actions written using PL/SQL
- When called, the procedure performs actions
- Can be called directly from other blocks
- Two parts
  - Procedure specification or header
  - Procedure body

PROCEDURES

- A procedure is a module performing one or more actions; it does not need to return any values.
- The syntax for creating a procedure is as follows:

```
CREATE OR REPLACE PROCEDURE name
[(parameter[, parameter, ...])]
AS
[local declarations]
BEGIN
executable statements
[EXCEPTION
exception handlers]
END [name];
```

PROCEDURES

- A procedure may have 0 to many parameters.
- Every procedure has two parts:
  1. The header portion, which comes before AS (sometimes you will see IS—they are interchangeable), keyword (this contains the procedure name and the parameter list),
  2. The body, which is everything after the IS keyword.
- The word REPLACE is optional.
- When the word REPLACE is not used in the header of the procedure, in order to change the code in the procedure, it must be dropped first and then re-created.

Example: Procedure

```
CREATE OR REPLACE PROCEDURE hello IS
   Greetings VARCHAR(20);
BEGIN
  Greetings:= 'Hello World';
  DBMS_OUTPUT.PUT_LINE(greetings);
END hello;
```

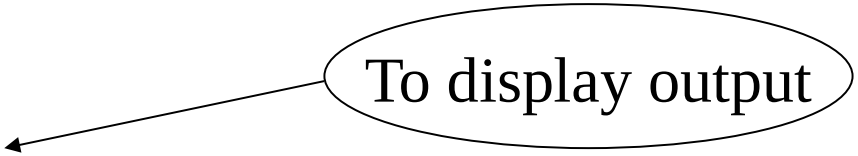
Example: Procedure

```
CREATE OR REPLACE PROCEDURE Discount
AS
CURSOR c_group_discount
IS
SELECT distinct s.course_no, c.description
FROM section s,enrollment e,course c
WHERE s.section_id = e.section_id
AND c.course_no = s.course_no
GROUP BY s.course_no, c.description,
e.section_id, s.section_id
HAVING COUNT(*) >=8;
BEGIN
FOR r_group_discount IN c_group_discount
LOOP
UPDATE course
SET cost = cost * .95
WHERE course_no = r_group_discount.course_no;
DBMS_OUTPUT.PUT_LINE
('A 5% discount has been given to'||
r_group_discount.course_no||' '||
r_group_discount.description
);
END LOOP;
END;
```


Calling a Procedure

In order to execute a procedure in SQL*Plus use the following syntax:

EXECUTE Procedure_name




```
set serveroutput on size 4000
EXECUTE hello;
```

Another way to execute it is from another PL/SQL block:

```
BEGIN
      hello;
END;
```

Arguments

- A value can be passed to a procedure when it is called (input)
- Must specify datatype
- Example (not actually a procedure):

```
increase_salary_find_tax(
increase_percent IN      NUMBER:=7,
sal              IN OUT  NUMBER,
tax              OUT     NUMBER)
```

- IN means the procedure can read an incoming value from that parameter when the procedure is called
- OUT means the procedure can use that parameter to send a value back to what called it
- increase_percent has a default value of 7

Arguments

- Following is a procedure with arguments:

```
CREATE OR REPLACE PROCEDURE
  increase (oldprice NUMBER, percent
  NUMBER := 5, newprice OUT NUMBER)
IS
BEGIN

  newprice:=oldprice+oldprice*percent/10
  0;
END increase;
```

Calling a Procedure with Arguments

```
DECLARE
  price_increase NUMBER(6,2) := 20;
  newp NUMBER(6,2) := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Current price: '||
  price_increase);

  increase(oldprice=>price_increase,newprice=>newp
  );
  DBMS_OUTPUT.PUT_LINE
                ('Price after increase: '|| newp);
END;
```

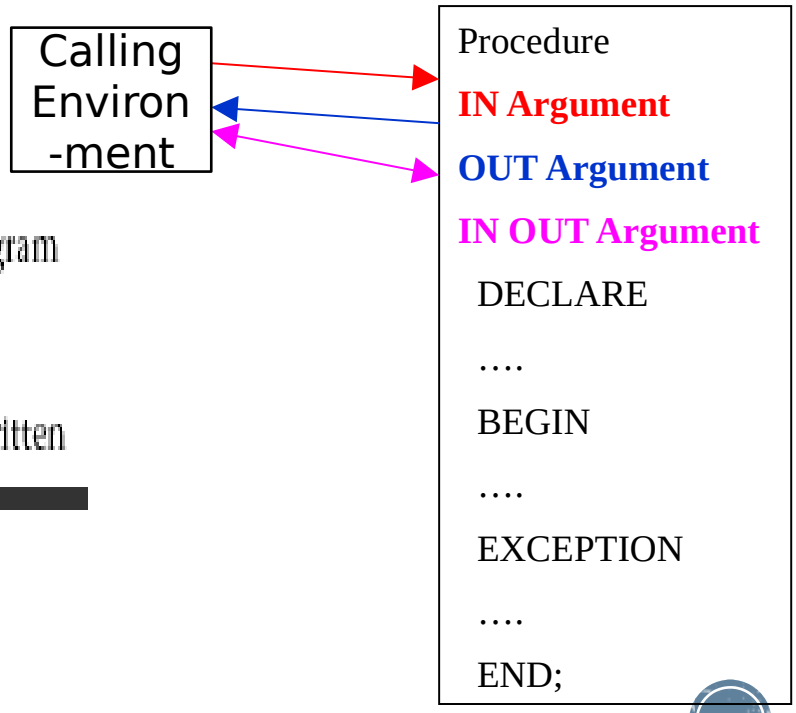
We should see a new price of 21

PARAMETERS

- Parameters are the means to pass values to and from the calling environment to the server.
- These are the values that will be processed or returned via the execution of the procedure.
- There are three types of parameters:
- IN, OUT, and IN OUT.
- Modes specify whether the parameter passed is read in or a receptacle for what comes out.

Types of Parameters

Mode	Description	Usage
IN	Passes a value into the program	Read only value Constants, literals, expressions Cannot be changed within program Default mode
OUT	Passes a value back from the program	Write only value Cannot assign default values Has to be a variable Value assigned only if the program is successful
IN OUT	Passes values in and also send values back	Has to be a variable Value will be read and then written



FORMAL AND ACTUAL PARAMETERS

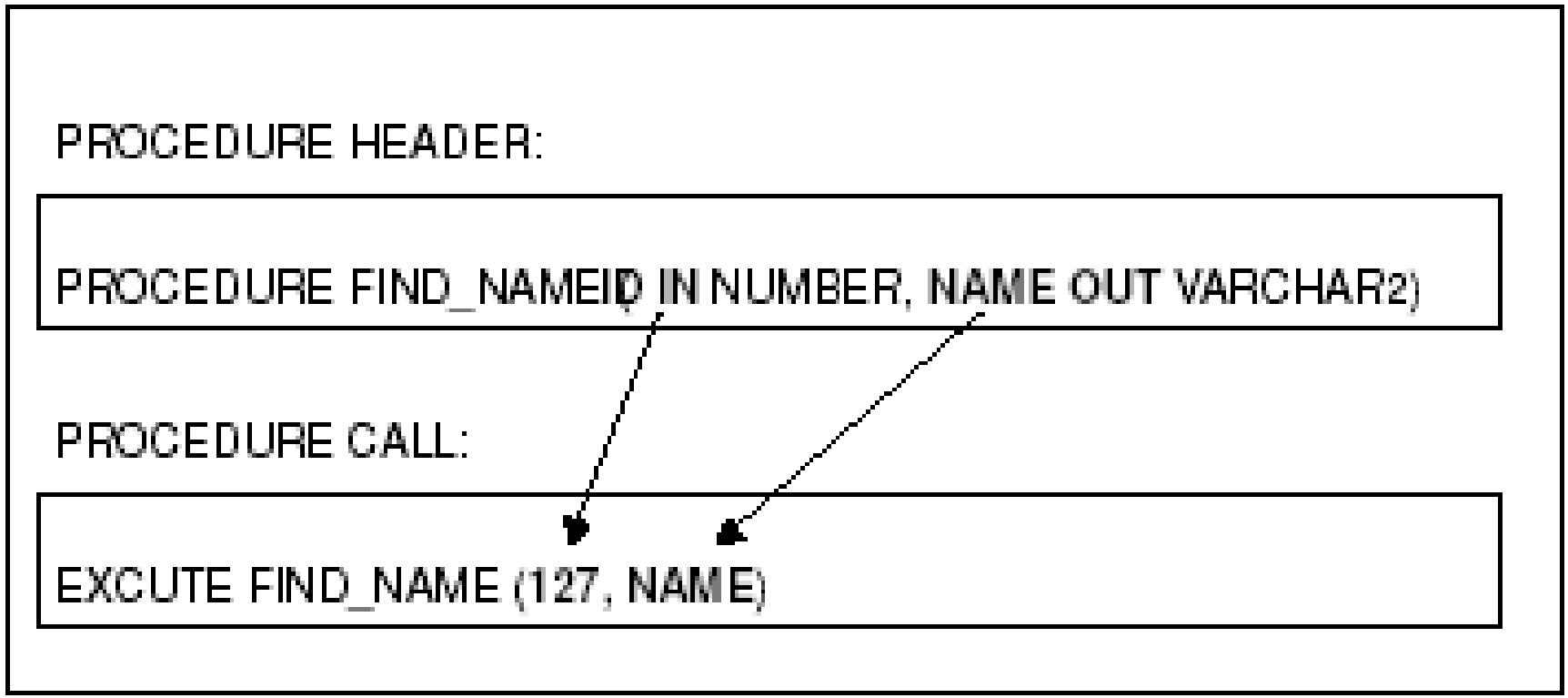
- *Formal parameters* are the names specified within parentheses as part of the header of a module.
- *Actual parameters* are the values—expressions specified within parentheses as a parameter list—when a call is made to the module.
- The formal parameter and the related actual parameter must be of the same or compatible data types.

MATCHING ACTUAL AND FORMAL PARAMETERS

- Two methods can be used to match actual and formal parameters: positional notation and named notation.
- *Positional notation* is simply association by position: The order of the parameters used when executing the procedure matches the order in the procedure's header exactly.
- *Named notation* is explicit association using the symbol =>
  - `Syntax: formal_parameter_name => argument_value`

- In named notation, the order does not matter.
- If you mix notation, list positional notation before named notation.

MATCHING ACTUAL AND FORMAL PARAMETERS



FUNCTIONS

- Functions are a type of stored code and are very similar to procedures.
- The significant difference is that a function is a PL/SQL block that *returns* a single value.
- Functions can accept one, many, or no parameters, but a function must have a return clause in the executable section of the function.
- The datatype of the return value must be declared in the header of the function.
- A function is not a stand-alone executable in the way that a procedure is: It must be used in some context. You can think of it as a sentence fragment.
- A function has output that needs to be assigned to a variable, or it can be used in a SELECT statement.

FUNCTIONS

- The syntax for creating a function is as follows:

```
CREATE [OR REPLACE] FUNCTION
function_name
(parameter list)
RETURN datatype
IS
BEGIN
<body>
RETURN (return_value);
END;
```


FUNCTIONS

- The function does not necessarily have to have any parameters, but it must have a RETURN value declared in the header, and it must return values for all the varying possible execution streams.
- The RETURN statement does not have to appear as the last line of the main execution section, and there may be more than one RETURN statement (there should be a RETURN statement for each exception).
- A function may have IN, OUT, or IN OUT parameters. but you rarely see anything except IN parameters.

Example

```
CREATE OR REPLACE FUNCTION show_description
(i_course_no number)
RETURN varchar2
AS
v_description varchar2(50);
BEGIN
SELECT description
INTO v_description
FROM course
WHERE course_no = i_course_no;
RETURN v_description;
EXCEPTION
WHEN NO_DATA_FOUND
THEN
RETURN('The Course is not in the database');
WHEN OTHERS
THEN
RETURN('Error in running show_description');
END;
```


Making Use Of Functions

- **In a anonymous block**

```
SET SERVEROUTPUT ON
DECLARE
v_description VARCHAR2(50);
BEGIN
v_description := show_description(&sv_cnumber);
DBMS_OUTPUT.PUT_LINE(v_description);
END;
```

- **In a SQL statement**

```
SELECT course_no, show_description(course_no)
FROM course;
```


Example

```
CREATE OR REPLACE FUNCTION discount
  (amount NUMBER, percent NUMBER:=5)
RETURN NUMBER
IS
BEGIN
  IF (amount>=0) THEN
        return (amount*percent/100);
  ELSE
      return(0);
          END IF;
END discount;
```

The IF-THEN construct allows for error checking

Example : Calling the Function

```
DECLARE
  current_amt NUMBER:=100;
   incorrect_amt NUMBER:=-5;
BEGIN
   DBMS_OUTPUT.PUT_LINE(' Order and Discount');
  DBMS_OUTPUT.PUT_LINE(current_amt || '       '||
  discount(current_amt));
  DBMS_OUTPUT.PUT_LINE(incorrect_amt||'       '||
  discount(incorrect_amt));
END;
```

Example

- Write a PL/SQL function that accepts price and onhand values, checks to be sure they are both greater than 0 and multiplies them together. If they are less than 0 return 0.

```
CREATE OR REPLACE FUNCTION total_amount (price
   NUMBER, onhand NUMBER)
RETURN NUMBER IS
BEGIN
   IF (price>0 AND onhand>0) THEN
         return (price*onhand);
   ELSE
        return(0);
   END IF;
END total_amount;
```