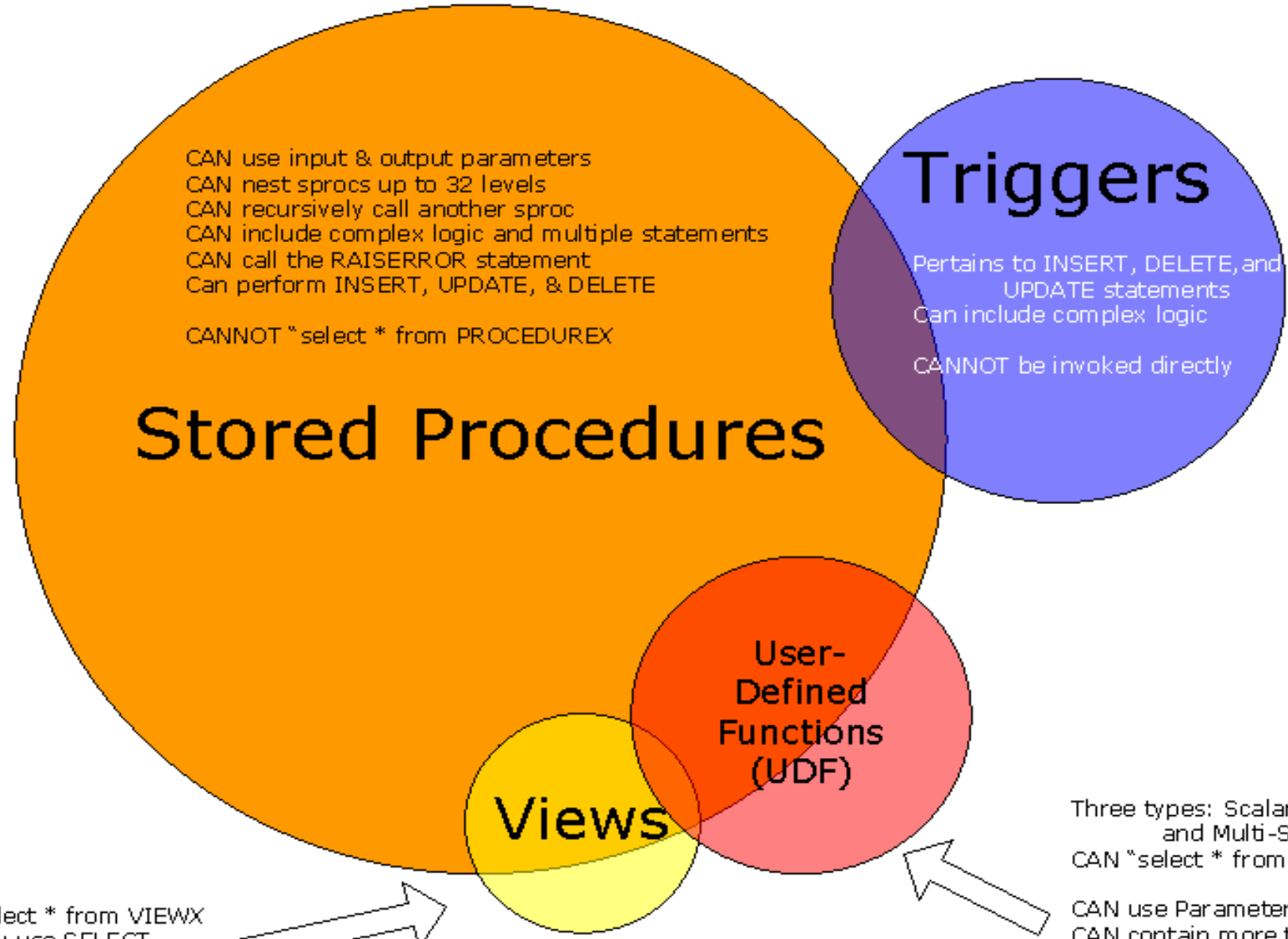
chapter 7:

Functions & Stored Procedures

Dr. Linda Mahmoudi



CAN "select * from VIEWX
CAN only use SELECT

CANNOT use Parameters
CANNOT use more than one statement unless SELECT with UNION is used.

Three types: Scalar UDF, Inline Table-Valued UDF, and Multi-Statement Table-Valued UDF
CAN "select * from FUNCTIONX

CAN use Parameters
CAN contain more than one statement if a Multi-Statement Table-Valued Function is used

CANNOT be used for INSERT, UPDATE & DELETE
CANNOT use *non-deterministic* functions

Difference between Procedures and Triggers

- Stored Procedures and Triggers within a database are similar constructs.
- They can both perform the same SQL statements.
- The biggest difference between the two is how they are executed.
- A stored procedure has to be executed by a user, while a trigger is executed by the system as the result of an event.
- Events that cause triggers to be activated include data inserts, updates and deletes.
- One drawback to using triggers instead of stored procedures is that they cannot accept parameters.

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

```
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
```

Example

```
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;
```

Example

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

```
EXECUTE Procedure_name
```

```
SQL> EXECUTE Discount
```

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

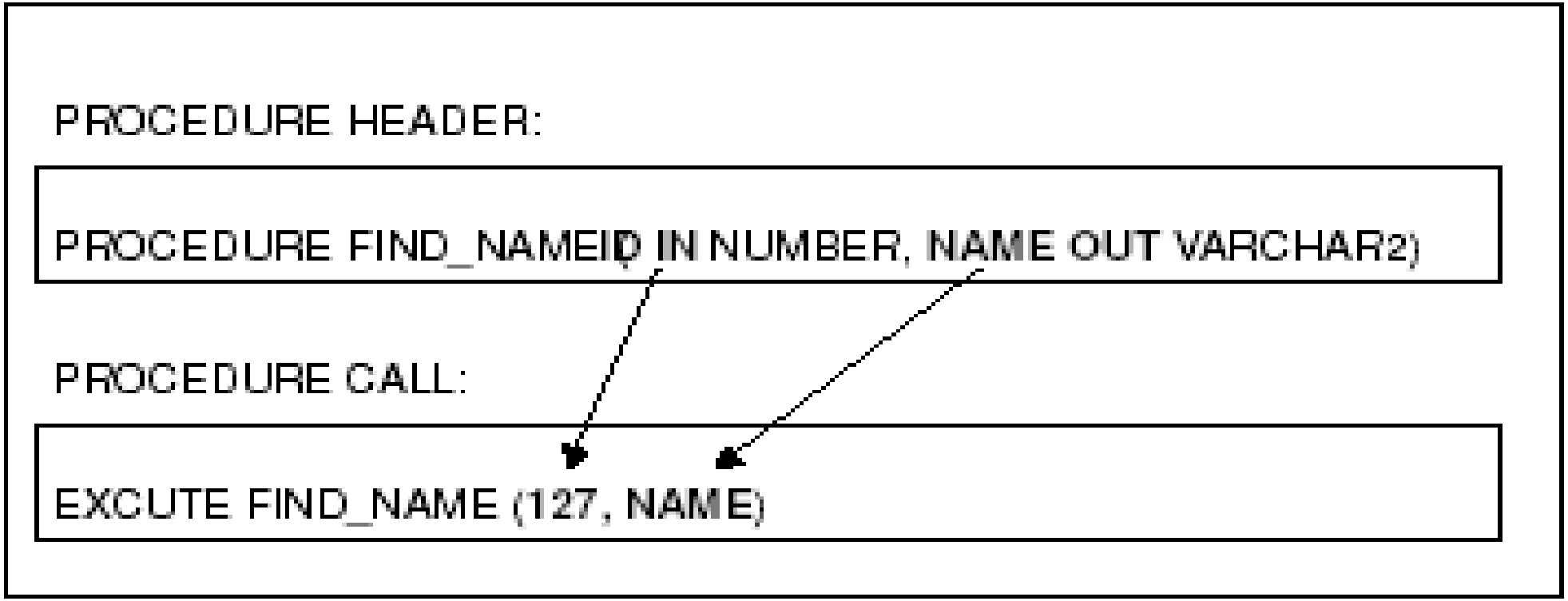
PARAMETERS

- **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



SQL Functions

- User can create functions in SQL Server for saving the SQL statements permanently in the system.
- The functions are calls as User Defined Functions (UDF). The UDF is the database object that contains a set of SQL statements.
- The function accepts input as parameters, performs actions and the result set is returned as action. The return value can be a result set or a single value.

SQL Functions

- The user defined functions has limited functionality as compared to the stored procedures. When user does not require any permanent changes to the database objects, the user defined functions are implemented. The database modifications are not possible through the functions.
- Depending on the use, the user defined functions are categorized as scalar functions and table valued functions.

SQL Functions

- 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.
- 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.

SQL 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;
```

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;
```